ISSN 0280–5316
ISRN LUTFD2/TFRT--5502--SE

Bridging over FPGA Computer Tools with VHDL

Martin Hägerdal

Department of Automatic Control
Lund Institute of Technology
March 1994

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
March 1994

Document Number
ISRN LUTFD2/TFRT--5502--SE

Author(s)
Martin Hägerdal

Supervisor
Walter Keller and Rolf Johansson

Sponsoring organisation

Title and subtitle
Bridging over FPGA Computer Tools with VHDL. (Sammanlänkning av FPGA datorverktyg med VHDL).

Abstract

It has been shown that the graphical interface System Design Station from the software manufacture Mentor Graphics can be used for design of a complete programmable component without having to deal with gate or component levels at all. The graphical code is by the program translated to VHDL and with the language as a medium brought through the different tools to component level. A complete top down design has been done to show the above.

Macrofunctions from the FPGA component manufacture, Altera, have been built into the design. This is done in System Design Station and demonstrates the flexibility of the program. The problems with System Design Station that have come up during the design phase are also described.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280–5316

ISBN

Language
English

Number of pages
104

Recipient's notes

Security classification

Preface

The relationship between the user and the programs described in this Master Thesis is most beautifully described by Shakespeare[1] in 'The Taming of the Shrew'. The user must treat the programs with vast amounts of love and affection mixed with small portions of firmness.

1. The taming of the Shrew, William Shakespeare

1 Introduction

1.1 Background

Up to now the computer tools (Computer Aided Engineering (CAE) tools) for developing hardware have been specific for the components that the developer wanted to use. The hardware designer has had to look at his or her problem and make an estimate before beginning work, which device most appropriately would solve the problem. If the incorrect judgement was made in the beginning of the design phase the designer risked losing a great deal of work.

With the new methods shown in this Master Thesis the developer is saved this, in many cases, anguished, early decision. He or she is now able to solve the problem in the manufacture independent programming language Very high speed circuit Hardware Description Language (VHDL) [1] and then decide which device is most appropriate for the problem. VHDL is a language for describing electronic circuits. It is defined by a manufacture independent organization IEEE. Many different manufactures produce software where VHDL is the programming language.

The Field Programmable Gate Array (FPGA) technology is of interest for numerous reasons. It makes it possible to save space, the printed circuit boards can be made smaller while more functions can be integrated in one circuit. It raises the Mean Time Between Failure (MTBF) because it cuts the number of connections between circuits and decreases the number of devices. It lowers the numbers of wires used, which lowers costs. It lowers the check cost and makes quality assurance easier.

Time is a shortage in modern industrial development and manufacturing. With the FPGA technology time will be saved in the development phase. Time will also be saved in the inevitable correction phase following the development.

1. IEEE Std. 1076-1987, IEEE

2 Problem Statement

Every FPGA producer has their own program for designing and mapping their component. Without VHDL as a bridge the user must first choose component and then design. With VHDL he can begin with the design without having decided which component is most suitable.

The problem is to show that programs from different producers, with VHDL as a bridge between them, can be used to produce a commercial FPGA. The choice of programs is made so that the problem can be analyzed, solved and mapped to the component by the programs. The bridge between the programs is VHDL.

2.1 Choice of programs

Programs from two different companies are chosen, Mentor Graphics and Altera. Mentor Graphics is chosen because their products are used at Siemens UB Med before work for this Master Thesis was begun. The work done for this Master Thesis will be integrated with e.g. the schematics already produced in Design Architect by Mentor Graphics.

Further, Mentor Graphic has just begun marketing a new product for analysis and synthesis of hardware problems, System Design Station. System Design Station is a graphical interface for VHDL programming. It is of interest to find out what this can give and what constraints it sets.

Altera is chosen because they have the biggest component on the market at the moment and the work that is to follow this Master Thesis at Siemens UB Med, needs a component of the size offered by Altera.

2.2 The hardware problem

In order to prove the above, the following hardware problem has been worked with. A problem concerning the new Somatom family was choosen. A complete top-down hardware design of this problem has been carried out.

The problem arises from the enhancement of the Somatom. The previous model of the Somatom is able to take X-rays in one of two fashions. A cross section picture off the patient or an overview off the patient..

The new model is able to take X-rays in the same fashion as the previous model but also able to make X-rays in a spiral mode. This involves mixing data from the two planes generated in the earlier model. The data that has been worked with in this Master Thesis is the positioning data. The old system contained two different sets of signals for the positioning, Ap and Apr for the information of where on the circle in the x-y plane the rotating part is and the Htp and Htpr for telling where the patient is positioned along the z-axis.

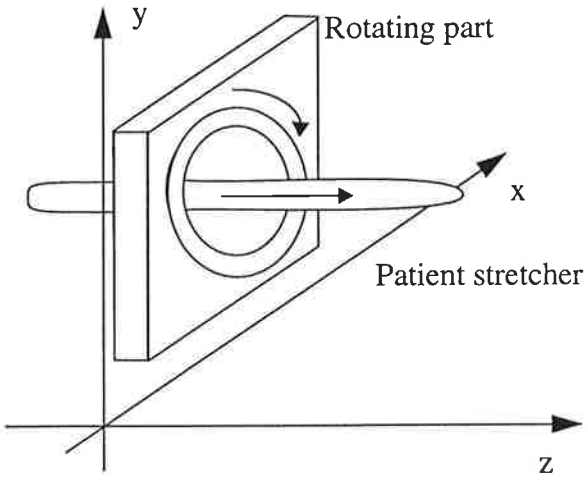


Figure 1: Space description of a Somatom.

The problem is mixing the four different signals from two physical lines to one physical line see Figure 2. The inputs have to be decoded to identify whether it is a long or short signal and then coded in the mux process in order to keep the rising edge stable, with a fixed dealay and to separate the four signals in the demuxing.

It is the wish of the development department that this should be done in hardware because of speed and programmed into a programmable logic device for flexibility and financial reasons. A number of error signals are also sought to be implemented.

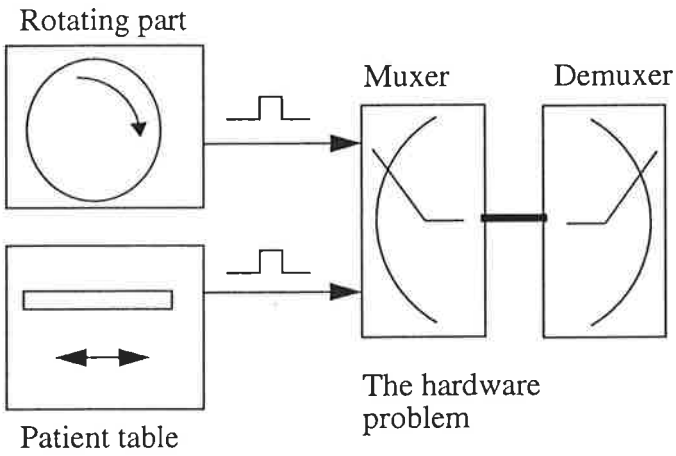


Figure 2: Principal sketch of data transport.

2.2.1 Ap and Apr

An inverted cogwheel is mounted on the rotating part of the Somatom. The teeth of the cogwheel obstruct a light beam. The obstructed light beam generates the Ap and Apr signals. The difference between the two kinds of signals is the duration. The Apr is twice as long as the Ap.

The Ap signal tells the logic the position of the rotating part of the Somatom. The Apr is a reset signal that is set once every rotation of the rotating part of the Somatom.

The Ap signal is specified to appear every 700 us - 1,3 ms. The specified times for the Ap and Apr signals are shown in Table 1.

Signal	Time high
Ap	200 us
Apr	400 us

Table 1: Ap and Apr signals time specifications

2.2.2 Htp and Htpr

The Htp signal is the signal that tells the system the position of the patient table. The Htp signal is not as frequent as the Ap signal. The Htpr is the reset signal.

It appears at an average off a one to ten ratio to the Ap signal. The specified times for the Htp and Htpr signals are shown in see Table 2.

Signal	Time high
Htp	200 us
Htpr	400 us

Table 2: Htp and Htpr signals specifications

2.2.3 Mode

As described in paragraph 2.1 a new mode is added in the enhancement worked with in this Master Thesis. It is the wish of the developer that the old modes should be workable parallel to the new one implemented. Because of the nature of the design only two choices are needed, Tomo / Topo mode or spiral mode.

2.2.4 Error signals

In the specification a desire for a numerous of error signals is expressed. They should all be available through a flip-flop memory function. The output should be activated when an error occurs. It should be able to reset from the outside and not be activated again until a new error occurs.

Rot_kurz is the error message for a rot signal that is to short.

Rot_lang is the error message for a rot signal that is to long

Phs_kurz is the error message for a phs signal that is to short.

Phs_lang is the error message for a phs signal that is to long.

Htp_n_m is activated if a Htp or Htpr signal arrives before the former Htp or Htpr signal has been mixed.

2.2.5 Activity signals

Four activity signals are also implemented. They are activated when either one of the corresponding signals are available on the out port Mux_out. They are also reset from outside the design described here. The four signals are Ap_akt, Apr_akt, Htp_akt, Htpr_akt.

2.2.6 Output

The output should be the four signals mixed together on one physical line. Every cycle is given a time gap of 650 us. In every cycle there is room for one Ap or Apr signal and one Htp or Htpr signal. The protocol is shown below in see Table 3.

Signal	Time high	Time low
Ap	200 us	200 us
Apr	250 us	150 us
Htp	100 us	150 us
Htpr	150 us	100 us

Table 3: The protocol for the output.

3 Methods

Several different programs have been worked with. They all have their special function in the chain from idea to the programmed chip.

It is the ambition of the development department to use SDS as the interface between the idea of the user and the computer. The other programs are used for transforming the data in various ways.

3.1 Programming languages

3.1.1 System Design Station, graphical programming language

The "programming language" in System Design Station is graphical. There are two different graphical ways of programming in System Design Station that differ somewhat from one another, flow diagrams and state machines.

The flow diagrams are called Data Flow Diagrams, see [20]. Their purpose is to be a tool for organizing and analyzing the ideas that will amount to a hardware design. Different containers where different actions or further analyses should take place are defined. Between them lines are drawn with names of signals that should affect or be affected by the different boxes.

The state machines are defined according to the well defined principals found in any standard literature e.g. Taschenbuch Elektrotechnik, VEB Verlag Technik Berlin, 1977[1]. The programmer defines a number of states where the machine is to wait for transition arguments to be set true. Between the states, transition lines are drawn with arguments constructed so, that the machine will change state following this line when the argument is true and the machine is standing in the state where the line begins. In the transition, a command or commands, that are included in the transition line are executed.

3.1.2 Very high speed Hardware Description Language (VHDL), MentorGraphics

VHDL is a standardized programming language for describing electronic circuits[2]. The advantage with using this language is not only speed but it makes the user manufacture independent. The standard is regulated by IEEE.

The resemblance to other programming languages is striking when looking at VHDL for the first time. The program allows many of the functions of a normal Pascal compilator such as e.g. procedures, functions, if-, case- and when statements.

1. p 668, Taschenbuch Elektrotechnik, Band 2 Grundlagen der Informationstechnik.
2. IEEE Std. 1076-1987, IEEE.

The most important difference between VHDL and Pascal is that VHDL allows concurrent[1] programing. In hardware, concurrence is common and this is the reason why VHDL supports concurrence. All statements in VHDL are divided into concurrent statements and sequential statements. Examples of concurrent statements are blocks, processes and concurrent procedure calls and of sequential statements wait, if, case and loop[2].

In VHDL a minimum of two different documents are needed to describe a problem, an entity declaration and an architecture body. They may be stored as two different files or stored together in the same file. The entity declaration "..defines the interface between the design entity and the environment outside of the design entity"[3] and the architect body is where the programmer enters the code. Several different architecture bodies may use the same entity declaration.

Two different versions of VHDL has been worked with, MenthorGraphics and the IEEE standard. They are very similar. The Mentor Graphics System 1076 is based on IEEE Std 1076-1987. The difference lies in that Menthor Graphic has implemented a number of functions and definitions not available in IEEE standard VHDL. Mentor Graphic has choosen to do so because they feel their version of VHDL gives the designer greater flexibility and makes it easier to use the language.

The practical aspect of the two versions apart from learning a few new definitions, is that the VHDL code written in Mentor Graphic VHDL, must be compiled with two different compilers. The one compiler compiles the code for simulation in Menthor Graphics own simulation tool, Quicksim. In Quicksim the programmer is able to simulate the code with all the Menthor Graphics specific definitions. The other compiler compiles the code for synthesis in Menthor Graphics Autologic.

If the programmer behaves correctly this is no problem because all Menthor specialities are taken care of by the Menthor system 1076 compiler.

3.2 System Design Station (SDS), MentorGraphics

The analysis and design parts of the program gives the programmer four tools to work with, Context Diagrams, Data Flow Diagrams, State Machines and VHDL code. The first three tools are graphically implemented and the last one is written as ordinary VHDL programming code. The analysis is done in the Context Diagram and the Data Flow Diagrams and the designs and specifications are specified in State Machines and VHDL code.

System Design Station gives the user a powerful tool in that it allows for the exchange of State Machines against other State Machines but also against other VHDL defined solutions, see the right hand side of Figure 3. The different choices can then be simulated in Quicksim. This is most helpful in the analysis for bugs and in trying out different design alternatives

1. Concurrent = "Existing or acting together or at the same time." The Concise Oxford Dictionary.
2. p 6-4, Mentor Graphics VHDL Reference Manual, Feb. 1993.
3. p 2-6 Mentor Graphics Introduction to VHDL, Feb. 1993.

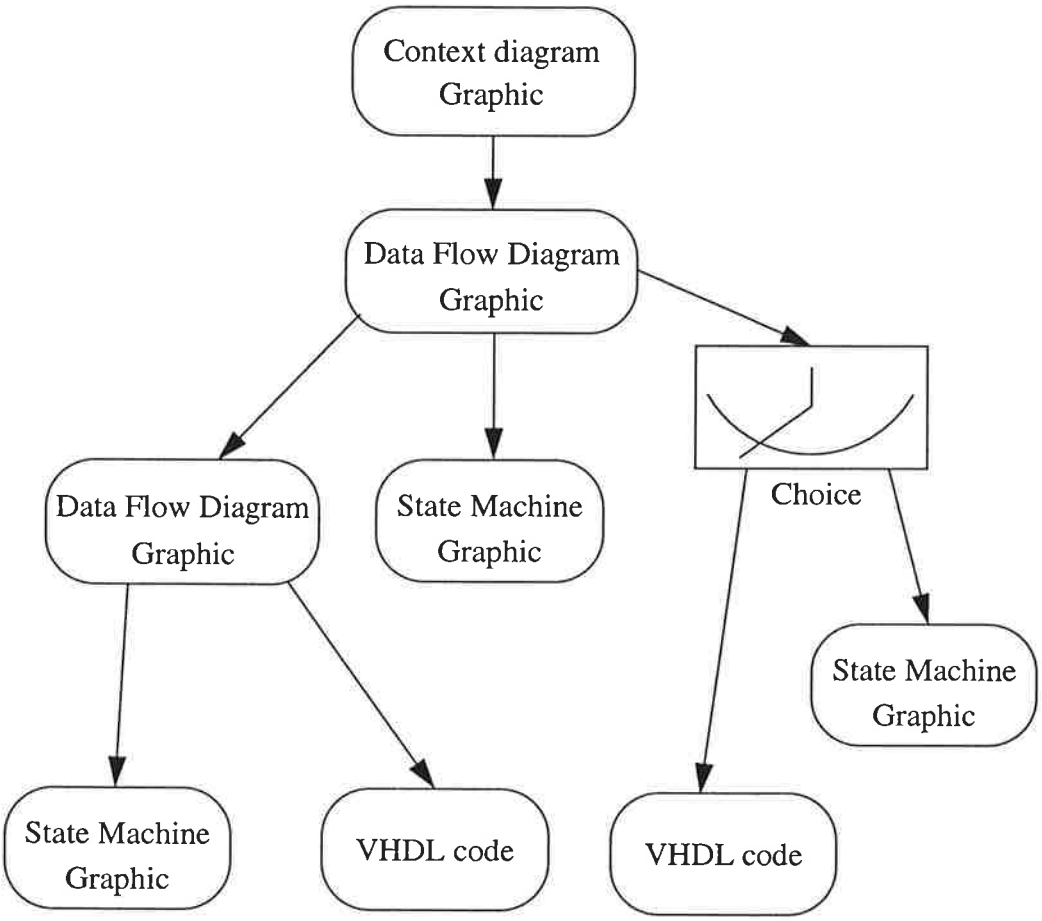


Figure 3: Principal design of a System Design Station program

3.2.1 Context Diagram

The Context diagram, see Figure 4, is the top level in the hierarchy. Here the outside world is defined in different blocks. The design is only represented as one box. Between the boxes representing the outside world and the design, lines are drawn. These represent the inputs and outputs of the design. For each transition a name and type is defined. The name should for readable reasons have something to do with the function as in normal programming. The type is either a standard type from VHDL e.g. bit defined as '0' or '1' or a type defined by the user e.g. an integer that has a constrained range, 0 to 127. Important for readability and simulation is that Mentor Graphic allows enumerated types e.g. "on, off", "know, don't know' etc.

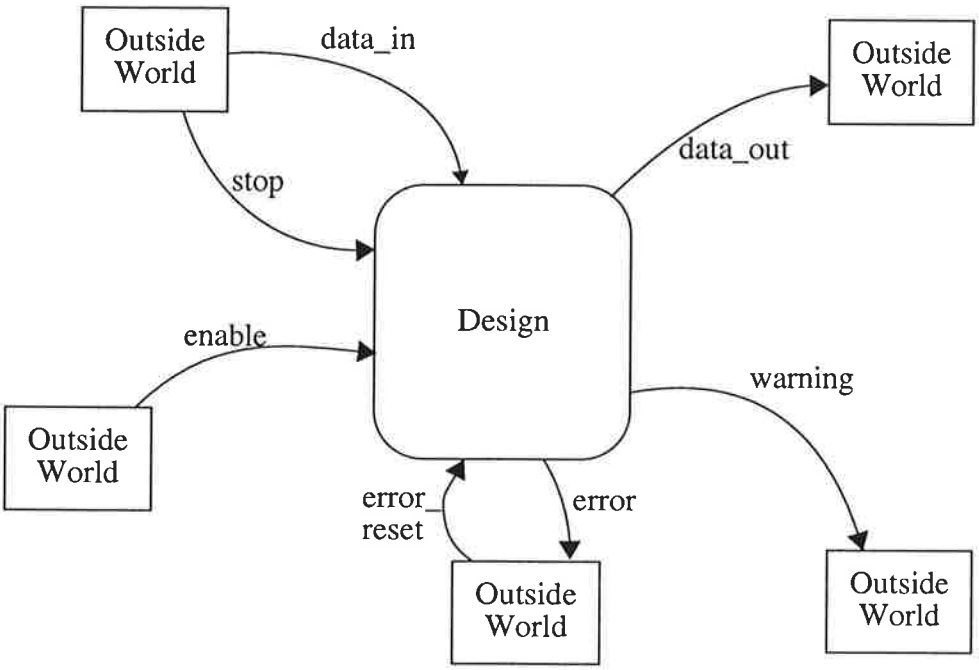


Figure 4: Principal sketch of a Context Diagram.

3.2.2 Data Flow Diagram

Beneath the Context diagram a Data Flow Diagram is defined. In the Data Flow Diagram the user is able to do analysis and organize the data flows within the design, see 3.1.1. In a design there must be one Data Flow Diagram beneath the Context Diagram but below this level numerous Data Flow Diagrams may exist, see Figure 3.

After the Context Diagram has been completely designed, the user asks the program to draw a Data Flow Diagram, see Figure 5. All the signals with their different definitions and origins in the outside world are then transferred to the Data Flow Diagram. This is only done the first time so the more information that is correct the first time, the better for the user.

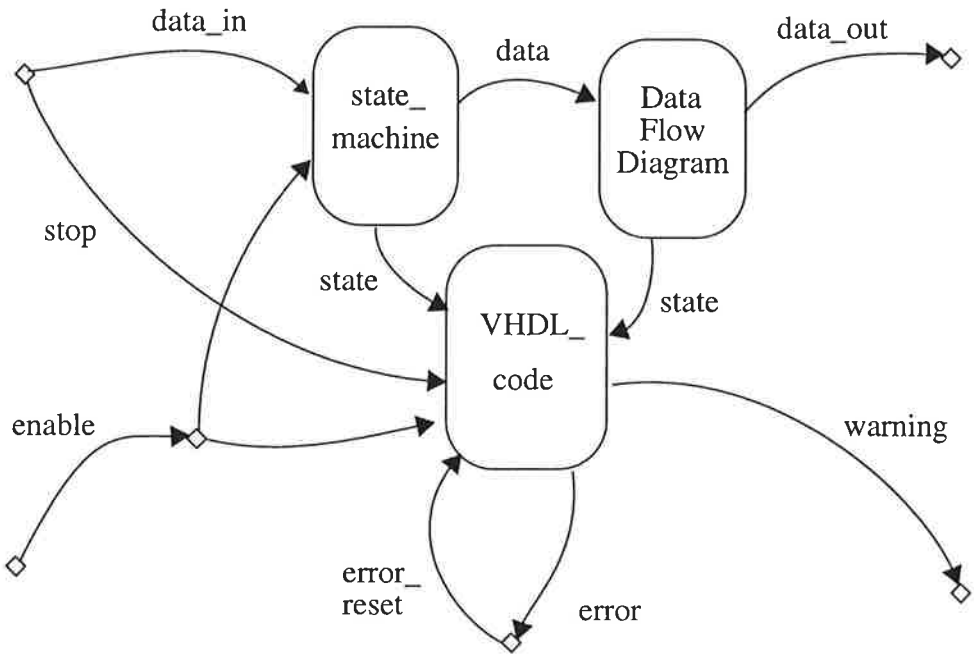


Figure 5: Principal sketch of a Data Flow Diagram.

Within the computer generated Data Flow Diagram the user is free to insert new boxes containing Data Flow Diagrams, state machines or VHDL code. It is also possible to define new data flows within the Data Flow Diagram. There is no problem in redrawing the lines representing the data flows as is shown in the picture above.

3.2.3 State Machine

For defining the functions of the system the user has two possibilities. He can choose between a State Machine and defining VHDL code. Every Data Flow Diagram branch must end with either a State Machine or VHDL code.

The state machine is drawn according to the principals described in 3.1.1 in a special sheet generated by System Design Station. The program generates a sheet with two states and all the signals going into the machine as transition arguments.

3.2.4 Generating VHDL code.

When a function needs to be described by VHDL code, System Design Station generates the entity declaration and an empty architecture body. In the entity, all the signals that are drawn as inputs or outputs to the block for which the VHDL code is written, are declared. No further editing of the entity is needed.

The hollow architecture, see Figure 6, is generated with a process and a minimum of nomenclature. The example shown here is generated with the option "synchronous clock"and "asynchronous reset". Therefore the process contains the sys_clk as argument and the IF statement with sys_clk is generated. The user is even instructed as to where the code should be entered.

```
--
-- Component : tony
--
-- Generated by System Design Station version v8.2_5.3 by wegerer on 19.01.94
--
-- background_clock :: sys_clk rising
-- background_reset :: sys_rst active_low asynchronous_reset
-- Source views :-
-- /user/s/wegerer/ar.sp/ω_messen/ω_messen/data_flow
-- /user/s/wegerer/ar.sp/ω_messen/ω_messen/types/header
-- $MGC_SYS1076_STD/standard/header
--

ARCHITECTURE spec OF tony IS
 BEGIN

 ---------------------------------------------------------------------
 vhdl_tony : PROCESS (
 sys_clk,
 sys_rst

 )
 ---------------------------------------------------------------------
 VARIABLE prop_delay : time := 1 ns;
 BEGIN
 IF ( sys_rst = '0' ) THEN
-- Enter RESET condition code here.
 ELSIF ( sys_clk'EVENT AND sys_clk = '1' AND sys_clk'LAST_VALUE = '0' )
THEN
-- Enter code here.
 END IF;

 END PROCESS vhdl_tony ;
END spec ;
```

Figure 6: Empty architecture body generated by System Design Station.

3.3 Design Architect (DA), MentorGraphics

Design Architect is a complete tool for generating VHDL code for schematics and entire designs. It can substitute System Design Station as an interface, see Figure 7. Design Architect is also a program in which symbols can be edited and VHDL code attached. This makes it possible to design with symbols just in the same way as is traditionally done on paper. These functions have not been used in this Master Thesis and are therefore not described further.

Design Architect has for this Master Thesis been used for editing sheets where the entity and an empty architecture already has been generated by System Design Station, see 3.2.4, or to make smaller changes in code altogether generated by System Design Station. Design Architect is also used for compiling code that has been in any way hand edited or otherwise has had faults within, difficult to detect. The program offers better help functions for debugging than is offered in System Design Station.

3.3.1 Help functions in editing mode

In Design Architect the code is edited in a sheet that symbolizes the file. The user has to enter his entire code or a part of it that can be compiled before it is checked for errors. To compensate for this some help functions are made disposable to help in getting the syntax correct. The functions are gathered in a palette on the side of the interface. They are all chosen by clicking but they are marked with words and not with icons.

The most helpful function is the "Insert Template". It is a library that contains all the statements available in the Mentor version of VHDL. When a statement is chosen the entire syntax for the statement is entered automatically and the programmer only has to enter the arguments. There are also some minor functions for organizing the code e.g. Capitalize and Indent.

3.3.2 Compiling and debugging

When all the code has been entered the programmer chooses compile from a menu and the compiling starts. When errors are detected the compiler stops and gives a message as to why the compilation failed. The user can now click on the error message and choose "highlight" from a menu. The place where the error has been registered by the program is highlighted with a yellow overlain color. The error is hopefully found and the compilation is done over again until the code is errorless.

3.4 Quicksim, MentorGraphics

Quicksim is the simulation tool for Mentor codes. With Quicksim it is possible to simulate single state machines, designs with numerous state machines and Data Flow Diagrams or designs where state machines, Data Flow Diagrams are mixed with VHDL code. It is also possible to mix a System Design Station solution with a complete schematic constructed in Design Architect. With Quicksim it is possible to debug a code for functional malfunctions, as opposed to the compilator where the syntactical indifferences are corrected.

Quicksim is chosen simply by clicking once on the icon of the design that is to be simulated and choosing "simulation" in the popup menu of System Design Station. A menu is shown where the user can set some parameters such as the time resolution of the simulation. It is defaulted to 0,1 ns. A number of functions for simulation and debugging are available.

3.4.1 Simulation

The user begins with choosing "open sheet" from an icon menu. This command opens a window with the VHDL code of the design that is to be simulated. Here the variables that are of interest are choosen by marking them. Variables that are commonly good to choose are e.g. sys_clk and sys_rst and others that influence how the system works.

The user then chooses a command called "Add>Traces" from a popup menu. A new window is opened, see Figure 7. In this it is possible to stimulate signals with the command "Force". The program offers the possibilities to force single values (e.g. mode_sp), a clock (e.g. sys_clk) or multiple values (e.g. rot_in). After the forces have been set the command "run" and a duration is choosen. It is now, as in the picture below, possible to read the results as well as the forces used.

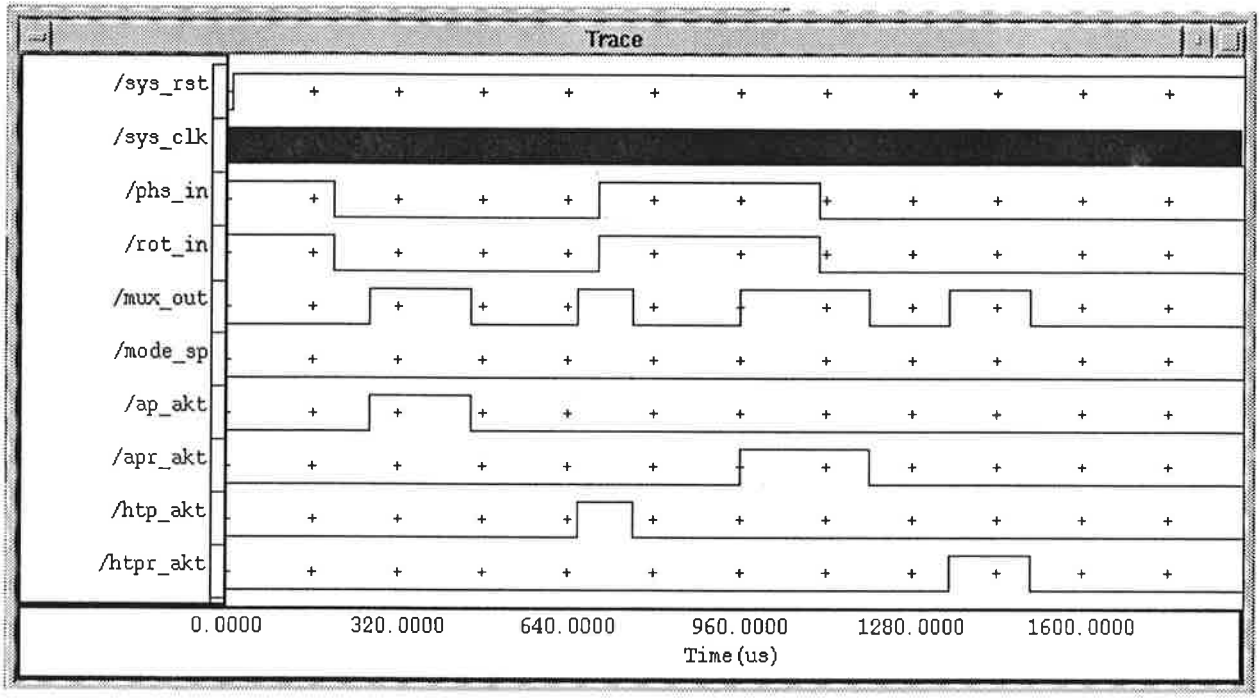


Figure 7: Trace window in Quicksim.

3.4.2 Editing waveform

If a forced waveform needs to be changed it is possible to do so graphically. The program has an icon menu with different functions, see Figure 8. The menu is activated by choosing a signal that is to be edited and clicking on the "Edit Waveform" icon. A duplicate of the signal appears under the name "forces@@/signal name" in the trace window and this can now be edited with the functions available under the other icons shown in the menu, e.g. add edge, change value. When one is satisfied with the editing, the simulation can be run again to see what the changes bring.

The program offers the possibility to save the waveforms and reusing them at a later time. This makes the tool very powerful because the design can be exposed to a set of forces as many times as is needed to sort out the problems.

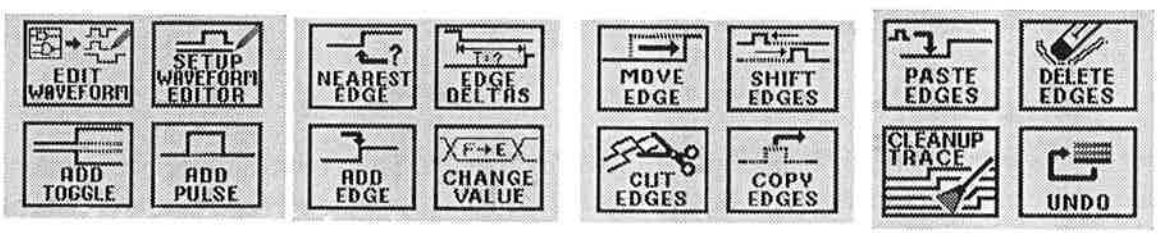


Figure 8: Icons in the dialog box for editing waveforms in Quicksim.

3.4.3 Debugging in Quicksim

One is off course not often so lucky that the design is right from the beginning. Here the program offers another very powerful set of tools for debugging. The code can e.g. be executed step by step, the step iteration button, and evaluated with different tools, see Figure 9. The part of the code that is currently executed will lighten up in red in the sheet window and it is very easy to follow how the code is executed.

The six step- buttons are all used for moving around in the code.They reflect the concurrence of the code in such a way that they are built on the idea of moving around one process at the time. step over a process, step into a process etc. The examine button is used for showing the current value of a variable in the code.

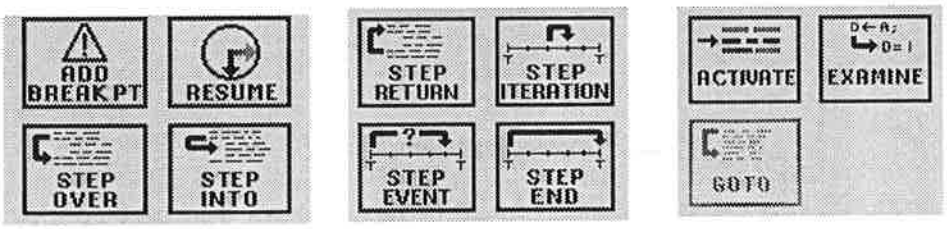


Figure 9: Icons in the dialog box for debugging the VHDL code

There are also other tools for finding out what is happening during execution. They are called List window, see Figure 10, and Monitor window which is similar to List window. The List window generates a list of when the value of a signal has changed and how it has changed. The monitor window shows the value of the variable at the current moment.

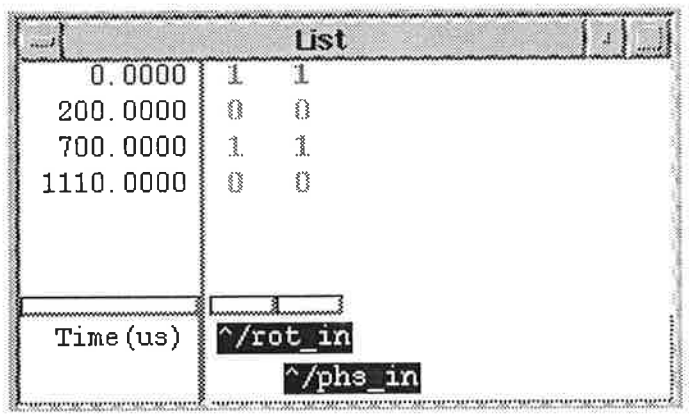


Figure 10: List window in Quicksim.

3.5 System 1076 Compiler, MentorGraphics

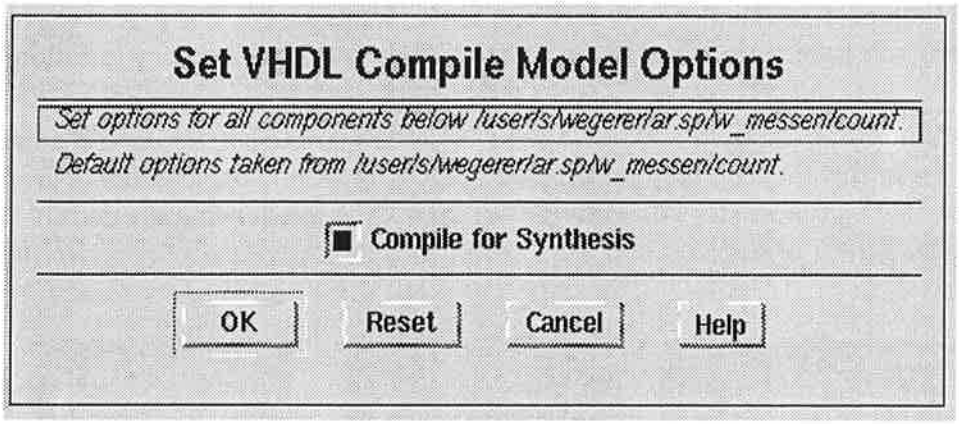


Figure 11: Compile for synthesis dialog box.

As described in section 3.1, two different versions of VHDL has been worked with. For this a subset in the compiler is needed. One option that translates the code so that it can be simulated in Quicksim with the Mentor Graphic specific definitions and one so that the code can be sent on in the chain of programs. For the user this does not mean a great extra effort. The extra work lies in one further compilation and keeping track of the button, Compile for Synthesis, shown in Figure 11, which is a menu that pops up after the button For All in Figure 22 is pressed.

There is a problem with compiling for synthesis. When one wishes to compile an entire hierarchy it is necessary to choose Compile for Synthesis for all components within the hierarchy by clicking on them and entering the menu above. This is a bug that has not been corrected as far as the v8.2_5.3 of System Design Station.

3.6 AutoLogic, MethorGraphics

Autologic has two major functions. The first is transforming a VHDL code, to a gate level description of the problem. This is called a schematic of the problem. The second is to map these schematics to the target technology and optimize according to a set of parameters. The Autologic allows optimization according to area and speed. The output is either an edf file that Maxplus2 uses as an input or a schematic which is a layout of the gates.

3.6.1 Synthesis

The synthesis uses VHDL code that has been compiled for synthesis, see 3.5 as an input. The program does offer the user some possibilities for affecting the synthesize mode but very few settings have been used in this work and are therefore left un commented.

3.6.2 Optimizing

The optimization is controlled by so called recipes, see Figure 23. These recipes are composed by the user combining different options such as (high - low) area and (high - low) speed. One command can be used many times and the output is not the same whether the command is used one time or twice in a row. As above the program offers the user to use different settings for different special situations nut these have not been used and are therefore left un commented.

3.7 Maxplus2, Altera

Maxplus is the program for fitting the edf file, generated by Autologic, to one or more components in the Altera family of components. The program finds the optimal use of the gates in the component and generates an output file with which a component can be programmed.

3.7.1 Architecture

The Altera architecture is principally built around a net of data buses with macro cells laid out in the holes off the net. The data buses, dedicated inputs, (programmable interconnect signals and expander product terms) are wide and therefore they offer great flexibility. Each macro cell, see Figure 12, in the 5000 family has one three input OR gate, an XOR gate and a programmable register that can emulate D, T, JK or SR operations. The architecture of the macro cell [1]allows each cell to have its own output. This is controlled by the Output Enable. For every macro cell it is also possible to choose between a global clock and an individual clock.

The Altera component offers expander product terms. These are "unallocated logic that can be used and shared by all macro cells"[2] in the component. The expander product terms make it possible to reduce the number of macro cells needed for the mapping of a design and the user is therefore able to fit bigger designs into smaller components with the expander product terms.

1. p 153 - 155, Data Book, Altera
2. p 155, Data Book, Altera

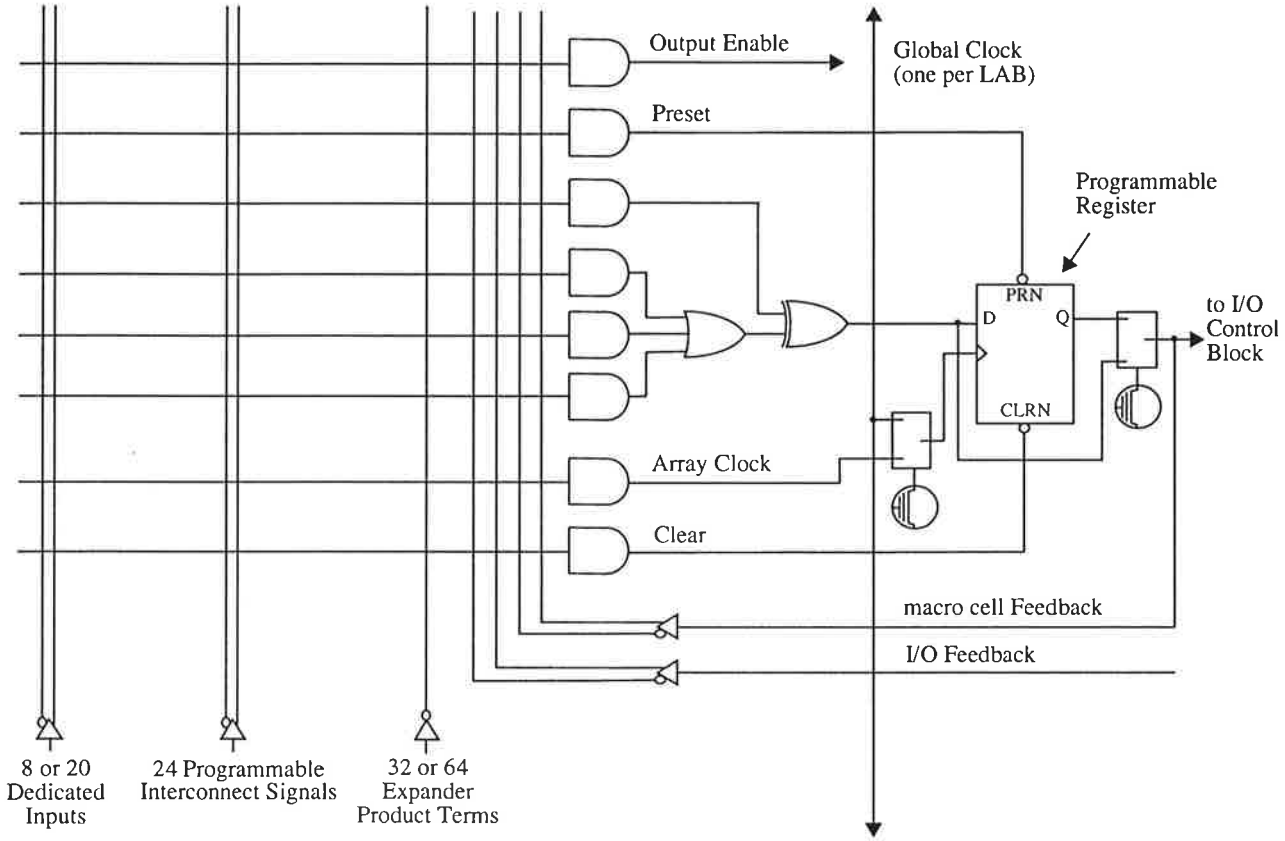


Figure 12: MAX 5000 macro cell.

3.7.2 Optimizing

Maxplus2 uses De Morgan's inversion[1] and other logic synthesis techniques (not specified in the Data Book) for minimizing the amount of gates needed. The Maxplus2 program optimizes the layout according to the number of macro cells needed, routing, propagation time and speed. This is the mapping process.

3.7.3 Usage of the Maxplus2

The Maxplus2 offers the user some possibilities to do settings in the optimization and mapping mode. For the work presented in this paper no settings have been used apart from the default ones. the judgement was made that this would not bring anything to our design. They are therefore left.

In Maxplus2 the only settings used has been the opportunity to choose components. This is done in a menu shown in Figure 24. The rest of the decisions have been left to the program.

1. p 11 Data Book, Altera 1993

4 Solutions

4.1 Design in SDS

The design contains two decoders and one coder. Nine different error signals are implemented, one control signal, two inputs and one output. All signals going to or from the design are defined as bits.

The two decoders are implemented in the containers Rot and Phs, appendix [20], and the coder is implemented in the container Control. The signals X, Y and Z represent internal variables in the containers Rot, Phs and Control. They must be laid out outside their respective containers in this fashion because of a bug in System Design Station.

The ω_messen is constructed for a signal ratio of 1 to 2 for the rot / phs signals. The average ratio between the two is 1 to 10 and a good margin is attained.

Further the signals are not synchronized to one another and one of them needs to be saved in a buffer when the state machine Control is busy, this is done in the container speich.

The Master for the design is only able to act on an error signal when it is set true and must be able to reset the error signal in await for the next one. this is taken care of in the container Tony.

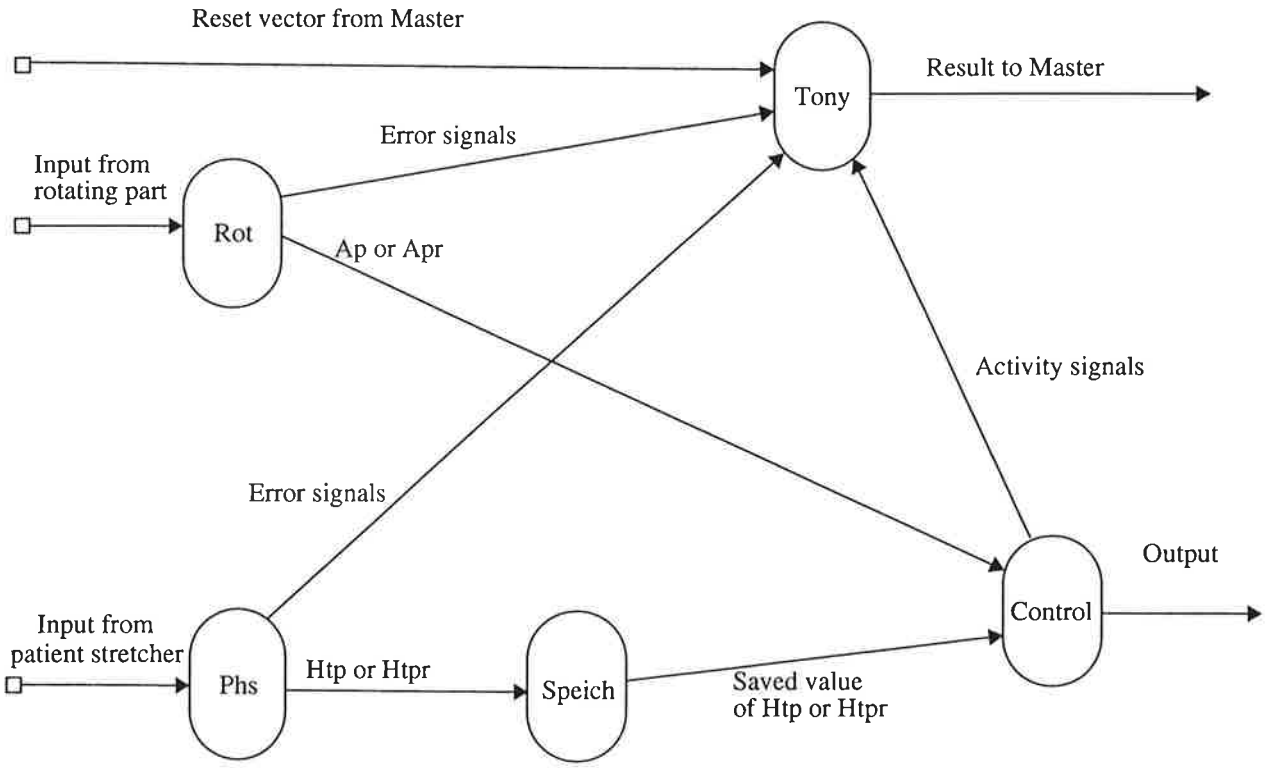


Figure 13: Principal sketch of Data Flow Diagram for ω messen

4.1.1 Rot

Rot is one of the two decoders in the system. It decodes if an Ap or Apr is being sent from the rotating part and sends the information on to the state machine Control. A true Rot_in triggers the system see Figure 14. This makes the state machine jump to the Decision state. If the triggering signal turns out to be only a ripple on the line, shorter than 128 microseconds, the state machine returns to the Start state. If the triggering signal wasn't a ripple, the state machine decides if the triggering signal is an Ap or Apr after 250 microseconds. The tolerance for the signals Ap and Apr is plus minus 50 microseconds. Therefore 250 microseconds is choosen as transfer time, and not 200 exactly, which is the duration for an Ap according to specifications. The design ω messen is designed to receive a new Ap or Apr every 650:th microsecond and therefore the state machine will return to the start state after 650 microseconds.

For a more thorough technical description see appendix [15].

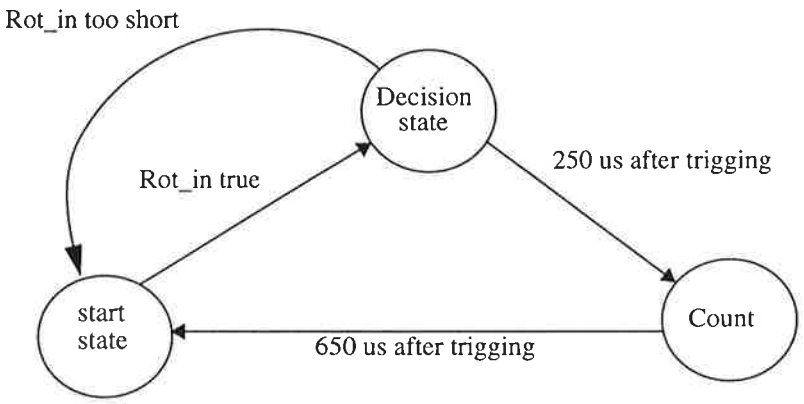


Figure 14: Principal sketch of the Rot state machine

4.1.2 Phs

Phs is the other decoder in the design. It decodes if an Htp or Htpr is being sent from the Patient Handling System and sends the information on to the state machine Control. A true Phs_in triggers the system see Figure 15. This makes the state machine jump to the Decision state. If the triggering signal turns out to be only a ripple on the line, shorter than 128 microseconds, the state machine returns to the Start state. If the triggering signal wasn't a ripple the state machine decides if the triggering signal is an Htp or Htpr after 250 microseconds. The tolerance for the signals Htp and Htpr is plus minus 50 microseconds. Therefore 250 microseconds is choosen as transfer time, and not 200 exactly, which is the duration for an Htp according to specifications. The design ω messen is designed to receive a new Htp or Htpr every 650:th microsecond and therefore the state machine will return to the start state after 650 microseconds.

For a more thorough technical description see appendix [14].

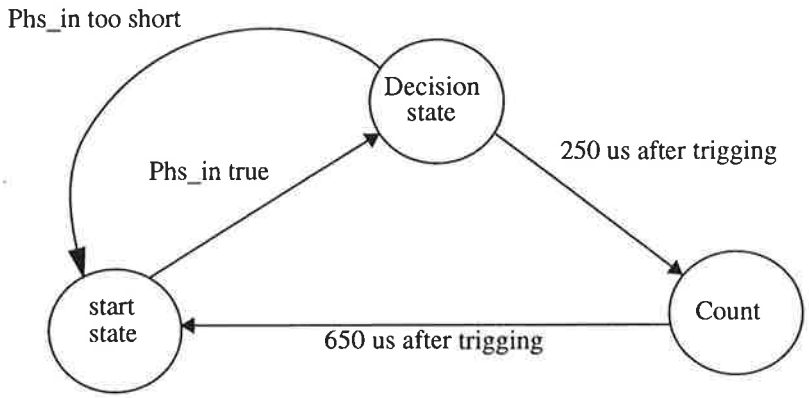


Figure 15: Principal sketch of the Phs state machine.

4.1.3 Speich

Speich is a buffer between the Phs state machine and the Control state machine. A buffer is needed because the two input signals to ω messen, Rot_in and Phs_in, are not synchronized with one another. The buffer is added between the Phs state machine and the Control state machine because the Rot_in signal is given a higher priority than the Phs_in signal and therefore the time delay is given the Phs_in signal.

Speich is drawn as two loops, see Figure 16. They are identical and triggered by a true Htp or a true Htpr. The state machine waits in the Htp state or Htpr state until Reset_s is set true by the state machine Control. This is the buffer function of the state machine. The Htp zw state or Htpr zw state are in-between states. They are only there to guard against having the state machine triggering twice on the same input signal.

For a more thorough technical description see appendix [16].

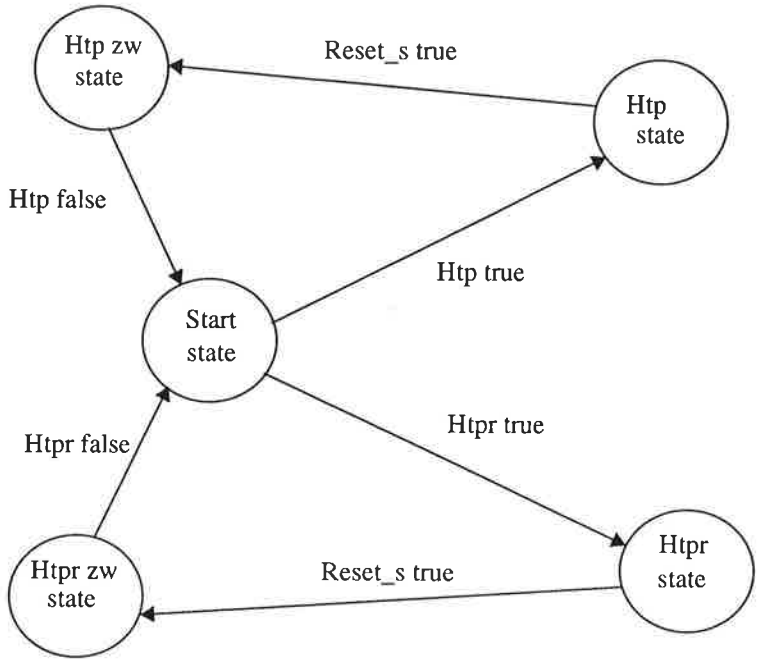


Figure 16: Principal sketch of state machine Speich.

4.1.4 Control

Control is the state machine where the four different signals are mixed together for the multiplexation, onto one physical line. Every time window is 650 microseconds long.The time window is divided into two parts. The first one is 400 microseconds long and the last one is 250 microsecond long, see Figure 17.

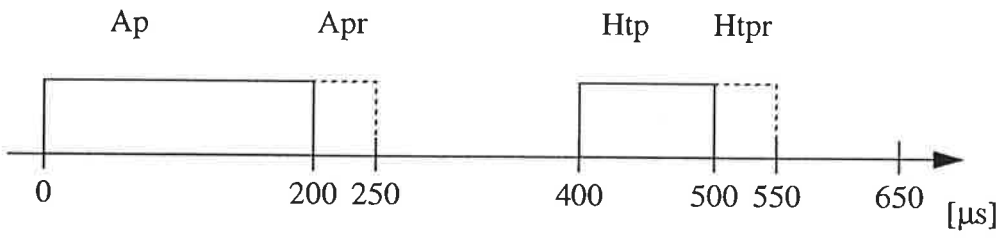


Figure 17: The time window for the muxed signal.

The division is done in this fashion because it is, for decodation purposes, favorable to have the time when the signal is true somewhat as long as the time when the signal is false. This is also the critical time for the entire design. It decides the maximum frequency with which signals can be received and sent but has nothing to do with the lower frequency limit.

The state machine, see Figure 18, is designed to work with only signals generated by the rotating part (Ap and Apr) or only signals generated by the Patient handling system (Htp and Htpr) or to mix them together. The three different modes are controlled by the signal mode_sp. When mode_sp is false the state machine works in spiral mode, i.e. a cycle must begin in either Ap or Apr state, and when it is true the state machine works in either Topogram (horizontal) or Tomogram (vertical) mode, i.e. a cycle may begin in any of the four states Ap, Apr, Htp or Htpr.

The task of the state machine is to code the muxed signal Mux out according to Figure 17. The Mux out signal is therefore set true whenever a transition is made to Ap state, Apr state, Htp state or to Htpr state. After the specified time that mux out is supposed to be true according to specifications, the signal is set false and remains in the state for the specified time that the signal is supposed to be false.

For a more thorough technical description see appendix [13].

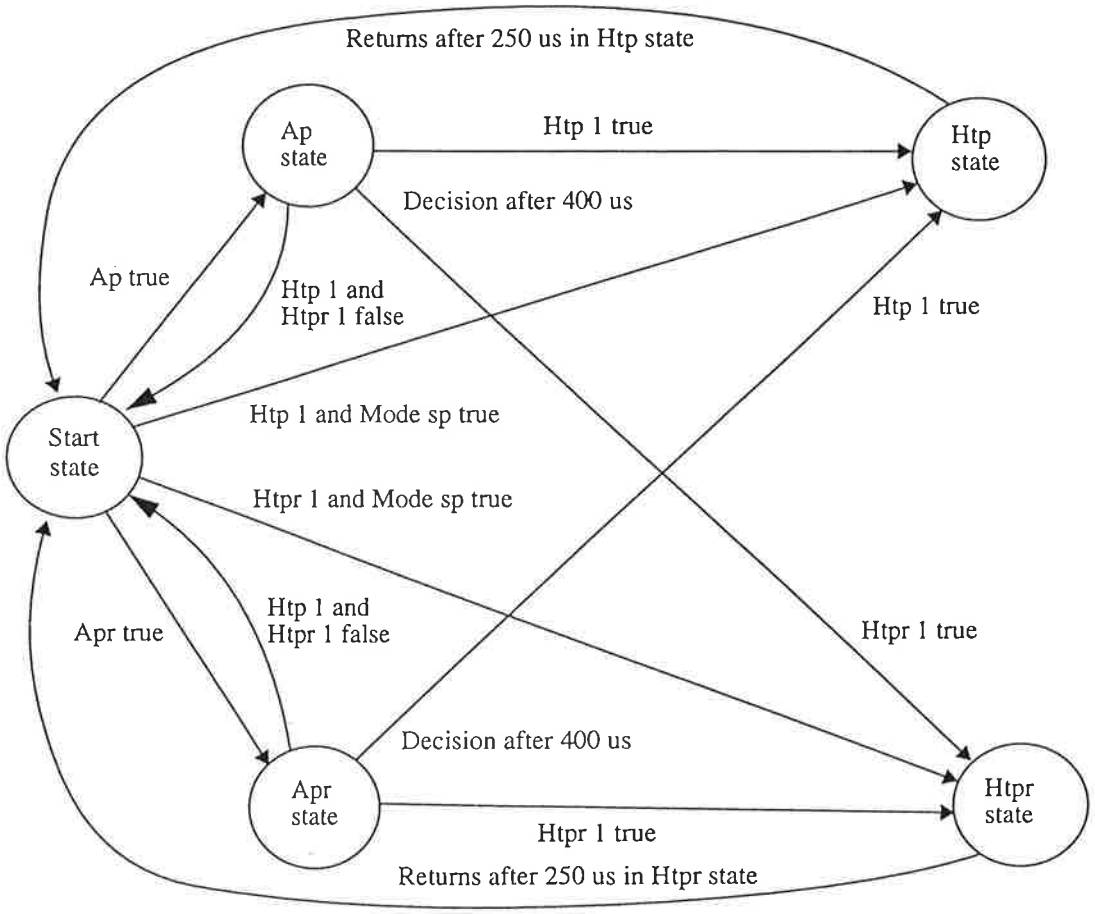


Figure 18: Principal sketch of state machine Control.

4.1.5 Tony

Tony is in difference to the other function blocks not described by a state machine. It is described directly in VHDL. This is a System Design Station feature but causes some unforeseen problems described in section 4.2.

The Tony code makes sure that only one message is given to the Master for every message generated in the design. When the Master has read the message it acknowledges this by resetting the part of Tony that deals with the actual signal.

All the error messages and activity messages are relayed through Tony, they are the error messages Rot_kurz, Rot_lang, Htp_n_m, Phs_lang and Phs_kurz, described in 2.2.4, and Ap_akt, apr_akt, Htp_akt and Htpr_akt, described in 2.2.5.

For a more thorough technical description see appendix [17].

4.2 Translation to VHDL

In System Design Station the user enters ideas predominantly in graphics. These graphics must be translated to, a for the computer understandable code. The graphics must also be tied together and the parts of the solutions described in VHDL code must be tied in. This is all done in System Design Station.

4.2.1 Generating VHDL code from state machines

The user has the possibility of generating all code at once, a part of the hierarchical design or of only generating code for one state machine or data flow diagram at a time. In Figure 19 the dialog box for generating code for the state machine Rot is shown.

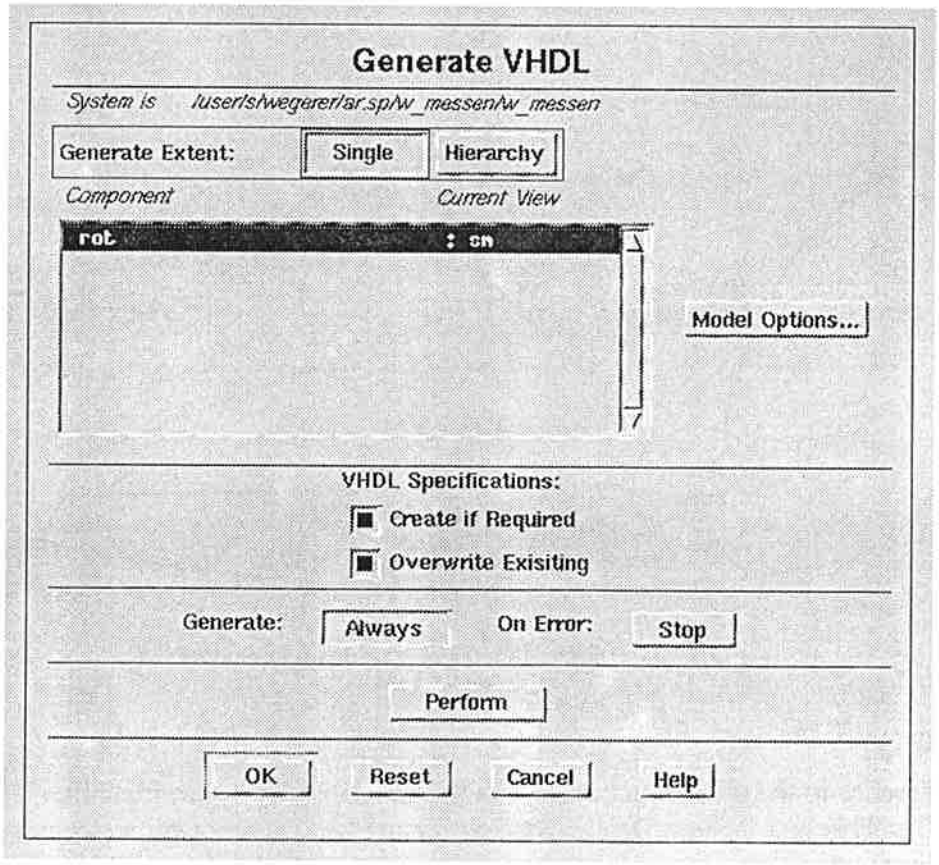


Figure 19: Dialog box for generating VHDL code.

The state machine, see Figure 18, is designed to work with only signals generated by the rotating part (Ap and Apr) or only signals generated by the Patient handling system (Htp and Htpr) or to mix them together. The three different modes are controlled by the signal mode_sp. When mode_sp is false the state machine works in spiral mode, i.e. a cycle must begin in either Ap or Apr state, and when it is true the state machine works in either Topogram (horizontal) or Tomogram (vertical) mode, i.e. a cycle may begin in any of the four states Ap, Apr, Htp or Htpr.

The task of the state machine is to code the muxed signal Mux out according to Figure 17. The Mux out signal is therefore set true whenever a transition is made to Ap state, Apr state, Htp state or to Htpr state. After the specified time that mux out is supposed to be true according to specifications, the signal is set false and remains in the state for the specified time that the signal is supposed to be false.

For a more thorough technical description see appendix [13].

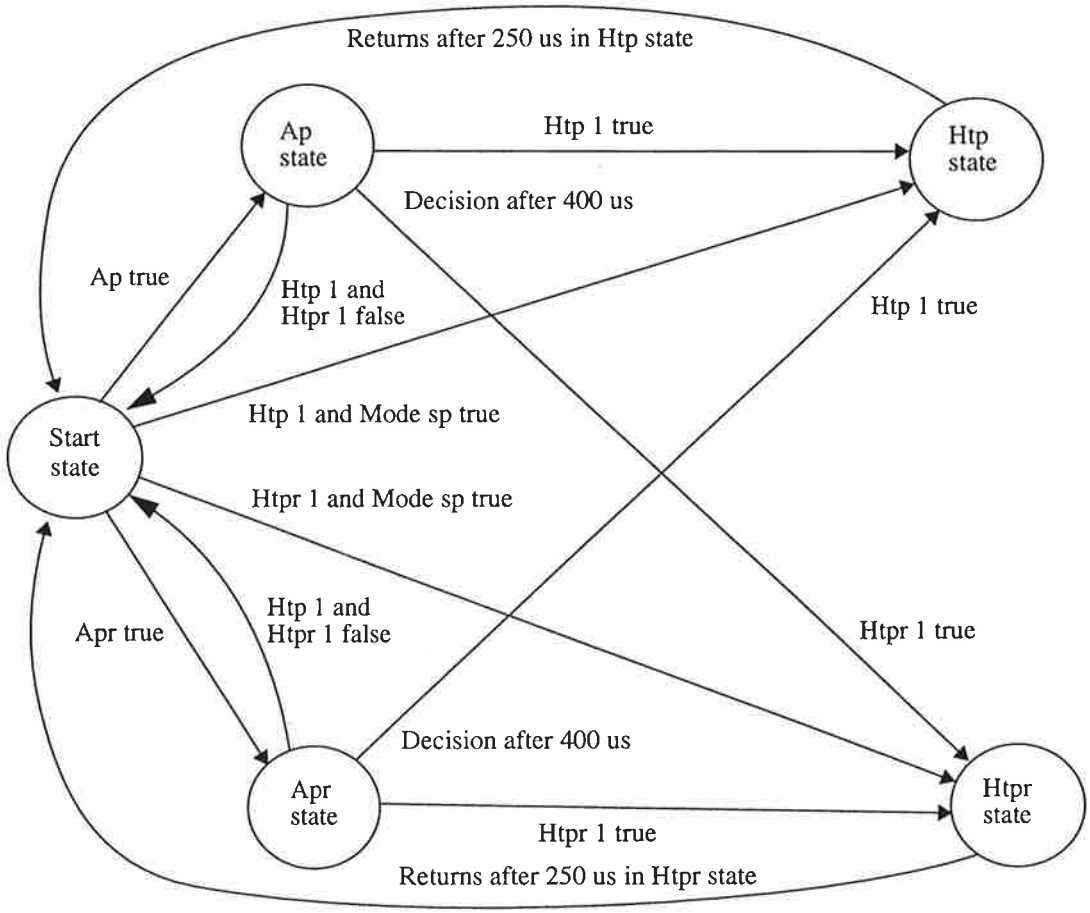


Figure 18: Principal sketch of state machine Control.

4.1.5 Tony

Tony is in difference to the other function blocks not described by a state machine. It is described directly in VHDL. This is a System Design Station feature but causes some unforeseen problems described in section 4.2.

The Tony code makes sure that only one message is given to the Master for every message generated in the design. When the Master has read the message it acknowledges this by resetting the part of Tony that deals with the actual signal.

All the error messages and activity messages are relayed through Tony, they are the error messages Rot_kurz, Rot_lang, Htp_n_m, Phs_lang and Phs_kurz, described in 2.2.4, and Ap_akt, apr_akt, Htp_akt and Htpr_akt, described in 2.2.5.

For a more thorough technical description see appendix [17].

4.2 Translation to VHDL

In System Design Station the user enters ideas predominantly in graphics. These graphics must be translated to, a for the computer understandable code. The graphics must also be tied together and the parts of the solutions described in VHDL code must be tied in. This is all done in System Design Station.

4.2.1 Generating VHDL code from state machines

The user has the possibility of generating all code at once, a part of the hierarchical design or of only generating code for one state machine or data flow diagram at a time. In Figure 19 the dialog box for generating code for the state machine Rot is shown.

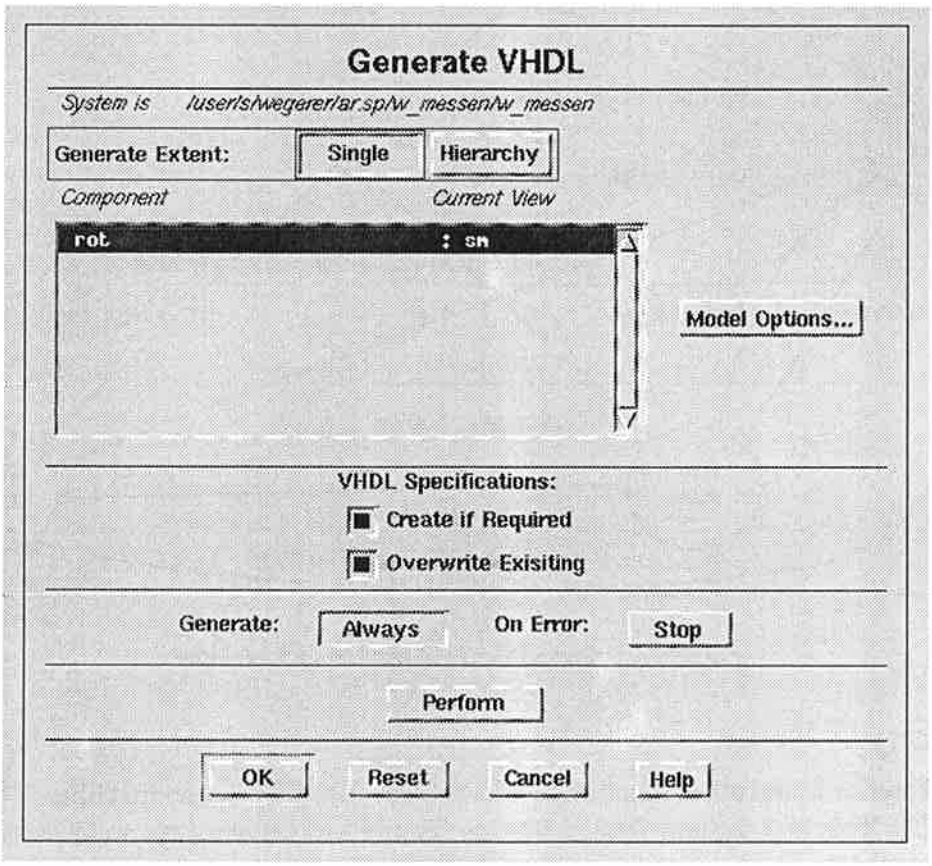


Figure 19: Dialog box for generating VHDL code.

At the top the user can choose between Single and Hierarchy. If Single is chosen code is only generated for the marked object, a state machine or a data flow diagram. If Hierarchy is chosen, code is generated for the object and all objects below the marked object.

My experience is that the three buttons Create if Required, Overwrite Existing and Generate Always must be chosen the way they are shown in the figure. If they are not chosen this way old versions of the code are not overwritten even if changes have been made since the last generation of code.

Under the button Model Options a new dialog box is shown. See Figure 20.

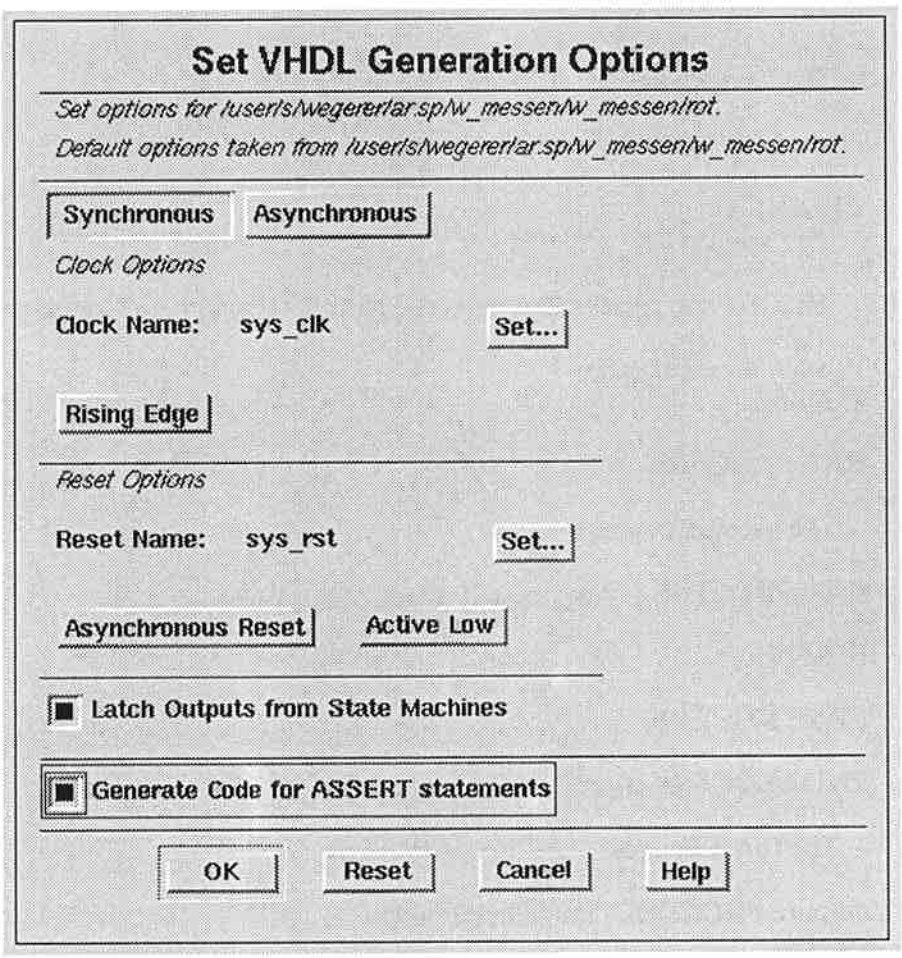


Figure 20: Dialog box for model options in generating VHDL code.

Here the sys_clk is chosen Synchronous and the sys_rst is chosen Asynchronous. The clock can be chosen either way. The sys_rst or global reset signal can be chosen either way as long as the code is only simulated. When the code is compiled for synthesis, see 4.5, the global reset function must be set asynchronous, otherwise it will not pass through the compiler.

The code generated from a state machine, with the above described settings, always has the same general configuration. Three processes represent the state machine. There are other ways of solving this problem as well but in System Design Station only this one is available. This way of generating the VHDL code generates warnings when the code is compiled for synthesis.

The three processes are the clkd process, the state process and the output process. The clkd process is the process that controls that only one state transition is done per clock cycle. This process is specific for codes generated with a synchronous clock. The state process is the process that controls which state the process should go to next. The output process is the process that controls what output should come from which state. Within the two last mentioned processes the transitions are entered as CASE statements.

A process is a concurrent statement and is therefore executed whenever the arguments in the parenthesis behind the word PROCESS change value. The statements within the process are all executed sequentially.

```
-- The first Process
--
clkd : PROCESS (sys_clk, sys_rst)

BEGIN
   IF ( sys_rst = '0' ) THEN
      state <= start_state;
   ELSIF ( sys_clk'EVENT AND sys_clk = '1' AND sys_clk'LAST_VALUE = '0'
) THEN
   state <= next_state;
END IF;

END PROCESS clkd ;
--
-- The second Process
--
state: PROCESS ( state, to_and_from, Value)

BEGIN

CASE STATE IS

END PROCESS state ;
--
-- The third Process
--
output : PROCESS ( sys_clk, sys_rst)
BEGIN

CASE STATE IS

END PROCESS output ;
```

Figure 21: VHDL code generated from a state machine.

4.2.2 To connect code from different sources

The user is able to generate code in individual blocks and test and simulate them by themselves. After each block has been fully tested and evaluated the connection can be done in System Design Station. System design Station is able to connect three different kinds of code. The hand edited VHDL code, the graphically defined state machines and the graphically defined data flow diagrams.

When code from different sources is connected the button Hierarchy in Figure 19 must be pressed. The same settings as the ones set in Figure 19 must also be set. If another setting is chosen the code will not be generated properly. For all levels beneath the selected data flow diagram VHDL code will be generated.

4.2.3 Code edited by the user

The user has the option of entering and editing VHDL code manually. This is then tied in with the computer generated code in the compilation, see 4.2.2. The user will have problems with this if a new name for the VHDL code other than the default one, spec, is not chosen. Since the settings have to be chosen as described above for compilation purposes all code will be overwritten, see [1], and a new empty architecture for VHDL will be generated under the name, spec.

4.2.4 Errors in the code

The generator of VHDL code does not test all the transition arguments for errors. It only checks if the argument is a simple action, where it checks for correctness, or a complex action, where it generates a warning. An example of a simple action is [1]

```
x:=1;
```

and an example of a complex action is

```
x:=x+1;
```

Since the complex actions are not tested for correctness the faults entered here will be checked by the compiler when compiling the VHDL code. The user then has to go back two steps to correct the fault instead of one step as normal.

The System Design Station User's Manual recommends that complex actions should only be used for simulation. If this advice were to be used, the System Design Station would not be very useful. See also appendix [1].

4.3 Compile VHDL for Simulation

The VHDL code must be compiled for simulation. This is done by the System Design Station. A dialog box as shown in Figure 22. appears. Here the compilation is set for a Hierarchical compilation of the state machine Rot. Just as when generating code it is important to set the button Compile to Always. If this is not done, old code that has been changed since the last compilation might not be recompiled.

The Mentor Graphics version of VHDL supports more functions than the IEEE standard VHDL. As is said in the on line help system BOLD browser, "Mentor Graphic has created packages that define various types and subprograms that make it possible to write and simulate a VHDL model within the Mentor Graphics environment".[2] Mentor Graphic has chosen to do this while they want to make VHDL programming easier and in some cases more close to the hardware.

1. p 2-18 System Design Station User's Manual, V8.2 Draft
2. p 9-33 Mentor Graphics VHDL Reference Manual, BOLD Browser on-line help, Mentor Graphics

Mentor Graphic does e.g. support a package called "qsim_logic" which includes a type definition called qsim_state. This is a type with four different states 0,1,X,Z. Where X is undefined state and Z is high impedance.

Qsim_state can be simulated in the Mentor Graphic program Quicksim, see 3.4 and 4.4. When the code is to be compiled for the target technology these Mentor Graphic specific definitions will be translated to IEEE standard definitions.

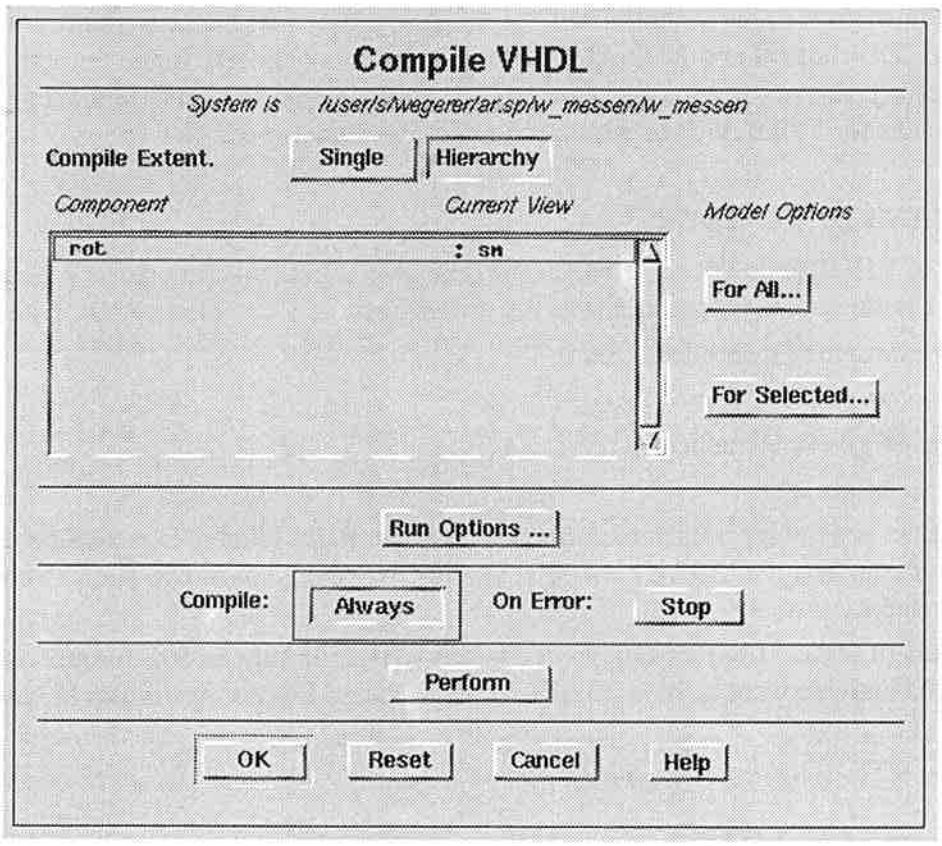


Figure 22: Dialog box for compiling VHDL.

4.4 Autologic

In our experiment the major part of the potential of the Autologic program is not used. We have only made smaller adjustments and used the translation part of the Autologic program.

4.4.1 Synthesis

In the synthesize mode no adjustments were made. The results show that adjustments in this mode doesn't give any substantive changes for the better.

4.4.2 Optimization

In the optimization mode a couple of settings have been chosen beside the default value. These settings have been found to enhance the fitting in Maxplus2 somewhat.

In the menu "Set Hierarchy Control" Flatten has been chosen for the entire design. The flatten commando transforms the solution from the hierarchical form it is given in the System Design Station, to one layer. Flatten gives us a smaller design in the Altera MAX and FLEX components.

The second set of settings that have been worked with are the recipes, see 3.6. Here it is found that the best result in the finished product is obtained if the recipe shown in Figure 23 is chosen. Noteworthy is that the only actual setting in the recipe is Area. This is set to no optimization, see [11]. The optimizer in Maxplus2 works more efficiently if it attains a code that has not been optimized at all rather than one that has already been worked on by another optimization algoritm.

The other two lines in the recipe only tell the program that a report of its work is wished for and where the user wants the computer to write the edf file. The edf is the output from autologic which is loaded in the next program in the chain, the Maxplus2 from Altera.

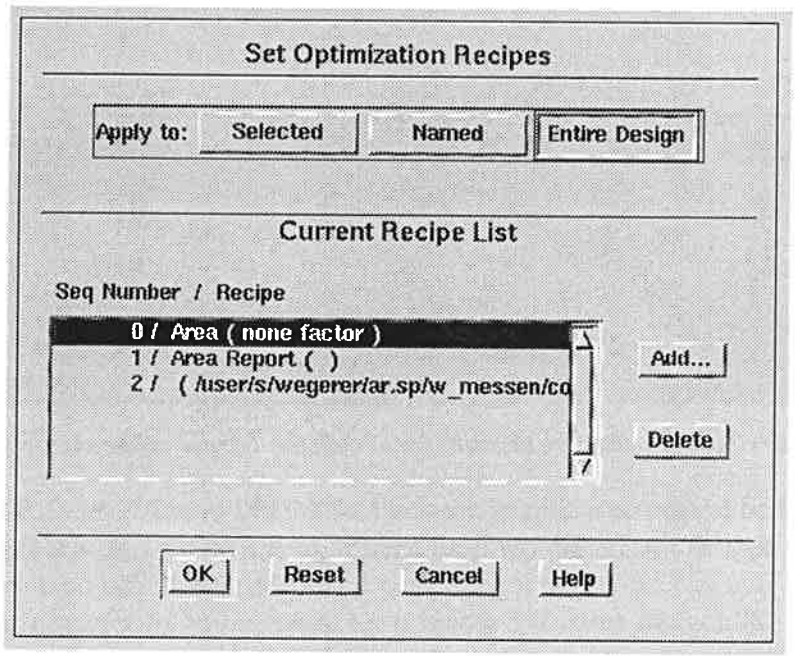


Figure 23: Dialog box for setting the recipes.

4.5 Mapping in Maxplus2

In Maxplus as in Autologic little has been done with the settings of parameters for optimizing. Here the only choice that have been made is which chips the program should choose between for mapping the solution into. This is done in the dialog box shown below, Figure 24.

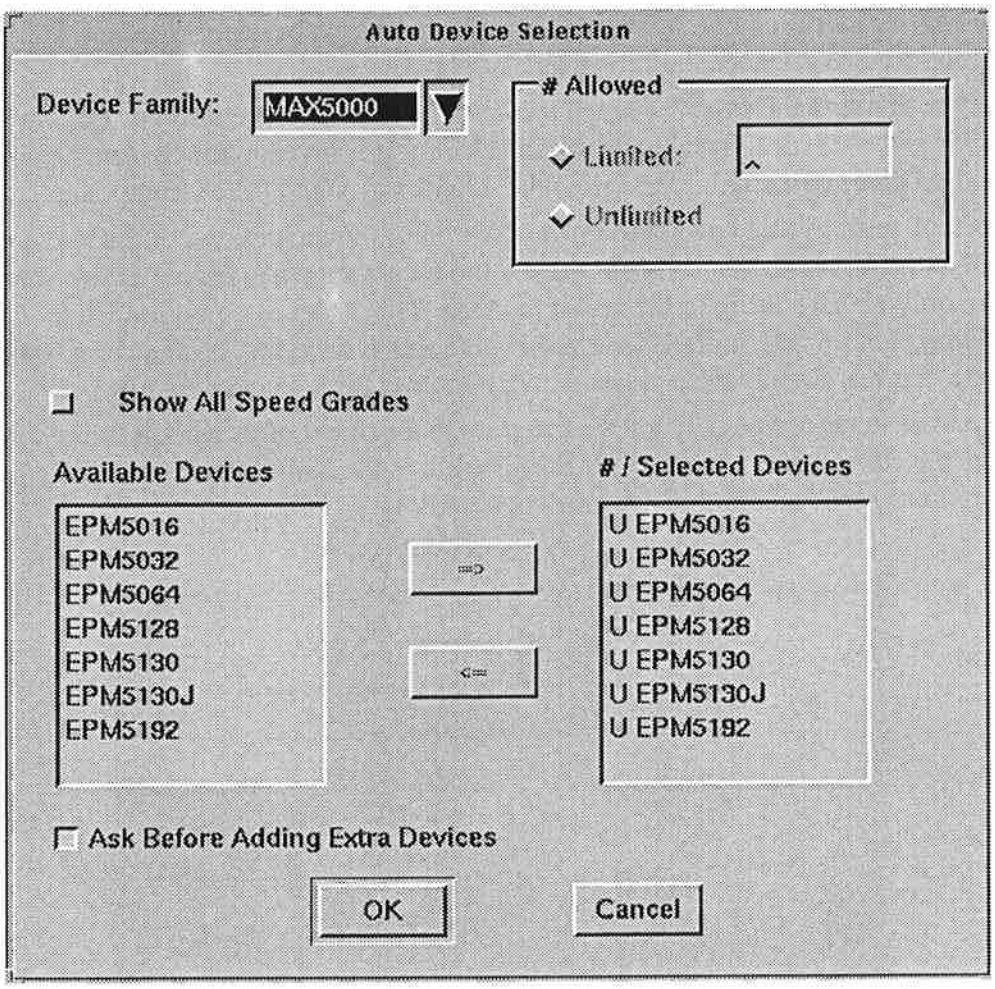


Figure 24: Dialog box for choosing which chip the design shall be mapped for.

For our solution we had to choose a chip of the size MAX 7192 or MAX 5192. This was the only size chip in which our design would fit. Below is an excerpt of the report file, see Figure 25. The utilization degree is 73%. That is 73% of the logical cells have been used. The next smaller size would be the 5128 which has 128 logical cells. We would need to get rid of 14 logical cells in order to fit our solution here.

** DEVICE SUMMARY **

Chip/ POF	Device	Input Pins	Output Pins	Bidir Pins	LCs	Shareable Expanders	%Utilized
messe_f	EPM5192	14	10	0	142	289	73 %
User Pins:		14	10	0			

Project Information/tmp_mnt/export/home/wegerer/ar.sp/ω_messen/messe_f/messe_f.rpt

** RESOURCE USAGE **

Total dedicated input pins used: 8 / 8(100%)
Total I/O pins used: 16 / 64 (25%)
Total logic cells used: 142 / 192 (73%)
Total shareable expanders used: 289 / 384 (75%)

Total input pins required: 14
Total output pins required: 10
Total bidirectional pins required: 0
Total logic cells required: 142
Total flip-flops required: 88
Total shareable expanders in database: 284

Synthesized logic cells: 54 / 192 (28%)

Figure 25: Excerpt of the report file for the fitting into a Max 5192 component.

4.5.1 Macrofunctions in Maxplus2

Counters are product term intensive. Since the logic in a Logic Cell is limited it is of interest to work around this problem.

The Altera components offer the possibility to use carry chain in FLEX 8000 and T flip flops in MAX. When using carry chain the carry generated by one macro cell propagates to the next. This is taken care of by a special link and therefore no extra logic is needed for this. T flip flops are favorable while they only need one bit of information to change state, where as a D flip flop needs to know the exact value of the next state. In order to use carry chain and T flip flops the counters must be described with macrofunctions, see appendix [1], [2] an [3].

A small example was made to illustrate this. A simple 7 bit counter in a state machine with two states was implemented in System Design Station and worked with through the chain of programs. The same counter was implemented in an editor in Maxplus2 as a predefined macrofunction and mapped by Maxplus2. The counter defined by the Maxplus2 macrofunction needed 10 macro cells and the counter defined in System Design Station needed 15 macro cells.

5 Evaluation

The problem presented in chapter two has been solved. The chain of programs has been worked through. A commercial design has been designed and has been successfully mapped to an Altera MAX 5192 component.

A single counter has been mapped to an Altera component using a macrofunction. It was possible to use macros although this is not supported by Altera for Mentor Graphic products and has never been done before. With this technique a lot of room is saved. Within the time available for the work for this Master Thesis it was not possible to try this technique on the bigger design of ω_messen.

A number of problems have been encountered along the way. The problems have been of both design nature and generated by the programs that have been worked with. Two kinds of software problems have been encountered, some that are generated by the individual programs and some that are generated in the interface between two programs. These have all been solved in such a way that the ongoing work can continue.

5.1 The solution of the technical problem

5.1.1 Simulation of the SDS design in Quicksim

All the different inputs and outputs have been simulated in Quicksim. They have all been found to coincide with the technical specifications for ω_messen. Below is a description of the simulations.

The code has been simulated in quicksim, see 3.4. The result is shown in appendix [30]. In the simulation a number of different cases have been tested. The system has been exposed to long and short signals on both inputs. All these were registered and in the cases of the short signals, they have not been sent out on the mux_out output. This is correct according to the technical specification, appendix [18].

The three different modes have also been tested, spiral, topogram and tomogram. The spiral mode is tested when mode_sp is false and the topo and tomo modes are tested when mode_sp is true. They all give the expected result.

In appendix [31] the simulation of the VHDL code written under Tony is shown. The values of the result_vector are set true when the error and activity messages are true. They are all reset as the different values of the reset_vector are true. The result_vektor is not set true again although the values of the error and activity messages are still true after the result_vector has been set false. This proves that the flip flop function works as it was designed to do, appendix [17].

In appendix [32] the simulation of the error message htp_n_m is shown. It is set true when an htpr or htp signal is blocking the buffer. In this case the htp signal (the third one on the line phs_in) is not to close. The distance is 650 microseconds which is correct according to the specifications, appendix [14]. What has happened here is that due to the way Rot_in signals has arrived in correspondence to Phs_in signals the Phs-in signal (the second from the left) has had to wait for a Rot_in signal to trigger the control state machine. As it waits in the buffer state machine speich another Phs_in signal arrives. But since the buffer is full it can't be sent on and is discarded. This is when htp_n_m is set true.

In appendix [33] the results from the simulation of the activity signals are presented. In the simulation it is evident that when the signal Ap, Apr, Htp or Htpr is laid out on Mux_out the corresponding activity signal is true.

5.1.2 Clock frequency in Maxplus2

The clock frequency and maximum delay time has been checked in the Maxplus2 timing analyzer. It is possible to work the Altera chip with a clock frequency of 11.11MHz. This is a good margin while the technical specification states 500 kHz as clock frequency for the chip.

The longest delay for a path in the chip is 76 ns. This leaves plenty of margin to the 2 microseconds with which the chip is designed to work.

5.2 Counter problems

The results shows that only macrofunctions reduce the space needed to a usable extent and that it is not worth the time and work to try to translate integer defined numbers to bit definition. Marginal gains can be attained by setting the transition conditions carefully.

Counters take up a lot of room. In the design worked with here, three counters needed to be implemented. In a binary defined number they are 9 respectively 7 bits wide. If the counters are not treated with care, the user stands the risk that the design will explode when mapped into the component. In an extreme case as much as 3,5 times the amount of room will be needed, depending on the component, for a counter when not treated at all compared to when implemented with macro cells.

Three things have been tried, working with macrofunctions from Alteras Maxplus2, setting the transition conditions where a counter is a part of the condition and transforming the integer defined numbers to bit defined numbers.Macrofunctions

In appendix [1] it is described how macro cells are used to reduce room in a component. Interesting to note is that the code produced in System Design Station is not component independent anymore. The changes that have been made in the code are described in the appendix.

In appendix [2] some of the problems that have been encountered while solving the macrofunctions problem are described.

The results reached in appendix [3] do not make sense after having read through the above. The result is presented here of two reasons. When using macrofunctions or in other ways manipulating your way outside of the mainstream that the programs here described provide, a certain knowledge of the generated code is necessary, both of VHDL as described previously and of the code generated by Maxplus2.

Secondly, it demonstrates in some way the dynamic of such a package of programs as has been worked with. The complexity and number of possible combinations make it possible to implement something that according to the programmer isn't implementable and obviously doesn't give the promised reduction of space in the programmed chip.

5.2.1 Setting the counter conditions

If the numbers defined in the transition arguments are set badly, more logic is needed to describe the argument than otherwise necessary. Since the amount of logic available in each macro cell is limited and not plentiful, see 3.7.1, this is worth looking at.

All the counter transitions have been reviewed. In the cases where it was possible the integer number has been changed to 7, 15, 63, 127 etc. The integer number will of course be changed to a bit defined number when compiled and fitted in the component. If the integer number is choosen as above and the number is defined from 0, the logic will only have to test if the top bit of the number is true or false. This will save some logic. In a design of type ω_messen fitted to a FLEX component a setting of this kind will give a reduction of 2% of the space available. Not much, might it be the 2% needed to fit at all.

5.2.2 Translating from integer to bit.

In appendix [2] it is shown that if the numbers that are integer defined are transformed to bit definition nothing is won. No experiment has been made to see if anything is gained from working altogether with bit defined numbers. This has not been done while a lot of the usability in the System Design Station interface would be lost.

5.3 Software problems

The problems run into during the design in System Design station have to the greatest extent been described in smaller reports written during the ongoing design. In the appendix the reports have been put together to what is presented in the appendix. Below follows a brief presentation of what is written in the different reports.

- In appendixes [5], [6] and [7] problems with System Design Station is discussed.
- In appendix [8] the implementation off an example in System Design Station is presented.
- In appendix [9] some experiences from VHDL are discussed
- In appendix [10] Quicksim and Design architect are commented
- In appendix [11] Autologic is commented

6 Conclusions

It is possible to design and map a commercial FPGA with the computer tools used in this Master Thesis. It has been shown that the snags and bugs in the programs are possible to work around and that the finished product is commercially usable. It is no bold guess to say that this is transposable to other FPGA manufactures and other producers of VHDL interfaces. Every manufacture of FPGA components or VHDL interfaces have their own peculiarities built in but in greater parts these problems are possible to overcome.

6.1 Space

Space is lost when not working with the language of the FPGA producer. A counter implemented with Alteras own design language needs one Logical Cell for every bit where as a counter implemented in VHDL needs 2 to 3 Logic Cells for every bit of counter width. Some space is lost due to that the problem is described in VHDL and some space is lost due to that we are using another program as an interface and translating the code to VHDL.

The Altera macrofunctions have proven to be the most efficient way of reducing space in the programmed component. As much as 60% can be saved in extreme cases. It is important to note that when using macrofunctions the code is manipulated in such a way that it is not manufacture independent anymore. The changes are not very great but never the less it takes some prior knowledge of the code to change it. This could be a problem if one wanted to change FPGA manufacture e.g. a year after the original design was made and some of the movability that this work wants to show is lost.

Another problem with Alteras macrofunctions is that because they are not supported by Mentor Graphics and vice versa they demand knowledge to a greater extent than the ordinary user needs to just run the programs.

Although time limited proving that it is possible to apply macrocells to a bigger construction it is in my view possible to apply them. With macrocells it is commercial to use the technique shown here also for bigger designs with wider counters. Without macrofunctions the interesting range of problems for which this setup of programs can be used is greatly diminished.

It is shown here that it is possible to work with these macros in System Design Station. This is very important because it shows that System Design Station is not only usable for smaller arranged problems but is in fact a professional tool usable for commercial problems. This is a strength for the program.

6.2 Software

The software used for this Master Thesis is by people in the business called "professional", as opposed to word processors, calculation programs and games. This software is different in the sense that it is more complex and more specialized than other software. This does off course also make it more difficult to learn. The experience gathered in this work is that the difficulty to learn does not only lie in the complexity and the abstractness of the problems worked with but also in the programs itself. The many errors and peculiarities described in e.g. appendix [5] contribute to this picture. This applies mainly to the Mentor Graphic software. But it must also be said that the Mentor software is a great tool when fully comprehended by the user.

6.2.1 System Design Station as interface

The System Design Station builds on an idea that is more and more becoming the mainstream of programming, graphical programming. It is very nice to be able to work with pictures that describe the problem instead of having to write pages of code and having to look out for every comma. The program also offers a great advantage when it comes to organizing the work. Everything is laid out in nice pictures where the data flows and signal flows are easy to follow. Compared with traditional programming this is a great advantage.

System Design Station has many bugs that need to be corrected, although none are lethal enough motivating giving up System Design Station altogether. One is often struck by the feeling that the program was given to the market before development had a fair chance of doing a good job.

7 Abbreviation list

CAE	Computer Aided Engineering
CT	Computer Tomograph
DA	Design Architect, from MentorGraphic
EPLD	Erasable Programmable Logic Device
FLEX	Flexible Logic Element matriX, from Altera
MAX	Multiple Array matriX, from Altera
MTBF	Mean Time Between Failure
SDS	System Design Station, from MentorGraphic
VHDL	Very high speed circuit Hardware Description Language

8 Vendors

MentorGraphics, Mentor Graphics Corporation 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070, USA

Altera, Altera Corporation, 2610 Orchard Parkway, San Jose, CA

9 References

At a first inspection the list of reference might look a bit unorthodox. Two groups may be identified. In the first group are the different manuals, a data book from the program manufactures and the IEEE standard for VHDL. These are referred to when describing the technical side of the problem.

In the second group are the Oxford Dictionary, a handbook in electronics and The taming of the shrew. These references are referred to for different reasons. The german handbook in electronics is chosen to demonstrate the generality of the tool state machines. The dictionary is referred to because the words used in technical descriptions and work does not always coincide with the everyday language and vocabulary. As W Shakespeare is concerned the preface speaks for itself.

[1] IEEE Std. 1076-1987, IEEE, Inc., 345 East 47th Street, New York, NY 10017, USA.

[2] Data Book, Altera Corporation, 2610 Orchard Parkway, San Jose, CA, USA.

[3] Mentor Graphics VHDL Reference Manual, BOLD Browser, on-line help, Mentor Graphics Corporation 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070, USA

[4] The Concise Oxford Dictionary, Oxford University Press, Walton Street, Oxford O X2 6DP, GB

[5] System Design Station User's Manual, V8.2 Draft, Mentor Graphic Corporation 8005 S.W. Boeckman Road, Wilsonville, Oregon 97070, USA

[6] Taschenbuch Elektrotechnik, Band 2 Grundlagen der Informationstechnik, VEB Verlag Technik Berlin, 1977. Prof. Dr. sc. techn. Dr. techn. h.c. Eugen Philippow, Technische hochschule Ilmenau,

[7] The taming of the Shrew, William Shakespeare

10 Appendices

[1] Macrofunctions in System Design Station.
[2] Notes on macrofunctions in System Design Station.
[3] Macrofunctions, a curious example.
[4] Integer to bit transform
[5] Comments on System Design Station
[6] Internal variables, Complex variables and Type definition in System Design Station
[7] Overwriting existing VHDL code in System Design Station
[8] Counter in System Design Station
[9] Comments on VHDL
[10] Comments on Quicksim and Design Architect
[11] Comments on Autologic
[12] Specifications for Clk Divide
[13] Specification for Control
[14] Specification for Phs
[15] Specification for Rot
[16] Specification for Speich
[17] Specifications for Tony
[18] Specification for ω_messen
[19] Context Diagram for ω_messen
[20] Data Flow Diagram for ω_messen
[21] State Transition Diagram for Control
[22] State Transition Diagram for Phs
[23] State Transition Diagram for Rot
[24] State Transition Diagram for Speich
[25] VHDL code for Control
[26] VHDL code for Phs
[27] VHDL code for Rot
[28] VHDL code for Speich
[29] VHDL code for Tony
[30] Simulation of ω_messen
[31] Simulation of Tony
[32] Simulation of htp_n_m
[33] Simulation of activity signals

11 Corriculum Vitae

Martin Per **Hägerdal**

Address:	S Esplanaden 16, 223 52 Lund, Sweden
Tel	+46 46 15 85 60
Born:	Lund, Sweden, March 19, 1966
Nationality:	Swedish
Civil state:	Single

Education:

Elementary school, Swarthmore, PA,USA	1976-1978
High school, Katedralskolan, Lund	1982-1985
Military service, Anti-Aircraft Battalion, Ystad	1985-1986
Electrical engineering and applied physics, Linköping Institute of Technology	1987-1988
Electrical engineering, Lund Institute of Technology	1988-
Master Thesis, Siemens Med, Erlangen, Germany	1993-1994

Languages:

Swedish, English, German

Working experience:

Bank clerk, Lundabygdens Sparbank, Lund	1986-1987 Summers 1988 1989
International Sales Department Barsebäck Nuclear Power Plant	Summer 1990
Technical Practicioner Tetra Pak, Romont, Switzerland	Summer 1992
Technical Practicioner ABB High Voltage Cable, Karlskrona	Summer 1993

Organizational engagements:

President of Kalmar students' club, Lund	1991
Chairman, Students' clubs of Lund	1992

References:

Allan Månsson
General Manager, Foreign Business
Barsebäck Nuclear Power Plant
Sydsvenska Värmekraft AB
Box 524
S-240 21 Löddeköpinge
tel Int +46+46-724134
fax Int +46+46-775793

Per Stjernquist
Professor
tel Int +46+46-116233

Håkan Wahlfrid
Personal Manager
Tetra Pak International
tel Int +46+46-361000

Herr Fritz Peter
Manager, Electronic Development CT
Siemens UB Med
tel Int +49 9131 84 3524
fax Int +49 9131 84 5281

1 Macrofunctions in SDS

1.1 Background

In a construction a counter is a common component. A counter that has been defined as an integer and written in VHDL might look something like the following:

```
PROCESS (sys_clk);
IF sys_clk'event AND sys_clk'lastvalue = 0 THEN
   IF enable TRUE THEN
      X <= X + 1;
   ENDIF
ENDIF
END PROCESS
```

Figure 1: For loop in VHDL.

If this is implemented with Maxplus2 a lot of extra room will be needed when the construction is mapped to a FLEX or MAX component. In Flex 8000 a counter that is 7 bit wide will need 36 logic cells (LC). If the same counter is mapped directly with a macrofunction written for the Maxplus2, only 7 logic cells are needed. With this background it is very attractive to try to use macrofunctions for implementing counters. The problem is that macrofunctions are not supported for Mentor Graphics products by Altera.

We have been able to implement the macrofunctions anyway, but with some difficulty. This is described below.

1.2 Method

1.2.1 The code, System Design Station and VHDL

Below follows a description of how a macrofunction should be implemented in a solution with counters. This part of the implementation has not been carried through while time was a shortage. It is shown here anyway because it is believed that this is the way such a solution should be done.

The counters have been taken out of the state machines and put in special blocks, see [1]. Within a block the administration of the counter is taken care of. The counter itself is instantiated by an empty architecture body in an independent file. In the empty architecture body, the Maxplus2 will map the macrofunction.

The block, where the counter is implemented, has two inputs and one or more outputs. The two inputs are Count_enable and Load. When Count_enable is false the counter counts up, one step every clock cycle. When Count_enable is set true the counter stops. When Load is low the counter is set to zero and when true nothing happens. The output is a number defined as an integer. The range of the integer number depends on the counter macro that is implemented. Here two different counters have been worked with. One 8 bit wide and one 16 bit wide. The 8 bit wide counts between 0 and 255 and the 16 bit wide counts between 0 and 65 536.

To move the counter outside of the state machine the state machines have to be changed. Every command including x, e.g. x <= x+1, is thrown out and replaced by commands that sets the two control commands, see [2], [3], [4]. Everything else is left untouched.

1.2.2 Compiling in SDS and Autologic

While Altera doesn't support macrofunctions described in Mentor Graphic a few irregular steps and measures have to be taken to get an edf file that the Maxplus2 can accept.

The VHDL code is compiled without any changes as to the normal compilation described in the Master Thesis. In Autologic, no changes are made in the Synthesis phase, in the optimize phase a few changes are made. The parts of the synthesized construction where the empty architectures are placed are given the constraints "don't touch" and "in place". For the rest of the construction "dissolve" is chosen. The recipe is chosen as described in the Master Thesis, i.e. no optimization. This brings us a construction where everything except the macrofunction area is mapped in the same level.

This construction is divided into areas labeled primitives and areas labeled non primitives. When an area is labeled as primitive this area is left untouched by the compiler. If the area is a non primitive the compiler will automatically look beneath it to search for missing pieces e.g. empty architecture bodies where we want to map our macrofunctions. A net list, such as the one we have just produced is per default labeled a non primitive and parts of it will therefore have to be renamed. This is done by editing viewpoints.

Before leaving Autologic viewpoints are added for the macrofunction areas. This is done under the menu edit>add>property. In the box Existing Property Name "model" is choosen and given an appropriate name.

Mentor Graphic has two different compilers for files that is to be compiled to edf format. One is available through Autologic and the other is available through enwrite. The one available through enwrite allows us to have primitive areas in a non primitive netlist.

The optimize command of Autologic produces different outputs. It produces, among others, an edf file and an eddm file. If this was a normal construction we could now use the edf file in Maxplus2 to map it. With this type of construction we load our eddm file into the enwrite compiler. This is done while the Autologic compiler doesn't allow mixing non primitiv areas with primitiv areas.

Enwrite is most easily accessed from Mentor Graphics Design Manager. Within Design Manager the map with the construction is choosen and enwrite is activated through the menu.

1.2.3 Mapping in Maxplus2

As described in the Master Thesis no special settings have been made in Maxplus2. This is true for the 5000 technique. In the 8000 technique carry chain has to be choosen. If this is not done the program will not use this possibility to decrease space and subsequently the mapped file will appear larger than need to be.

1.3 Result

1.3.1 Results from a single counter

A singel counter has been implemented, see Report w15. The counter is seven bit wide. It has been mapped into the MAX 5000 and the FLEX 8000 technology. The result is shown in Table 1.

Technology	Construction	With Macros	Without Macros
FLEX 8000	SDS counter	11 LCs	36 LCs
	Only macro	10 LCs	
MAX 5000	SDS counter	10 LCs	15 LCs
	Only macro	9 LCs	

Table 1: Number of Logic Cells needed under different conditions.

The greatest gain is done in the FLEX technology. This is due to that the FLEX technology supports carry chain and this can only be fully utilized with the macrocells. An interesting information is that when mapping a single macrocell, we use 10 LCs in FLEX and 9 LCs in MAX. With our administration of the counter we only loose one macrocell in the each of the technologies.

1.3.2 Results from the w_fit

Due to time shortage and to problems run into this was not tried out fully. The problems we have run into are transforming vektor defined variables to single bit defined variables, and some problems with synthesising the code in Autologic. When judging the work done for this appendix it is no impossibility to implement macrocells for bigger constructions but it demands an even greater knowledge of the programs than is needed for the smaller example.

1.4 Conclusion

A way has been found to implement Altera macrofunctions in System Design Station, although this is not supported by the program manufactures. The way is far from straight forward but it is workable.

With the macrofunctions, the area needed in the mapped component is drastically reduced. In the FLEX technology only 11 Logic Cells, are needed for a 7 bit counter, when using macrocells that can utilize carry chain, compared to 36 when not using macrofunctions. In the MAX technology the gain is not as dramatic but still not unimportant.

2 Appendix

[1] Data Flow Diagram for w_fit

[2] State Transition Diagram for control

[3] State Transition Diagram for phs

[4] State Transition Diagram for rot

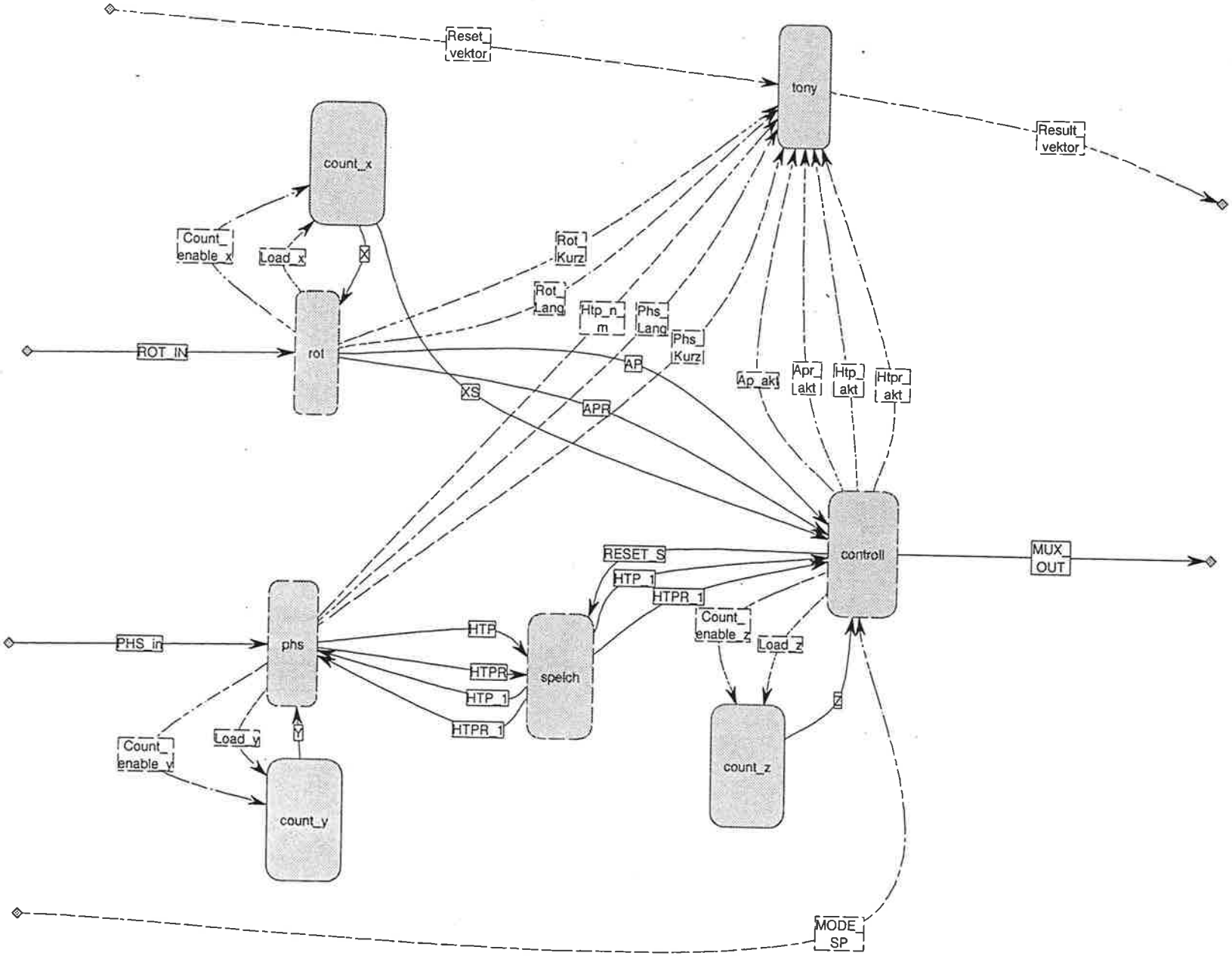
[5] VHDL code for count_x

[6] VHDL code for count_y

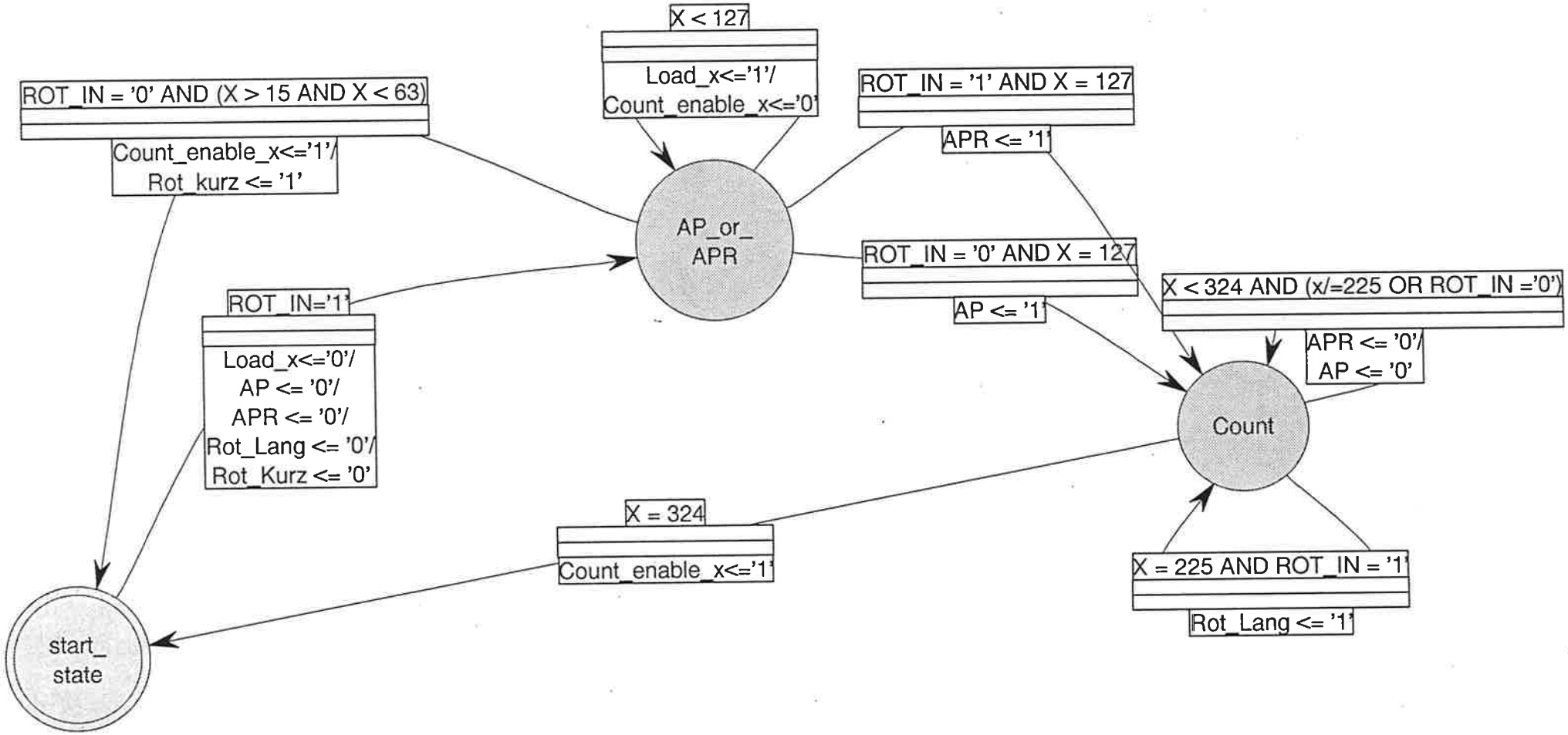
[7] VHDL code for count_z

[8] VHDL code for soft_count_8

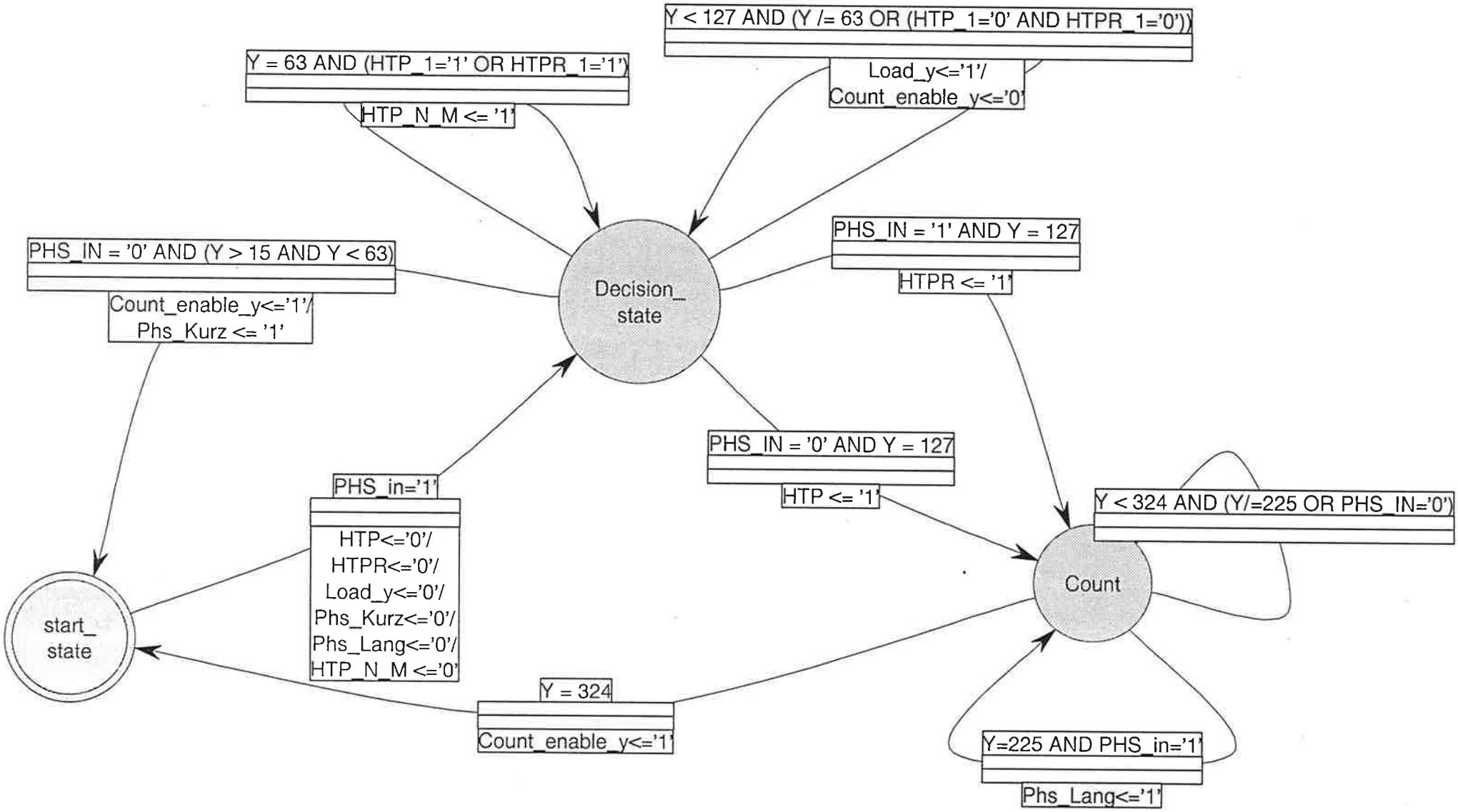
[9] VHDL code for soft_count_16



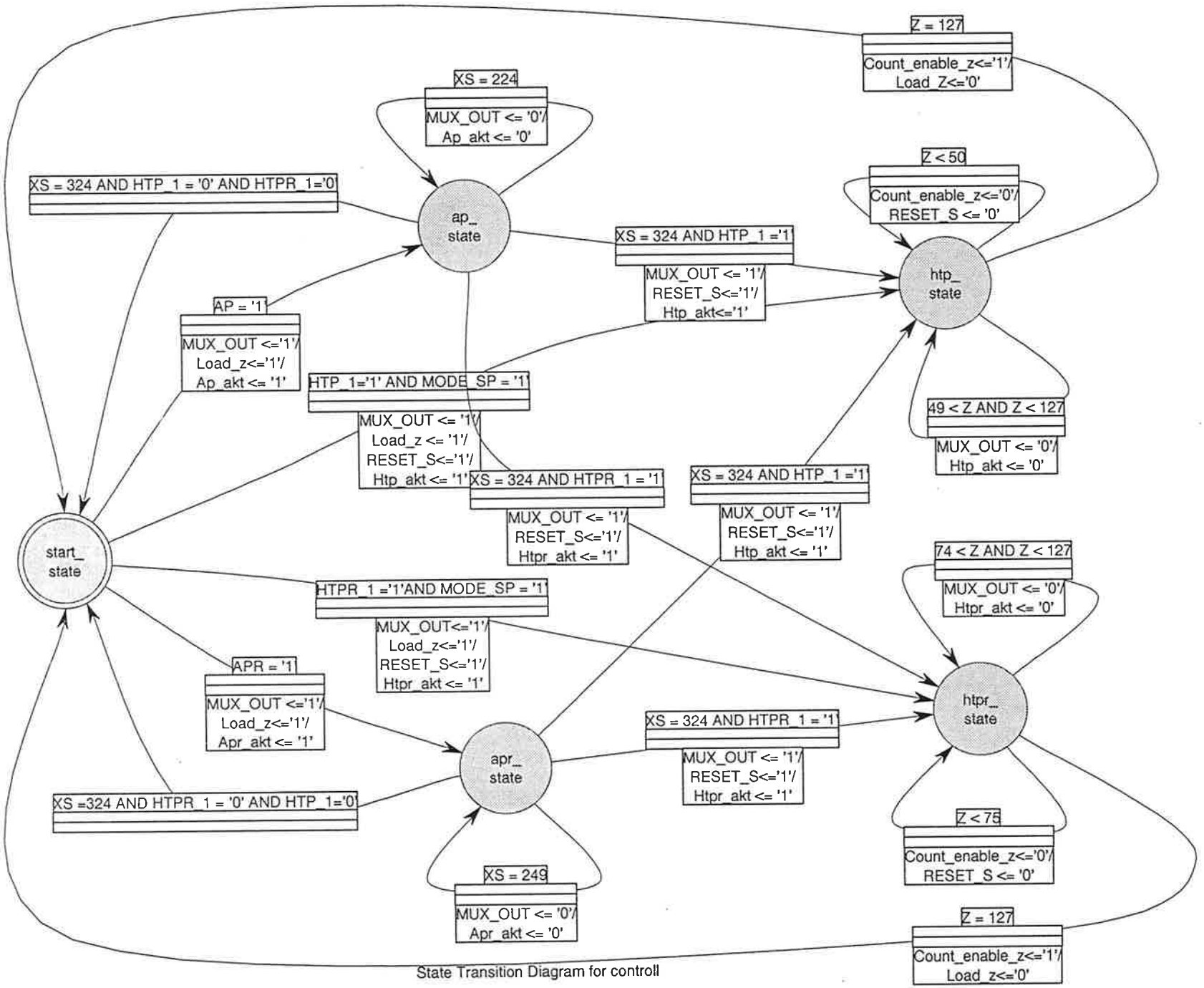
Data Flow Diagram for w_fit



State Transition Diagram for rot



State Transition Diagram for phs



State Transition Diagram for controll

1 Notes on macrofunctions in SDS

In this report work done with counters in System Design Station is presented. Counters are of interest since they take up a lot of room. Space is a shortage in all hardware construction and it has been judged by the Siemens development department that if the amount of space needed for counters can not be decreased the current setup with software and hardware components must be abandoned.

The Altera company offers its users the possibility to use macrofunctions for mapping certain large and difficultly mapped functions in Maxplus2. This is intended to enhance speed and decrease room in the programmed component. The problem is that Alteras Maxplus2 doesn't support this function for Mentor Graphics products.

A counter has been implemented in System Design Station, see appendixes 1 - 5. The parts implemented in the flow diagrams and VHDL code is only the bureaucracy for the counter. The counting part is not included in the code at all. A blank space has been left in the VHDL code in appendix 5 where the macrofunction is to be mapped by the Maxplus2.

The construction has been taken through the entire Mentor software and an efd file has been generated.

Firstly: In the original file the output was mapped as a vector and the Maxplus2 didn't accept this. This has been hand edited away.

Secondly: We got error messages of the kind "can't find output_1". We have previously encountered the same kind of error messages, when by mistake including empty architecture bodies in files worked with in Maxplus2. This leads us to suspect that the program wasn't able to read the lmf file correctly. Since we have not been able to sort out how the word behind LIBRARY should be written we believe this to be a source for our error.

Thirdly: The entire mnt8_b.lmf was copied to the working directory and what was in our lmf file was copied to it. This resulted in the error message "can't find 8count". 8count is the name of the macrofunction we are trying to map.

Fourth: The macrofunction was copied to the working directory. This gave the same result as described above.

2 Appendix

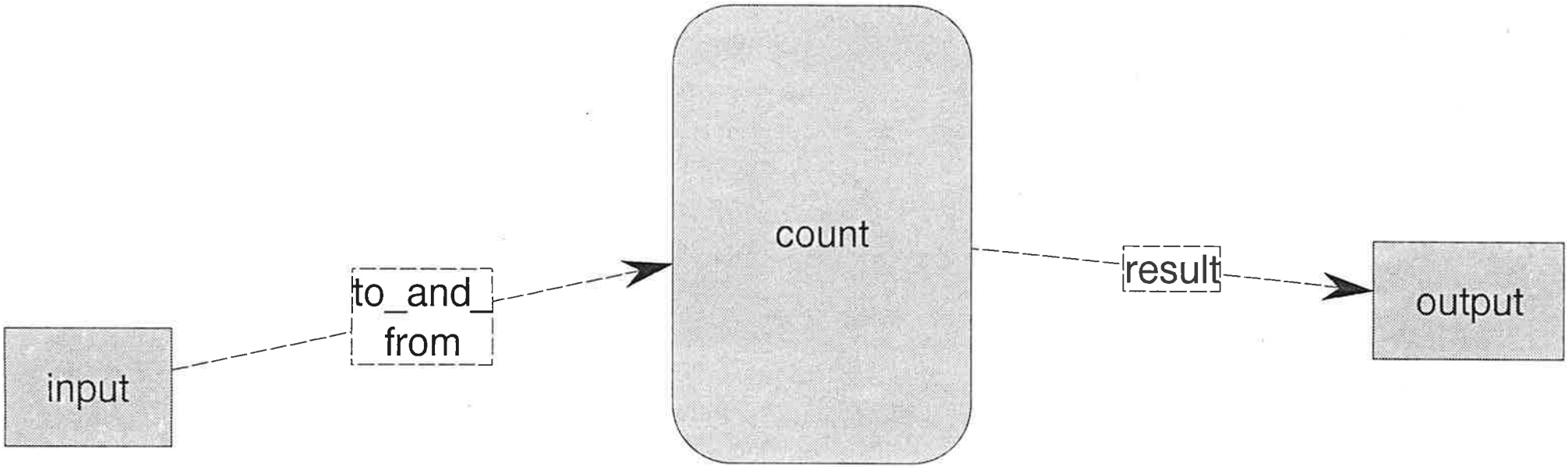
[1] Context Diagram for count

[2] Data Flow Diagram for count

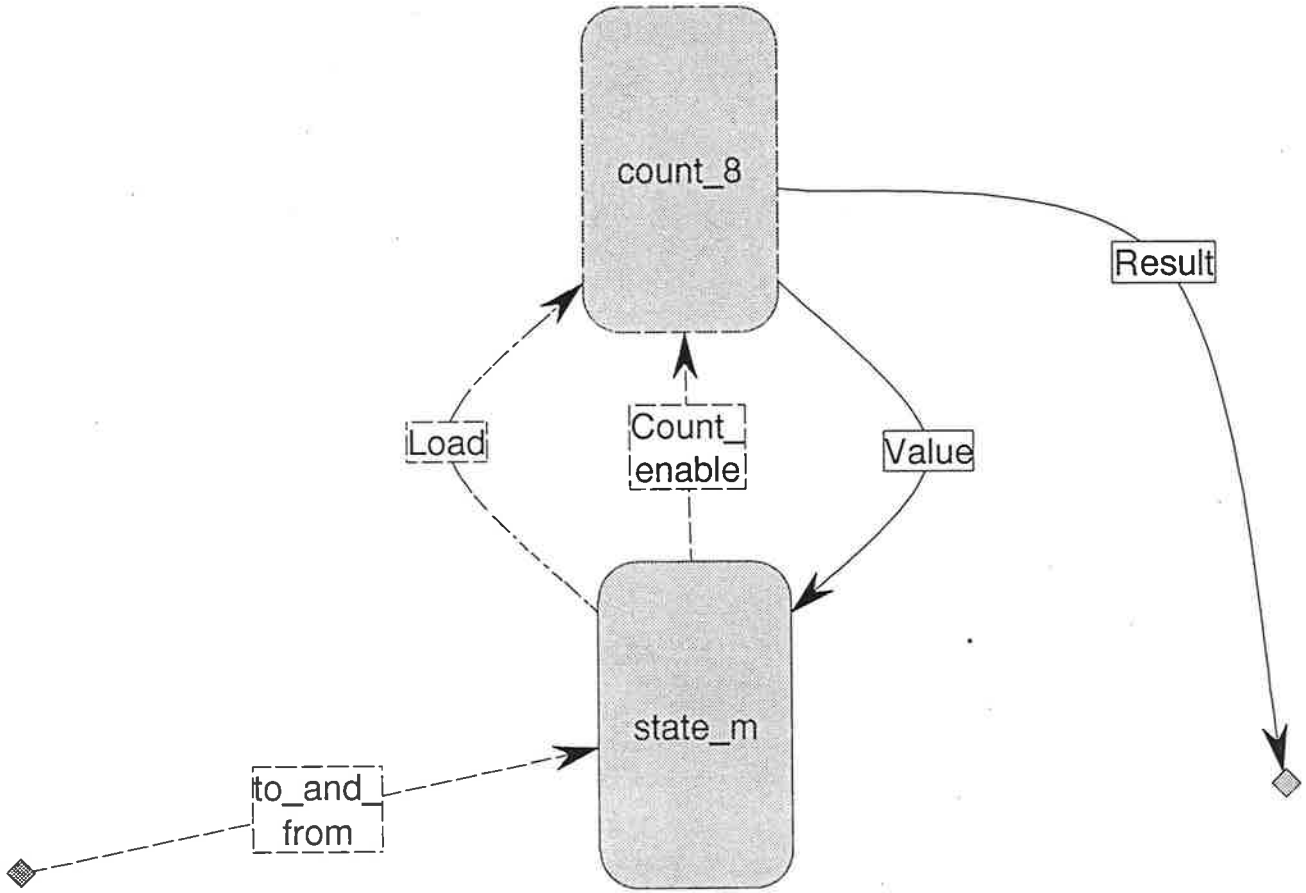
[3] State Transition Diagram for state_m

[4] VHDL code for count_8

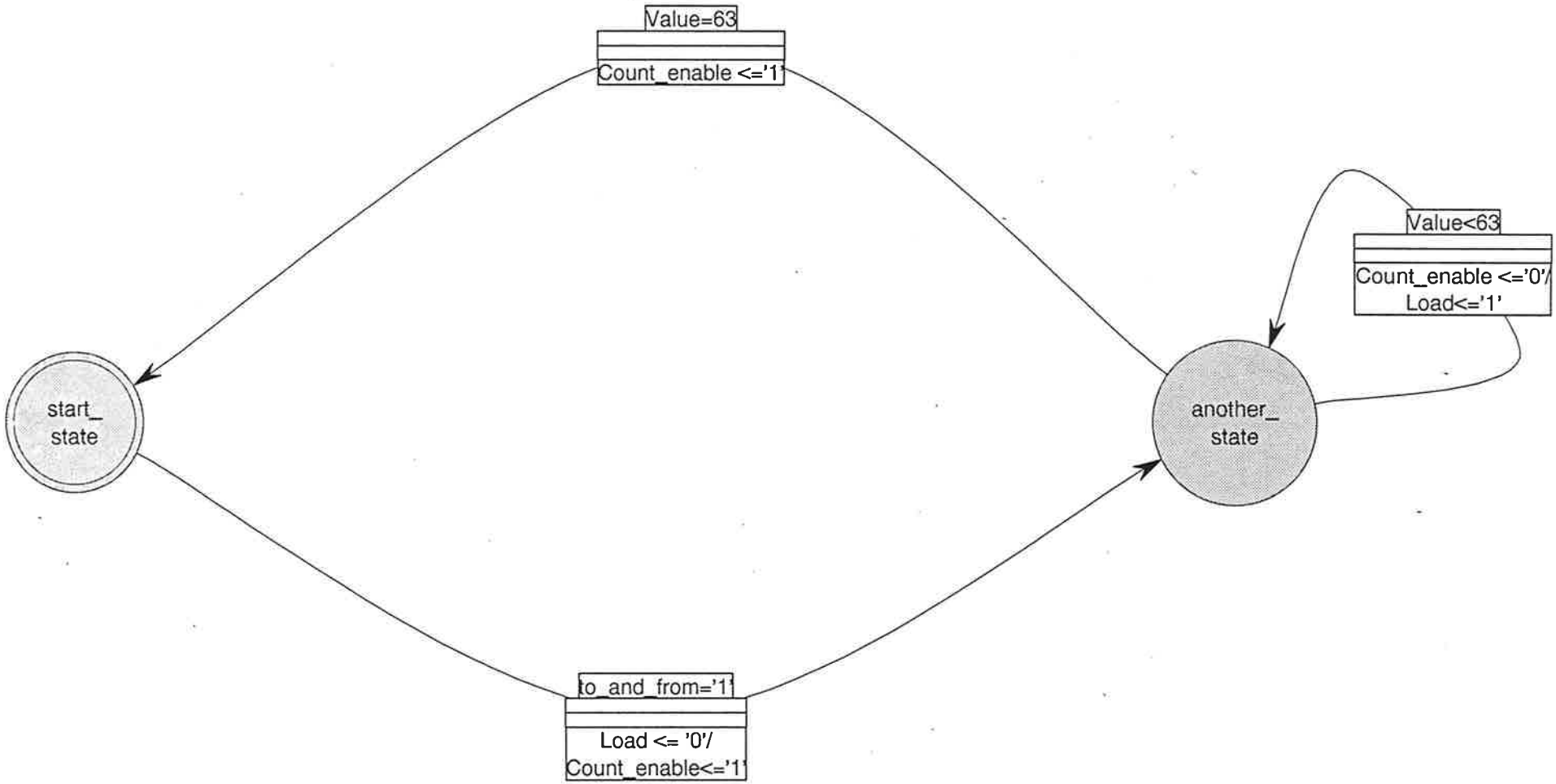
[5] VHDL code for soft_count_8



Context Diagram for count



Data Flow Diagram for count



State Transition Diagram for state_m

1 Macrofunctions, a curious example.

The problem has been to fit a construction edited in System Design Station (SDS) into a FLEX 7000 component. As shown in table 1 the construction needs 76% of the biggest chip and 81% of a smaller chip in the FLEX 7000 family to fit. This is unsatisfactory because of spacial and economic reasons. The goal is to be able to fit the entire construction into one FLEX 7000 family component.

In this experiment the macro functions of the Altera Max+Plus II has been tried. The macro functions are supposed to be mapped more efficiently in the component than the corresponding function described in VHDL.

The counters are the single parts of the construction that use the most space in the programmed component. Therefore it is interesting to see in what way they might be minimized.

The counter that is replaced is an integer defined counter. It is set to count from 0 to 324. Therefore a minimum of nine bits are needed to implement it. Because it is integer defined a transformation must be made from the bit defined output of the macro function to decimal. The program uses the decimal number in all the boolean expressions.

1.1 The rot code

The macro function is implemented in the original code as a VHDL component. A component is a concurrent statement and can therefore not be used within VHDL functions whose arguments are executed serially. Such functions are IF, WHILE, PROCESS etc. Because of this, the implementation of macro functions in a code already generated by SDS causes some problems.

The component is entered in the beginning of the code, although the place in the code doesn't make any difference, because all concurrent commands are concurrently executed. The counter command is controlled with flags that are set within the serially executed commands. The names of the flags are to_zero and count_upp, app 2.

As mentioned above the output of the macro function is bit defined. When the value of the counter is used in the program it is used as an integer. We therefore need a transform that takes care of this. MentorGraphics provides a predefined function in their Predefined System-1076 Packages[1] to_integer, figure 1.

```
to_integer (val : qsim_state_vector, x : integer)
RETURN integer;
```

Figure 1Definition of bitvector to integer transformation function in VHDL by MentorGraphics.

This function transforms a vector defined as a qsim_state_vector to an integer. It is also possible for the user to define which value a x-valued bit should have. This is entered as x in the above code. It is important that the vector is qsim_state defined. The function doesn't accept anything else.

1.2 The file count_16

This is a separately edited file, app 3. It has its own container. The main file, rot, knows of its existence through the component definition and the command USE work.all in the rot file, app 2.

1. p 44, System_1076 Quick Reference Booklet ver 8.2, MentorGraphics, 8005 S.W Boeckman Rd, Wilsonville, Oregon, USA

The file contains an entity and an architecture. The entity is defined as usual with a port, stating the inputs and outputs and their types. Here qsim_state definitions are choosen for all signals. This choice is made because the above described function, figure 1, is qsim_state defined.

The architecture is left blank. In this area the Altera Max+plus II will map the macro function automatically.

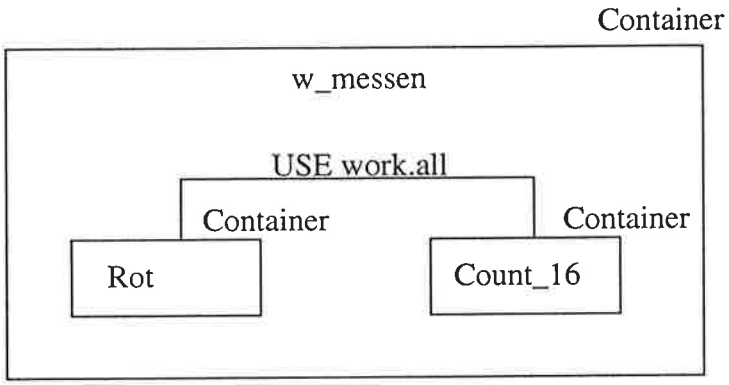


Figure 2Sketch of how the containers are connected.

1.3 The macro function 16CUDSRB

Altera provides over 300 macro functions for the Max+Plus II program. They range from Adders over Converters to Digital filters. Here a counter has been worked with.

The 16 bit counter was choosen because of size, figure 3. The counter that is to be replaced is a nine bit counter. In the macro functions the user has to choose between a 8 or 16 counter and there is nothing in-between.

This counter is a very flexible component. It provides the user with the functions: clear all outputs to low level, Reset all outputs to high level, Shift Left, Shift Right, Count Down and Count Up. Only the first and the last function is used in our example.

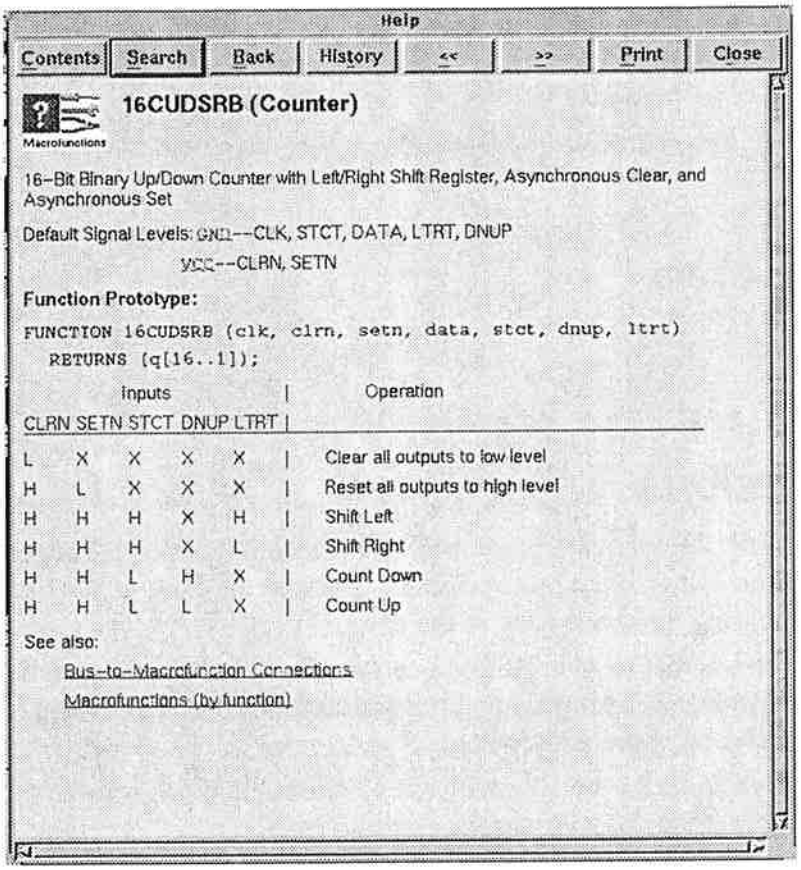


Figure 3The definition of 16CUDSRB as described in the Max+Plus II on-line help

1.4 The .lmf file

The Library Mapping file , app 4, (.lmf) is "... used to map cells in an EDIF Input File to corresponding Max+Plus II primitives and macro functions ..." and to "... use descriptions of Altera-provided functions that are optimized for EPLD architecture, letting you fit more logic into a single device ..".."[1] In our case the optimized file is the macro function.

Here the "count_16" file and its ports is mapped to the macro 16CUDSRB.

1.5 Results

Our experiment shows that more space on the component is needed with the macro function than without. The result is presented in table 1 and 2. Without the macro function 78% of the capacity of the two chips is utilized. With the macro function 82% of the capacity of the two chips is utilized.

1. p 6 Max+Plus II Compiler ver 1.0, Altera Corporation, 2610 Orchard Parkway, San Jose, CA

Device	Degree of utilization
EPM 7192 GC 160	76%
EPM 7128 LC 84	81%
Total	78%

Table 1Utilization of the components using the original code.

Device	Degree of utilization
EPM 7192 GC 160	97%
EPM 7128 LC 84	72%
Total	82%

Table 2Utilization of the components using the tampered code.

1.6 Conclusions

The result is discouraging. After having implemented the macro function the degree of chip utilization increases instead of decreasing. What has been tried here and they way it has been tried is not a possible solution for decreasing the space used in the chip.

A possible reason for this is that an overkill has been made concerning the functionality of the macro function. The original counter is 9 bit wide and it is replaced with a 16 bit counter. The 7 bits not used in our example might take up space somewhere.

The macro function used provides the user with six different functions of which only two are used in this example. This might effect the space usage negatively.

2 References

[1]System_1076 Quick Reference Booklet ver 8.2, MentorGraphics, 8005 S.W Boeckman Rd, Wilsonville, Oregon, USA

[2]Max+Plus II Compiler ver 1.0, Altera Corporation, 2610 Orchard Parkway, San Jose, CA

3 Abbreviation list

SDS	System Design Station
VHDL	Very high speed Hardware Description Language

4 Appendix

[1] Original code for rot

[2] Changed code for rot

[3] Code for count_16

[4] lmf file for rot

[3]

1 Integer to bit transformation

We have choosen to begin with trying to transform the integer defined number to bit and then do the incrementation. This is tried because it doesn't involve as much changing of the original code as implementing a macrofunction does. The two codes and their bydefinitions are shown in figure 1 and 2.

```
X <= X + 1;
```

Figure 1Code for incrementing the integer value number.

```
SIGNAL add_upp : qsim_state_vector (8 DOWNTO 0) := "000000001";
VARIABLE Xbit  : qsim_state_vector (8 DOWNTO 0) := "000000000";

Xbit := to_qsim_state(X,9);
Xbit := Xbit + add_upp;
X <= to_integer (Xbit,0);
```

Figure 2Code for the Integer to bit transform and incrementation of the bit value number.

In the compilation the same configuration for the compilers are choosen. This is true for the entire experiment down to the maxplus2 program. Here we have choosen to let the program choose device by itself. Our construction doesn't fit into one device and consequently two devices have been programmed. The program has choosen the devices

Device	Degree of utilisation
EPM 7192 GC 160	76%
EPM 7128 LC 84	81%
Total	78%

for the untampered code and the devices

Device	Degree of utilisation
EPM 7192 GC 160	72%
EPM 7128 LC 84	88%
Total	78%

for the changed code. There is no difference in the total degree of utilization. The difference in utilization within the two blocks is probably a result of that the program has begun with fitting in different devices.

2 Results

This experiment proved that a more powerfull manipulation must be used to shrink the area demanded. The method used didn't save us any space at all.

1 Comments on SDS

2 Introduction

At a first glance SDS looks like a toy and easy enough for a child to use. This is very appealing in the world of engineering where many things are abstract and complicated. An, a little more thorough inspection gives the user a quite different picture.

In order to use SDS to a greater extent, one has to have a good knowledge of VHDL, the compilation of VHDL for synthesis and SDS itself. It turns out that there is a number of constraints when programming SDS, some lie in the compilation between SDS-VHDL, some in the compilation of the VHDL code for synthesis and others in the SDS program. The SDS program is has not been completely developed and the user runs into problems like the one described in section 2.4 and 2.7.

2.1 Example: update window, redraw window

When working from the navigator window the command for refreshing the window is update window and when working in the design window the command is redraw window. This is not an actuall problem but unfortunately the example is a good instantiation of the general impression of the program.

2.2 Example: New type definition

Under File>Open Type Definitions a menu is available. In this menu there is a button called create. This might be interpreted as if the user hear is able to create a new types file. This is not so!!

The user must first enter the menu under Setup>Set Type Definition Packages. Here a name of the types must be added. It is therafter possible to edit it through the first menu mentioned

2.3 General impression

The examples point to the two general categories of bugs that I have found, lingual and logical.

The above problems are only small pieces in a very big program and one might argue that these small pieces are nonimportant. This is true for the isolated pieces themselves. The problem is that the program is littered with these small pieces and they screen the function of the program from the user.

As a beginner it is very frustrating to work with a program constructed in such a way because one never knows if the fault or missunderstanding is generated by oneself or by the program. All of the trouble that has been gone through making the interface of the program is therefore undone while it is the beginner who is mostly helped by a good interface.

The idea on which the program is based is a very good one. It is very satisfying to be able to solve ones problems in a graphical way. It makes it very easy to overview and transforms something abstract to a more graspable level.

Unfortunately the finish of the program is not good. Because of this a great deal of the otherwise very positive impression is lost and it is difficult to work with the program as the programmers thoght in the beginning.

3 Problems in the DFD

3.1 Problem

A new box, in the top Data Flow Diagram (DFD), for implementation of a DFD is opened. This is not possible in this case while I have another box in the same construction named "transform" and this is the same name that the program gives all new boxes.

At this time I'm not able to recognize the name confusion and try to solve the problem by opening a box containing a VHDL code instead. When the problem with the name confusion is discovered a new DFD is generated to replace the VHDL.

4 Different clocks for different parts if the system

4.1 The program

When compiling from SDS to VHDL code the user can chose between compiling as an asynchronous system or a synchronous system. If the synchronous version is chosen a system clock must also be chosen. It is the system clock that controls the steps in the state diagrams.

The system clock for a transform or a system is defined when compiling from diagram form to VHDL language. The SDS-VHDL compiler offers the possibility to define different clocks for different parts of the system.

4.2 The system

Our system has a master system clock with a frequency of 1 MHz. This is to fast for the system and we have therefor constructed a module clk_divide, app [7] and [13], to divide the frequency. The output of speich is intended to be used as an internal clock with the modules rot, phs, speich and transform, app [2].

When we try to compile the system with these different clocks, this is not possible. The compiler is not able to do it. Mentor Graphics, the company that has made the program is looking into this problem.

The problem is related to a bug in the program. This bug is beeing looked into by the people at Mentor Graphics. They have told us that there will not be a solution to the problem before december, when there is a release of a new version of the program. In the meantime they have suggested an alternative solution that will be tried out next week.

5 Compile for Synthesis

We have a system that has been completely compiled and simulated. When trying to compile the same system for synthesis this is not possible. We get a message concerning the type of the clock, see appendix. To avoid a previous message from a previous attempt, the clk_i signal has been relayed as an outsignal in this compilation, see above.

6 Marking objects in SDS

The way of marking what one wants to work with is not good. One has to remember what one has marked earlier and a lot of faults appear especially when working on large systems that are to large for overviewing simultaneously on the screen.

6.1 Result

The new box still has the name tony in the DFD, but in the filebrowzer the file has another name. When I try to change the name manually in the DFD I get the error message "name occupied by other user"

6.2 Conclusion

The conclusion one can draw from this is that the program is too sensitive. It is to easy to make a mistake and these mistakes give to great an impact and take to much time to clear up.

The problems we have run into so far is when an internal variable is relayed to another block outside of the block where it is being used. The compiler gives the error message: unresolved signal has to be resolved.

7 Problems with paths

If the path of the container in which the system has been constructed, is changed, the program is unable to reconstruct the old paths. This is not possible although all the files that have been generated are kept together in the same container.

The paths can off course can be reconstructed manually but this takes time to do and if the system is large this could mean hours of extra work. The manual reconstruction also demands extra learning time, and this time could be used better.

When lookong closer at the problem one finds that this might be a necessity because the environments where the program is operated is not uniform. One can run the program on Unix, HP etc. Keeping this in mind it is still not understandable why the path generation can not be done for an entire hierarchie at a time.

1 Internal-variables

We have still not been able to solve the problem with the internal-variable in a satisfactory way. The solution we have been able to come up with is shown in the diagram below. We have to take the internal-variable-signal outside of the box in which it is being used and lead it back into box again.

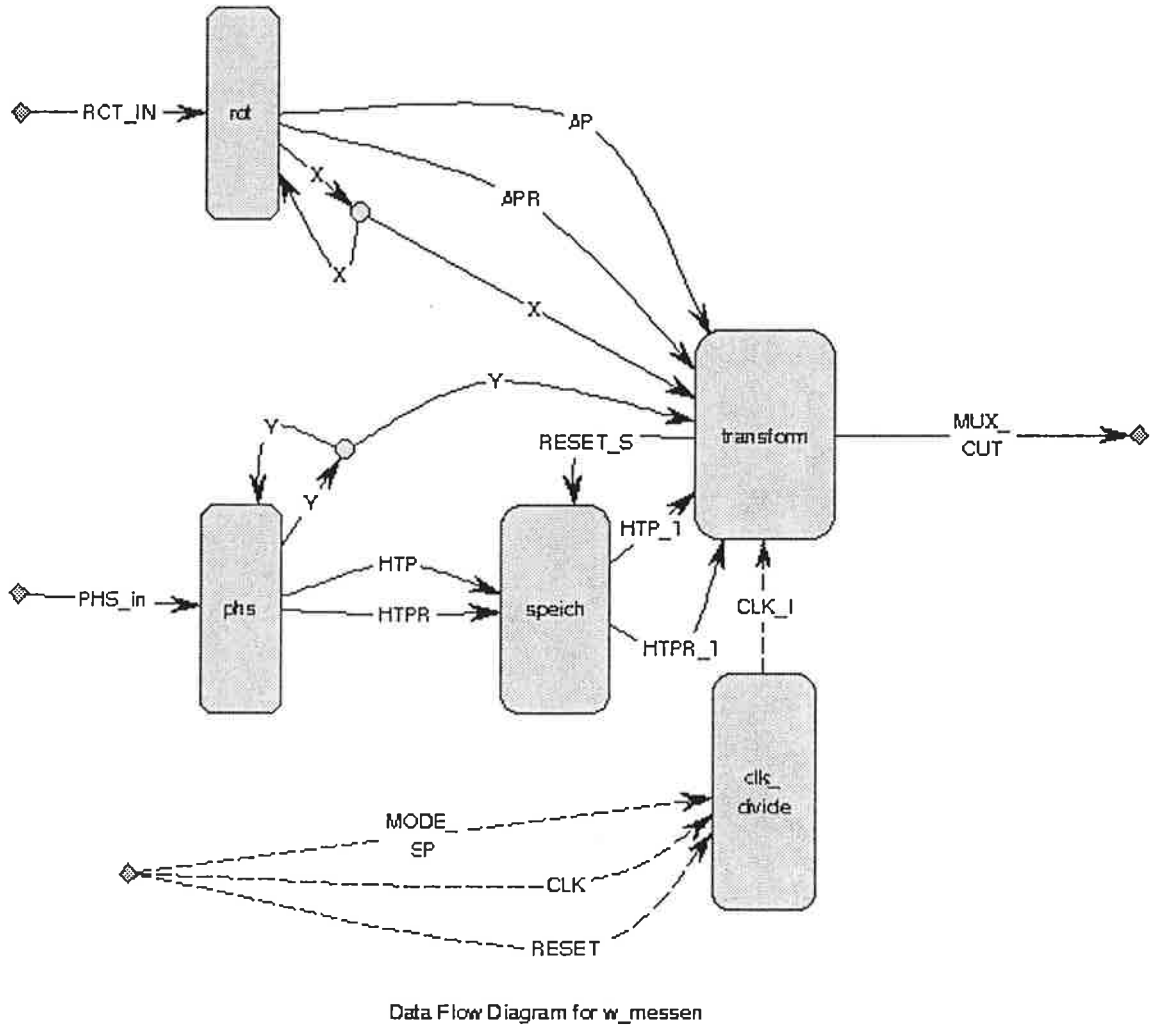


Figure 1Data-flow-diagram for mixing the two signals ROT_IN and PHS_IN.

Using this solution from the last problem (Report w2) we have run into new problems. When compiling the graphics to VHDL-code the program ads BUS to the last line in the Entity (see below)

```
-- Component : phs
--
```

```
-- Generated by System Design Station version v8.2_5.3 by wegerer on 29.09.93
--
-- background_clock :: sys_clk
-- background_reset :: sys_rst
-- Source views :-
-- /tmp_mnt/net/node_4a609/vol1/local_med/wegerer/ar.sp/w_messen/w_messen/data_flow
--  /tmp_mnt/net/node_4a609/vol1/local_med/wegerer/ar.sp/w_messen/w_messen/types/header
-- $MGC_SYS1076_STD/standard/header
--
LIBRARY work_sds ;
USE work_sds.types.all;

ENTITY phs IS
PORT (
PHS_in : IN bit;
sys_clk : IN sys_clk_type;
sys_rst : IN sys_rst_type;
HTP : OUT bit;
HTPR : OUT bit;
Y : INOUT integer BUS
 );
END phs ;
```

Figure 2Code generated from the graphical description-program SDS to the VHDL-format

This is done by the program because it believes that Y must consist of multiple signals. When compiling this code to simulatable code the compiler does not know what value or kind of value to assign Y, while it is defined as a multiple-signal variable in the Entity-code, but used as a single-signal variable in the Architecture-code. The best way of solving this problem so far is to edit the file and taking away the BUS word by hand.

2 Relaying an internal variable

The problems we have run into appears when an internal variable is relayed to another block outside of the block where it is being used. The compiler gives the error message: unresolved signal has to be resolv

This is a problem that occurs because VHDL is a concurrent programming language and with this kind of construction the variable is able to be assigned two different values at the same time. This must be resolved with a resolve function or differently constructed.

In our case the signal is of type integer and there is no standard function in VHDL for resolving it. The only standard function for resolving signals defined in VHDL is for bit type signals.

This has been solved so far by dividing the variable into two variables. One that is used internal and one that is given the value of the first one and relayed outside when the outside is dependant of this information.

3 Complex-variables

When writing conditions in transition-boxes in the state-machine-mode that the program views as complex action, these conditions will not be checked by the program. They will be translated to VHDL-code exactly the way they are written in the transition-box. In our state-machine for the rot and phs boxes, (see Appendix), we have 19 different such transitions.

This makes it important to be somewhat familiar with VHDL-code in order to use SDS comfortably and to a usable extent.

4 Type-definition

I tried to open a new type-definition file by using Open<Typesdefinition.. and in the popupmenu writing the entire address of a presumed types file, "wegerer/ar.sp/w_messen/types/types". This did not work and I did it all over again this time only writing "types" in the dialog-box. This was accepted by the program and I was able to define my types.

When I later tried to open a new state-machine I was not allowed to do so by the program, because it couldn't find all my type-definition-files. When I tried to delete the faulty type-definition-file this was not possible. My only way of solving this problem was to throw away all my work and begin all over again!

5 Run in Q-sim mode

If run is commanded in Q-sim mode without giving a time, the simulator goes into an eternal loop.

6 New VHDL compilation

When one wishes to recompile VHDL-code, the always button has to be pressed. If this is not done the VHDL-code will not be recompiled at all. There should be an easier solution to this problem than what is implemented today.

7 Appendix

1. State Transition Diagram for rot
2. State Transition Diagram for phs

1 Overwriting existing VHDL code

1.1 Background

The report of this week is a product of the workings of the SDS program. The part I have produced this week is only defined by the VHDL code and there is subsequently no State Machine definition. This causes a problem when the entire system defined by a numerous of different State Machines is to be tied togheter into one working system.

1.2 Problem

The only way I have found so far for tying different parts togheter into one working system is to generate new VHDL code for the entire system and thereby creating the correct paths for the system. If this is done when one of the parts is defined as a VHDL code the program will generate a new VHDL code for it as well overwriting the old one. This can be avoided if a different name is choosen than the standard one suggested by the program when generating the original code. The program always gives the generated VHDL codes the same name no matter if the code is generated to define a function or whether the code is generated to create the proper paths to make the bigger unit functionable.

If one avoids the first problem with the overwriting one still has the problem of having to tell the program which of the two versions of code is to be the one used. If this is neglected it could give the user a lot of unnecessary agony before the problem is found.

1.3 Conclusion

It is a weakness in the program that man made code can be overwritten by accident when generating code for tying the different program parts togheter. These two functions should be separated in future versions of the program or the computer should at least generate different names in the two cases described above.

1 Counter in System Design Station

The program SDS has been studied from two different views. Top-down construction and VHDL direct coded.

1.1 Top-down construction.

1.1.1 Problem

Our aim was to construct and simulate a system that could count, be asked not to count, resume counting and be reset.

1.1.2 Method

We began by building a contextdiagram and in it specifying in- and out-signals to our system. The input is a control-signal (control) which can assume three states, (count, hold and reset). The output ("stand") is an integer. It looks odd that the output is relayed back into the system but this finds its explanation further down in the paragraph concerning the state-machine.

In this small example no further analyzes is conducted in the level Data Flow Diagram.

The State Transition Diagram or State Machine (SM) describes a simple counter. It has three different modes, (count, hold and reset). If one wishes to use "stand" as an internal variable (count_state, State Transition Diagram for transform, App 3) one needs to relay it back into the system as described in paragraph one. The program does not accept that internal variables are defined in SMs.

In order to accept the system for simulation it is compiled to VHDL-language (VHDL code for sm, App 5). In this example the ability to write VHDL-code directly is not used.

1.1.3 Result

In the simulation it is evident that we have accomplished the intention of the exercise. The simulation shows how the SM reacts to different control-signals while situated in different states.

2 VHDL direct coded

2.1 Problem

To prove that a problem can be solved with a more direct approach, VHDL-code is written directly and simulated. The problem is chosen much less complicated than the above, but I believe the possibilities of the VHDL-code in the SDS-program is somewhat presented. A nand-gate is simulated, this only requires one line of code in the architecture-program-part, the rest is for administrative purposes.

2.2 Method

As in all programming with SDS the beginning is the Context Diagram and the Data Flow Diagram. Beneth these two levels the VHDL-code is implemented. All the code presented in the appendix except the one line describing the nand-gate is generated by the program.

2.3 Result

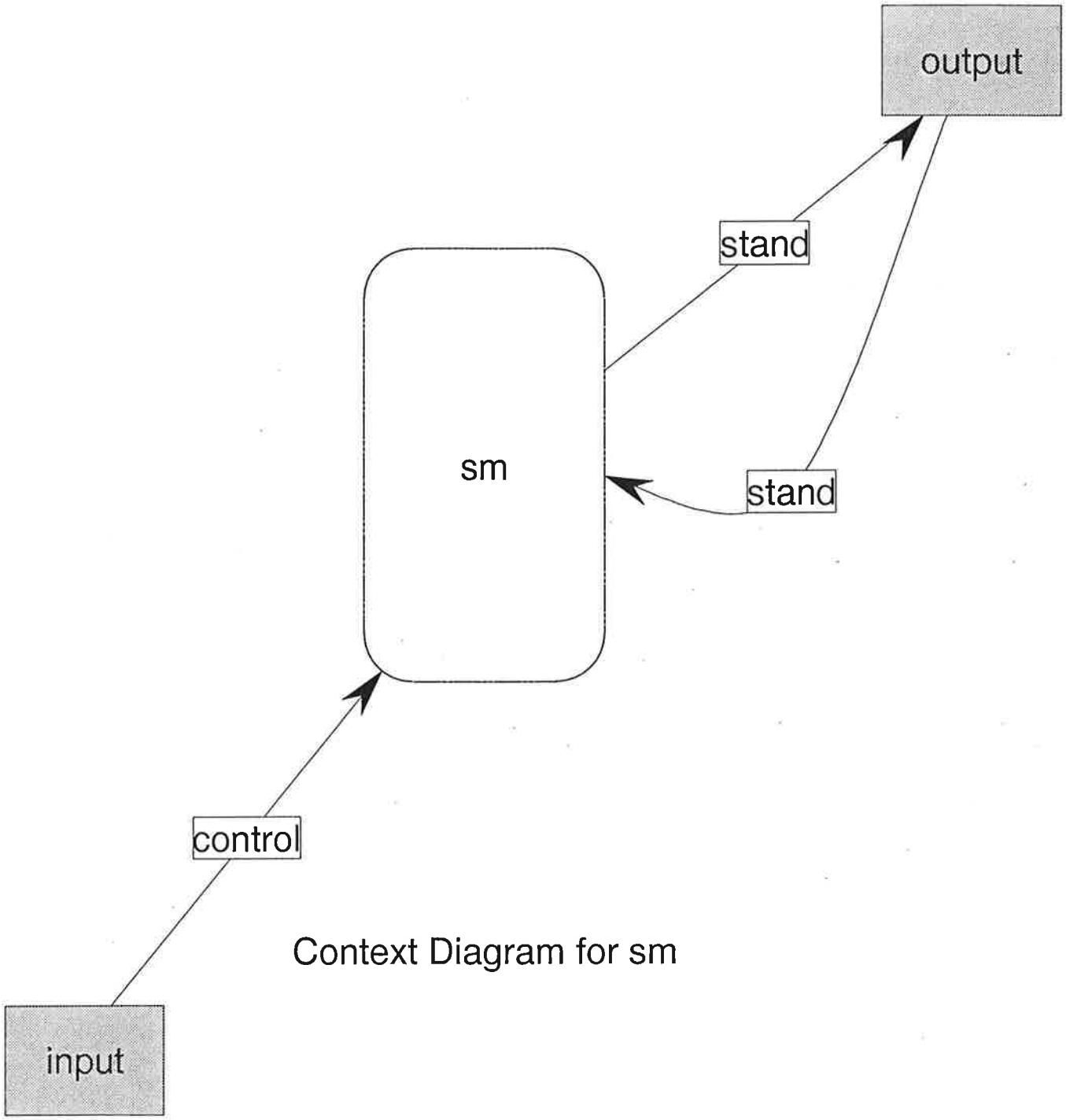
In the simulation it shows that the program reacts as one could expect a nand-gate to react. What is of interest is that it is possible to simulate the propagation-time through the nand-gate.

3 Summary

It is not possible to have an internal variable in a SM. The SDS is a powerfull tool for writing VHDL-code in because it generates a great deal of the administrative code by itself.

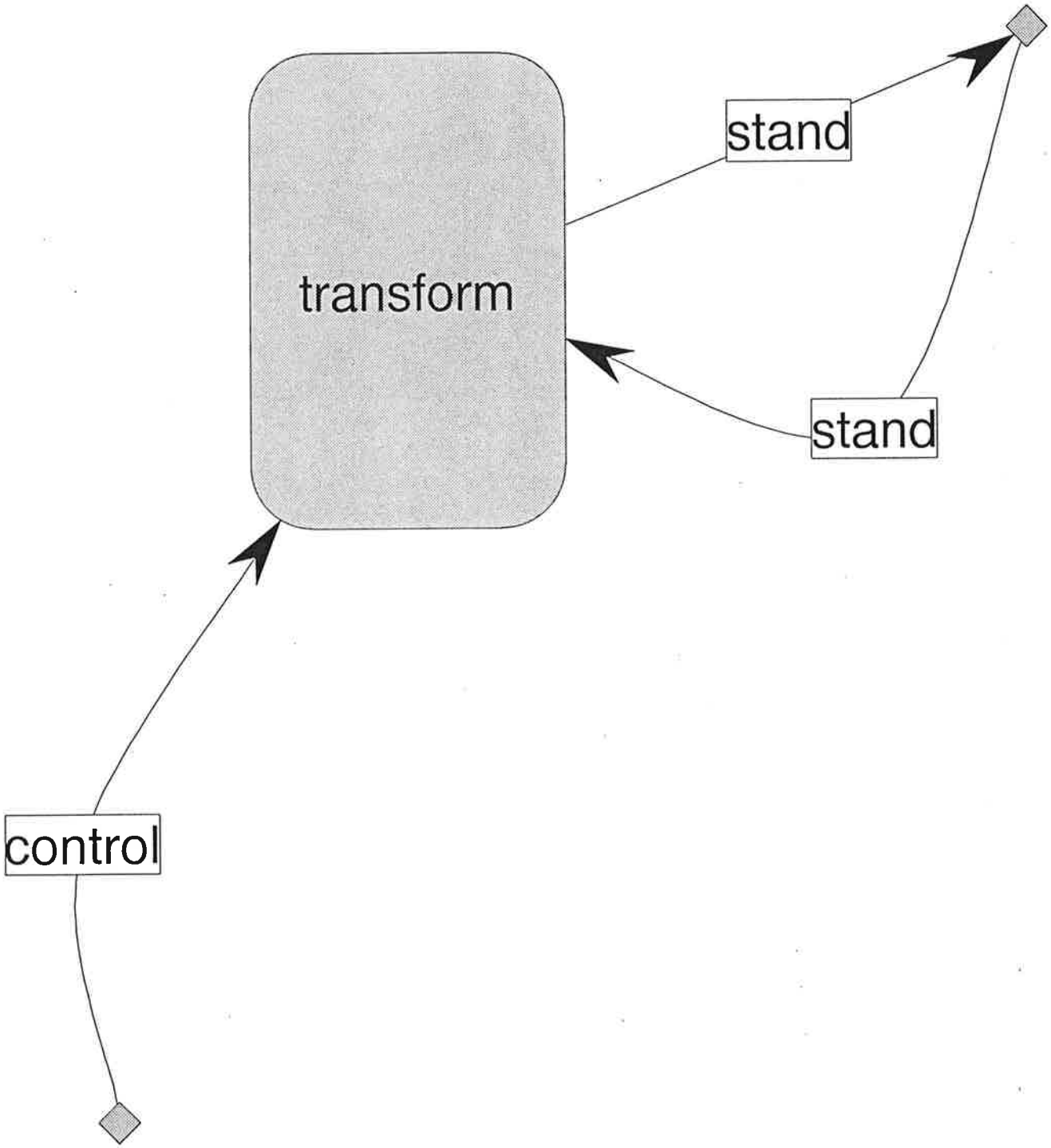
4 Appendix

[1] Context Diagram for sm
[2] Data Flow Diagram for sm
[3] State Transition Diagram for transform
[4] Simulation of sm
[5] VHDL code for sm
[6] Simulation of nand-gate
[7] VHDL code for nand-gate

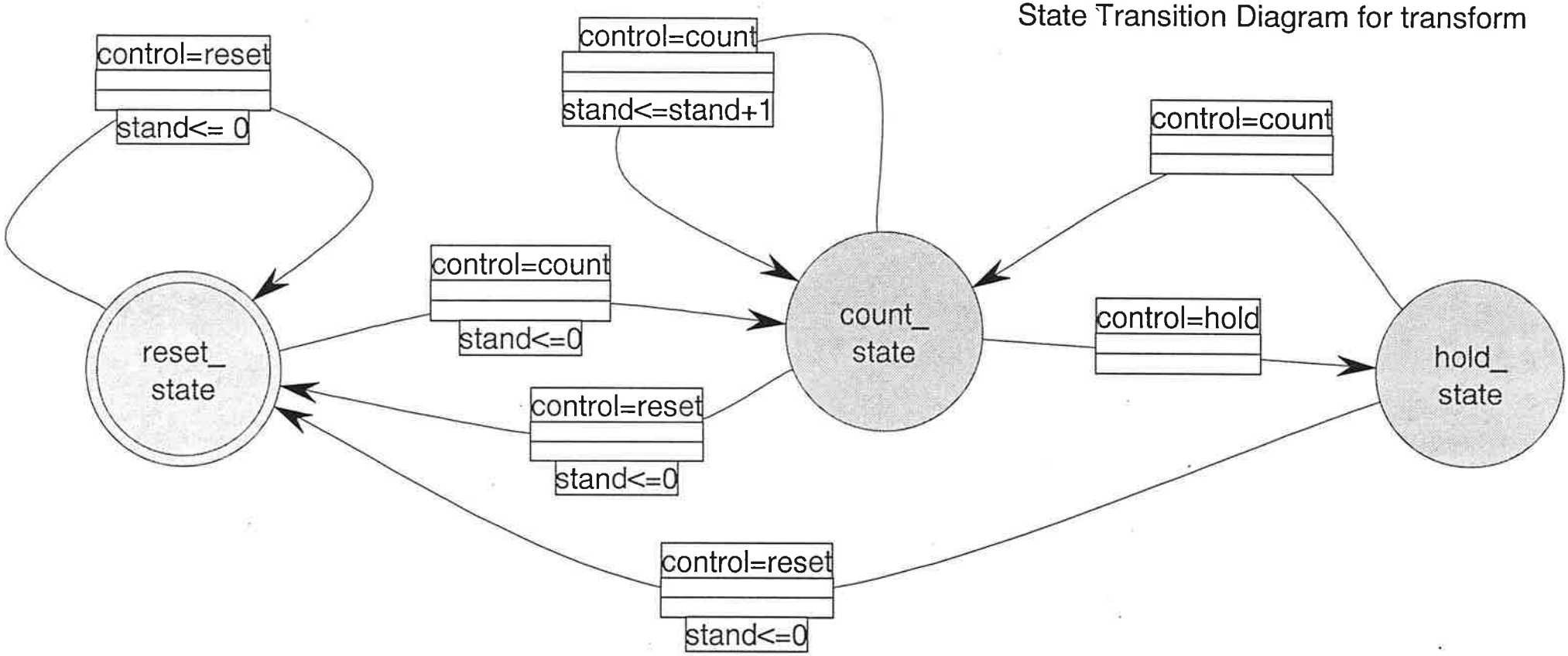


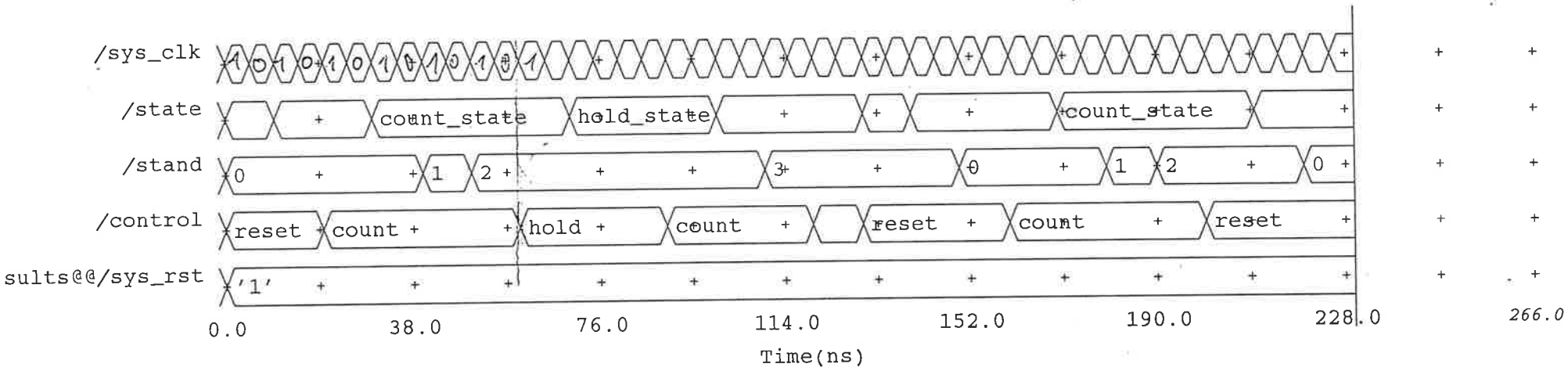
Context Diagram for sm

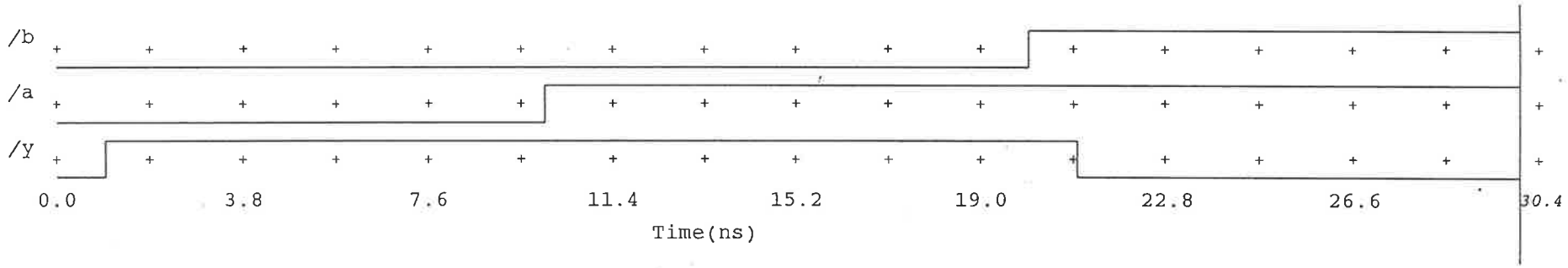
Data Flow Diagram for sm



State Transition Diagram for transform




/sys_clk
/state
/stand
/control
sults@@/sys_rst
count_state
hold_state
count_state
0
1
2
3
0
1
2
0
reset
count
hold
count
reset
count
reset
'1'
0.0
38.0
76.0
114.0
152.0
190.0
228.0
266.0
Time(ns)


/b
/a
/y
0.0
3.8
7.6
11.4
15.2
19.0
22.8
26.6
30.4
Time(ns)

1 Comments on VHDL

1.1 Course VHDL

This week I have taken a course in VHDL. The course was nice and well carried out. In one week it is possible to be familiarized with a major part of the syntax. This demands prior experience with another programing language. My experience is that such a course is necessary if efficiency with VHDL should be reached in reasonable (economic) time.

1.2 SDS - VHDL

An understanding of VHDL is important if, work with SDS is to be carried out in any greater extent. SDS offers a nice programing environment, but if anything with a small complication level is to be constructed one must be able to understand and write VHDL code.

1.3 VHDL

The resemblance to other programming languages is striking when looking at VHDL for the first time. The most important difference is that VHDL allows concurrent programing. In hardware, concurrence is common and this is the reason why VHDL supports concurrence.

1.3.1 VHDL compiler environment

1.3.1.1 General impression

The compiler/editor environment is in great parts good. The overall impression is that the programmers ambition has been to support every known and a couple of unknown functions in VHDL. This gives a design environment that is very difficult to overview. My experience is that there is no collision of interest between the two, but in this case, the problem has not been solved.

These problems are not any trouble once the user has grown accustomed to the environment. But this is not optimal since one of the big points with a user friendly interface is that it should be easy to learn and this effect is somewhat lost.

1.3.1.2 Editor

The editor is not a bad one but is it not a great one either. There is a nice vhdl-palette on the side of the sheet, where the work is done. The user accesses the different functions by clicking on text lines that describe the functions. It has some nice functions, for example a function that writes the correct syntax of a given function on the worksheet. It would help the accessibility if the palette was controlled by icons.

When writing code the only support is the palette mentioned above. My experience with editors is that functions like the one mentioned above come to better use if they are incorporated in the program automatically.

The default of the program when delivered should have functions like the one mentioned above implemented and the flexibility of the program should lie in the ability to manipulate such functions and be able to shut them of.

1 Comments on Quicksim and Design Architect

1.1 The simulator

In Qsim the designer is able to test the design concerning different input signals and the specification limits. Qsim has many functions of which most are incomprehensible for the beginner. The user has to work quite some time with the program in order to understand it and get full use of the majority of the functions. Once the program is learned it is a good tool for the purpose it was designed for.

1.1.1 Dofiles

The simulator supports dofiles. These are macron written in ordinary text files and executed with the command "dofile".

In Qsim there is a function called "transcript window". In the "transcript window" one finds the commands used by the program itself in writing. The program will respond to these commands in the same fashion as it responds to commands selected in the menus.

The user is able to write these commands in a file and execute them by simply writing "dofile" and the file name in the trace window. This is extremely helpful in the simulations, while the user is able to repeat the same simulation many times and making small changes every time. It saves a lot of time because a simulation may involve a lot of commands to be useful.

1.2 Problems with compilation in Design Architect

I have now found another one off all the peculiarities that seems to be built in the mentor program package.

I have generated the outlines of a VHDL code under a DFD. I then edit it in Design Architect. When I try to compile the code in Design Architect I get the error message " undefined variable " of one of the signals that I have designed for to be an outsignal. In the entity code SDS has designated this signal to be an outsignal.

When I compile the code in SDS I have no such problems. The code is compiled without a statement!

There seams to be something rotten in the program of Mentor!

1 Comments on Autologic

1.1 MentorGraphics Autologic

1.1.1 Description

A look at Alteras Autologic has been taken. Autologic has two major functions. The first is transforming a VHDL code, to a gate level description of the problem. This is called a schematic of the problem. The second is to map these schematics to the target technology and optimize according to a set of parameters. The Autologic allows optimization according to area and speed.

The optimization is controlled by so called recipies. These recipies are composed by the user combining different options such as (high - low) area and (high - low) speed. One comand can be used many times and the output is not the same whether the command is used one time or twice in a row.

1.2 MaxPlus2 from Altera

1.2.1 Description

With MaxPlus2 the user fitts the netlist produced by Autologic to the Altera chip. The user is able to choose which chip he or she wants to fit the construction in. It does not offer as many possibilities for optimization as the Autologic, but this is not necessarally any drawback. The philosophy of the Altera programmers seems to by that they understand the workings of their chip better than others and that is difficult to argue.

1.3 Going through Autologic and MaxPlus to the Chip

1.3.1 Description

The autologic and maxplus are two programs for optimizing and fitting the VHDL code to the the Chip. There is a certain redundance in the functionality concerning the optimization. One can choose to do the optimization in either Autologic or MaxPlus. If one chooses MaxPlus time is saved.

Nomatter how one chooses to do the optimization, the optimization mode in Autologic must be used. This has to do with that the optimization mode includes mapping.

1.3.2 Small evaluation of Autologic

Different recipies for the optimization of a testobject has been tried. They have been optimized in Autologic and fitted in a F8000 series chip choosen by the MaxPlus program. The MaxPlus program has in all the different cases choosen the EPF81188MC240 chip. The result shows that the best efficency is reached when not trying to optimize in the Autologic program att all.

The area occupied on the chip varies between 48 - 55%. The variation is not very big, and the fact that no optimization yields a better result than a hevily optimized one is a bit confusing.

This is a very discouraging result. Our goal is to fit the entire construction ROSY in this chip and it is many times larger than the example worked with here.

2 Appendix

[1] DFD diagram w_messen

[2] VHDL code for tony

[3] Specifications tony

[4] Simulation of tony

1 Specifications Control

Clock frequency: 500 kHz.

Input signals: AP (bit), Apr (bit), Htp_1 (bit), Htpr_1 (bit), Mode_Sp (bit), XS (integer)

AP triggers transform.
Apr triggers transform.

Htp_1 triggers transform when Mode_Sp is '1'. When Mode_Sp is '0' it will not be processed until an Ap or Apr signal has been processed.
Htpr_1 triggers transform when Mode_Sp is '1'. When Mode_Sp is '0' it will not be processed until an Ap or Apr signal has been processed.

Mode_Sp chooses the mode. When low the mode is set to mux and when high the state machine lets through whatever signal is high.

XS is controlled by rot and used as external clock. It has values $112 < X < 312$ and is counted up once every clock cycle. It controlls the length of the Ap and Apr singals on Mux_Out.

Output signals: Mux_Out (bit), Reset_S (bit).
Reset_S is set high when transform is ready for a new Htp_1 or Htpr_1

Ap_akt is set when Mux_Out is set high by an Ap signal.
Apr_akt is set when Mux_Out is set high by an Ap signal.
Htp_akt is set when Mux_Out is set high by an Htp_1 signal.
Htpr_akt is set when Mux_Out is set high by an Htpr_1 signal.

Mux_Out is the out signal of transform. It has a protocoll according with the table below.

Signal	Time high	Time low
Ap	200 us	100 us
Apr	250 us	150 us
Htp	100 us	150 us
Htpr	150 us	100 us

Internal variable: Z (integer)
Z is counted up to 125, one count every clock-cycle and controlls the length of the Htp and Htpr signals on Mux_Out.

Cycle time: 650 (us)

Delay time: 2 (us)

Verbal description:

The four inputs to the state machine are Ap, Apr, Htp_1 and Htpr_1. The first two are generated by the state machine Rot. The last two are generated by the state machine Speich. If mode_sp is true all four signals can trigger the state machine. If mode_sp is false the state machine can only be triggered by the Ap or Apr signals.

If the state machine is triggered by Htp_1 or Htpr_1 the state machine will go to the htp_state or htpr_state correspondingly. In the transition Mux_out will be set true. The Mux_out signal will remain true for 100 respectively 150 microseconds. After the Mux_out signal is set false the state machine will remain in the state for another 150 respectively 100 microseconds. It will then return to start_state.

If the state machine is triggered by an Ap or Apr signal it goes to the Ap_state or Apr_state correspondingly. In the transition the signal Mux_out is set true. This is set false 200 respectively 250 microseconds later. The timing is controlled by the signal Xs. This is a signal generated by the state machine Rot. The states Ap_state and Apr_state are left after 200 respectively 150 microseconds.

After the Ap and Apr states three different transitions are possible. If no Htp or Htpr signal is registered the state machine returns to start state. The conditions for this transition is that the Htp_1 and the Htpr_1 signals are generated by the state machine speich are false.

If Htp_1 is true the state machine continues to the htp_state. In the transition Mux_out is set true and remains so for 100 microseconds. The transition where Mux_out is set false is controlled by an internal counter, Z. The state machine remains in the htp_state for 150 microseconds whereafter it returns to the start_state.

If Htpr_1 is true the state machine transitions to the htpr_state. The transitions in this state are also controlled by the counter Z. There is no conflict in this since only one of the two states are in use simultaneously. Mux_out is set true in the transition to htpr_state and set false 150 microseconds later. The state machine remains in the htpr_state for another 100 microsecond whereafter it returns to the start_state.

1 Specifications Phs

Clockfrequency: 500 kHz.

Input signals: Phs_in (bit), Htpr_1 (bit), Htp_1 (bit)

Phs_in triggers the system.

Htpr_1 gives information if the Phs signal has been blocked in the mixing process.
Htp_1 gives information if the Phs signal has been blocked in the mixing process.

Output signals: HTP (bit), HTPR (bit), Phs_Lang (bit) and Phs_Kurz (bit).

HTP is 1 clock-cycle long and set high when $128 \leq$ Phs_In ≤ 256 (us).
HTPR is 1 clock-cycle long and set high when $256 <$ Phs_In < 450 (us).

Phs_Lang is set to '1' when $450 \leq$ Phs_In (us).
Phs_Kurz is set to '1' when Phs_In ≤ 127 (us).

Htp_n_m is set when Phs_in has been detekted as a signal (after 128 us) and Htp_1 or Htpr_1 are still high.

Internal variable: Y (integer).
Y is counted up to 312, one count every clock-cycle.

Cycle time: 650 microseconds (325 clockcycles).

Delay time: 256 microseconds (127 clockcycles).

Verbal description:

Phs decodes which signal the patient handling system sends. The two signals are Htp and Htpr see 2.1.2. It also detects if the signal is within specified boundaries. The errors detectable are a too short signal and a to long one see [14] Specification for Phs.

Phs is implemented as a state machine, see [22] State Transition Diagram for Phs. The main part of the state machine is a counter. It is defined as an integer for purpose of simplicity. It is counted up once every clock cycle until a too short signal is detected or it has counted to 324.

The state machine leaves the start_state when Phs_in is true. If Phs_in is false between 32 and 127 microsecond after initiation, (16 to 63 clock cycles), the state machine goes back to start_state. This is to sort out too short signals that might be caused by faults in the system.

If Htp_1 or Htpr_1, outputs from Speich, are true 128 microseconds after initiation, the error message Htp_n_m is set true. This tells the master that the previous Htp or Htpr is blocking the next state machine, Speich. Therefore the signal in the Phs state machine will not be sent on to the speich state machine and will be lost.

After 256 microseconds, 128 clock cycles, the state machine makes the transition to the count_state. In the transition the state of the Phs_in signal is checked. If Phs_in is true Htpr is set true and if it is false Htp is set true.

After 450 microseconds the state of the Phs_in signal is checked again. If it is true the error message Rot_lang is set true. The state machine returns to the start_state when the counter has counted to 324, 650 microseconds after it begun, and a full cycle is completed.

1 Specifications Rot

Clock frequency: 500 kHz.

Input signals: ROT_IN (bit)

Output signals: AP (bit), APR (bit), Rot_Lang (bit), Rot_Kurz (bit) and, XS (integer).

AP is 1 clock-cycle long and set high when 128 ≤ Rot_In ≤ 256 (us).
APR is 1 clock-cycle long and set high when 256 < Rot_In < 450 (us).

Rot_Lang is set to '1' when 450 ≤ Rot_In (us).
Rot_Kurz is set to '1' when Rot_In < 128 (us).

XS is given the value of X when X is in the range of 127 < X < 324 clock-cycles.

Internal variable: X (integer)

X is counted up to 312, one count every clock-cycle.

Cycle time: 626 microseconds (313 clock cycles).

Delay time: 256 microseconds (127 clock cycles).

Verbal description of the specification:

Rot decodes which signal is being sent from the rotating part of the Somatom. It is able to differentiate between an Ap signal and an Apr signal, see 2.1.1. It is also able to detect certain errors in the transmission. The detectable errors are a too short input, and a too long input, see [15] Specification for Rot.

Rot is implemented as a state machine, see [23] State Transition Diagram for Rot. The main part of the state machine is a counter. It is defined as an integer for purpose of simplicity. The counter is counted up once every clock cycle until a too short signal is detected or it has counted to 324, (650 microseconds).

The state machine leaves the start_state when the Rot_in is true. It returns to start_state if Rot_in is false between 32 and 127 microsecond after initiation, (16 to 63 clock cycles). After 256 microseconds the state machine checks if Rot_in is true or false and goes to the count_state. If Rot_in is true the output signal Apr is set true and if it is false the Ap is set true. In the count_state the Ap or Apr signal is reset to false.

The counter counts to 324. An output XS is set to the value of the integer defined counter X. The XS signal is used as an input by the control state machine. In System Design Station it is impossible to use an internal variable in one state machine as an input in another state machine. Therefore the user must use a construction such as the one shown in the count_state.

If Rot_in is true after 450 microseconds the error message Rot_lang is set true. The counter continues to count until 324. This is the condition for going back to the start_state.

1 Specifications Speich

Clock frequency: 500 kHz.

Input signals: Htp (bit), Htpr (bit), Reset_S (bit)

Htp triggers the system.
Htpr triggers the system.

Reset_S tells the system that transform is ready for a new reading and resets the system.

Output signals: Htp_1 (bit), Htpr_1 (bit)

Htp_1 is set high when Htp is high and reset when Reset_s is high and Htp is set back to low.
Htpr_1 is set high when Htpr is high and reset when Reset_s is high and Htpr is set back to low.

Delay time: 2microseconds (1 clock cycle).

Verbal description:

Speich is implemented as a state machine, see [24] State Transition Diagram for Speich. The state machine leaves the start_state when either Htp or Htpr are true. They are generated as outputs by the Phs state machine. If Htp is true Htp_1 is set true in the transition to the htp_1_state and if Htpr is true Htpr_1 is set true in the transition to the htpr_state. The next two transitions in both branches are triggered by the same signals. The first transition is made when the Reset_s signal is true. Reset_s is an output signal from the Control state machine. The Control transform sets Reset_s true when it is ready for a new value from speich.

The transition back to start_state is done when Htp or Htpr is set back to false. If the state machine was triggered by Htp, it is the setting of Htp to false, that triggers the transition and vice versa. This transition is there because if the resetting of the signal is not registered the state machine will run around as long as Htp or Htpr is true and tell the Control state machine that numerous Htps or Htprs have arrived for every signal that actually arrives.

1 Specifications Tony

Clockfrequency: 500 kHz.

Function: The program implements nine different flip-flops. Each flip-flop has two input signals and one output signal. The input signals are, the error message signal from the process and the reset signal coming from the controlling circuit outside of this process. The reset signal tells the tony process that it has read the error message and is ready for a new message. The output is set high by a rising flank of an error message and reset by the input _Res.

Input signals: Reset_vektor (8 downto 0, bit vector), Phs_Kurz (bit), Phs_Lang (bit), Rot_Lang (bit), Rot_Kurz (bit), Ap_Akt (bit), Apr_Akt (Bit), Htp_Akt (bit), Htpr_Akt (bit), Htp_n_m (bit).

Output signals: Result_vektor (8 downto 0, bit vector).

Cycle time: 4 microseconds (2 clock cycles).

Delay time: 4 microseconds (2 clock cycles).

Verbal description:

Tony is implemented as nine different flip flops, one for every signal. It reacts to the positive flank of an error message and gives an output signal on the bus Result_vector. This is read by the Master who resets the flip flop through the bus Reset_vector. The flip flops are described in the code as seen in Figure 20. The FOR loop is executed every clock cycle.

The construction of the code relies on the concurrence of the VHDL programming language. The assignments that are made are only updated after the FOR loop is run through. Therefore it is possible to have a construction as the one on lines 3 and 4. In an another strictly sequential programming language the statement in the FOR loop would always be true.

Here the value of 'Vektor_int(n)' is assigned to 'Int_vektor(n)' on line 3. The variable 'Int_vektor(n)' is not updated until the loop is run through completely and a new clock cycle is current. On line 4 the condition is true under the circumstances that 'Vektor_int(n)' was true when the execution of the FOR loop was begun. This is also the description of a positive flank of a signal. The reset is done in the IF command on lines 7 and 8.

```
1 FOR n IN 0 to 8
2  LOOP
3      Int_vektor(n) <= Vektor_int(n);
4          IF Vektor_int(n) ='1' AND Int_vektor(n) = '0' THEN
5              Result_vektor(n) <= '1';
6          END IF;
7              IF Reset_vektor(n) = '1' THEN
8              Result_vektor(n) <= '0';
9          END IF;
10 END LOOP;
```

1 Specifications ω_messen

Clock frequency: 1 MHz.

Input signals: Rot_In (bit), Phs_In (bit), Mode_Sp (bit), Clk (bit), Reset (bit)

Rot_In triggers ω_messen. It has the constraints 128 (us) < Rot_In < 450 (us). The signals may not be closer than 650 (us). If Rot_In is shorter than 256 (us) ω_messen detects it as an Ap signal and if it is longer than 256 (us) ω_messen detects it as an Apr signal.

Phs_In triggers ω_messen when Mode_Sp is '1'. When Mode_Sp is '0' a Rot_In signal is needed to let the Phs_In signal through. It has the constraints 128 < Rot_In < 450. The signals may not be closer than 650 (us). If Phs_In is shorter than 256 (us) ω_messen detects it as an Htp signal and if it is longer than 256 (us) ω_messen detects it as an Htpr signal.

Mode_Sp decides whether the system operates in spiral mode ('0')or in singel axis mode ('1'). In spiral mode the two signals are muxed and in singel axis mode the signal that is active is let through.

Output signals: Mux_Out (bit), Phs_Lang (bit), Phs_Kurz (bit), Rot_Lang (bit), Rot_Kurz (bit).

Phs_Lang is set high when Phs_In is longer than 450 (us).
Phs_Kurz is set high when Phs_In is shorter than 150 (us).
Rot_Lang is set high when Rot_In is longer than 450 (us).
Rot_Kurz is set high when Rot_In is shorter than 150 (us).

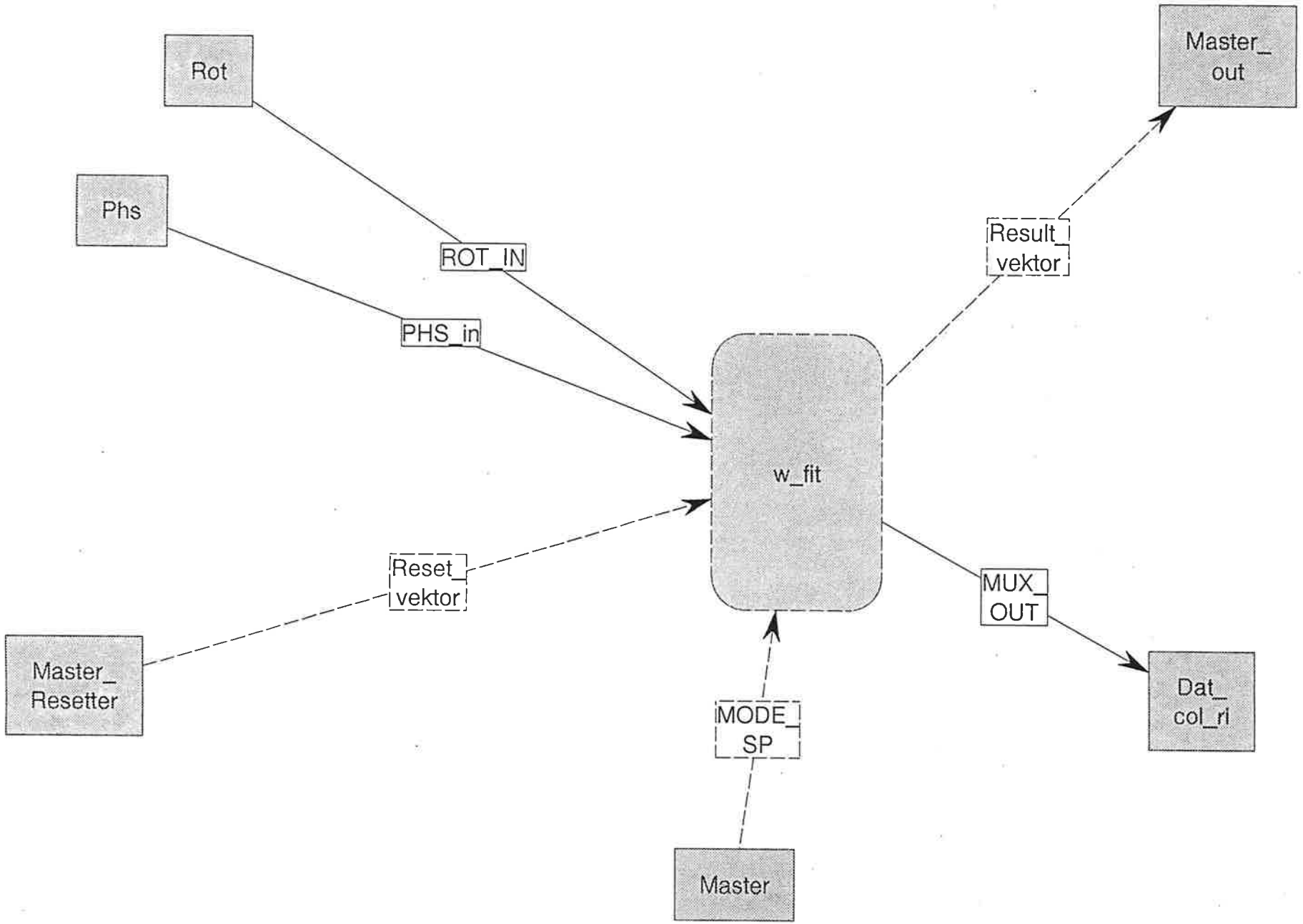
Ap_akt is set high when Mux_Out is aktivated by an Ap signal.
Apr_akt is set high when Mux_Out is aktivated by an Apr signal.
Htp_akt is set high when Mux_Out is aktivated by a Htp signal.
Htpr_akt is set high when Mux_Out is aktivated by a Htpr signal.

Htp_n_m is set high when a Phs_in arrives and the Phs-in that arrived before has not been processed yet.

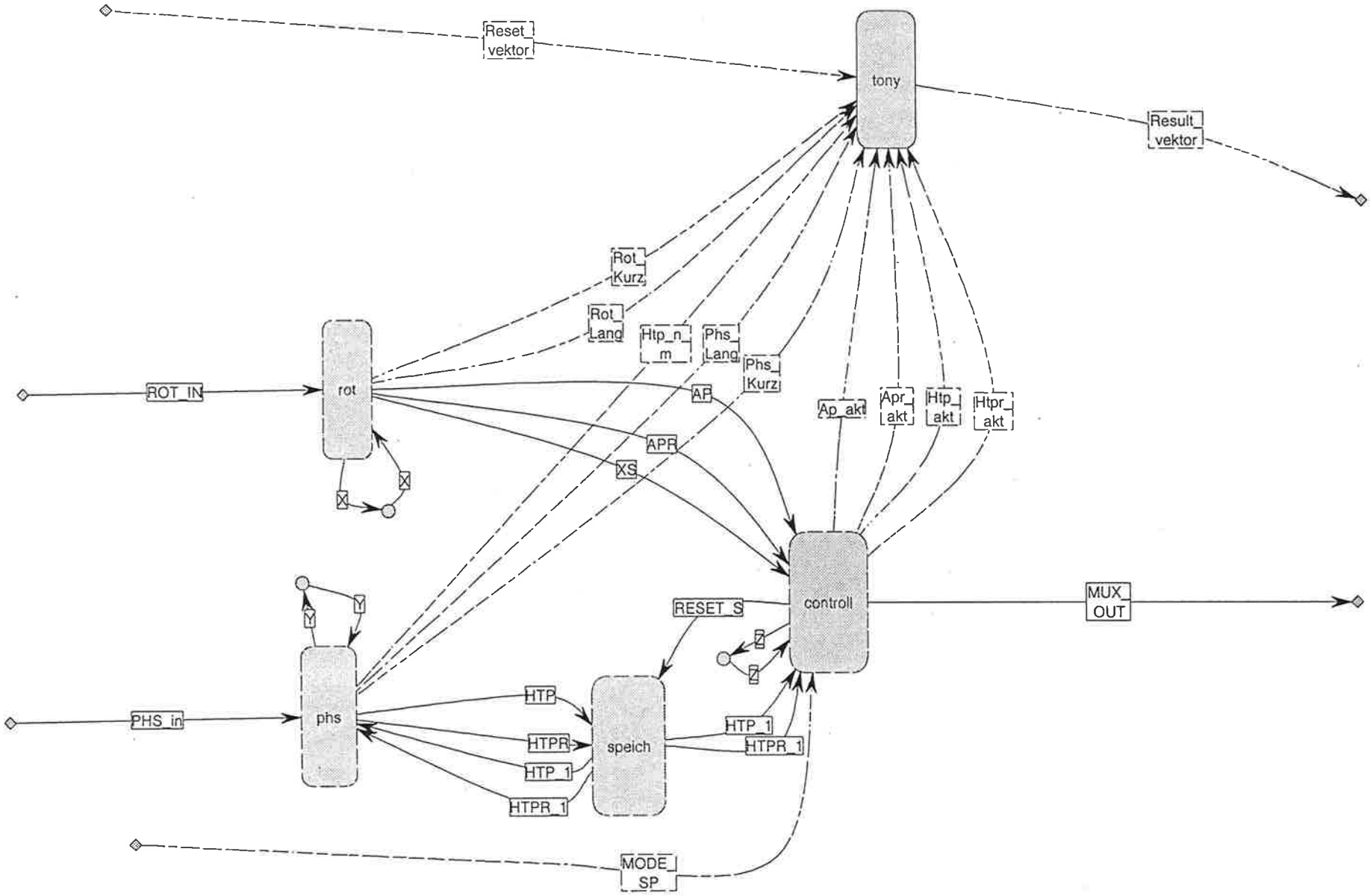
Mux_Out is the out signal of ω_messen. It has a protocoll according with the table below.

Signal	Time high	Time low
Ap	200 us	200 us
Apr	250 us	150 us
Htp	100 us	150 us
Htpr	150 us	100 us

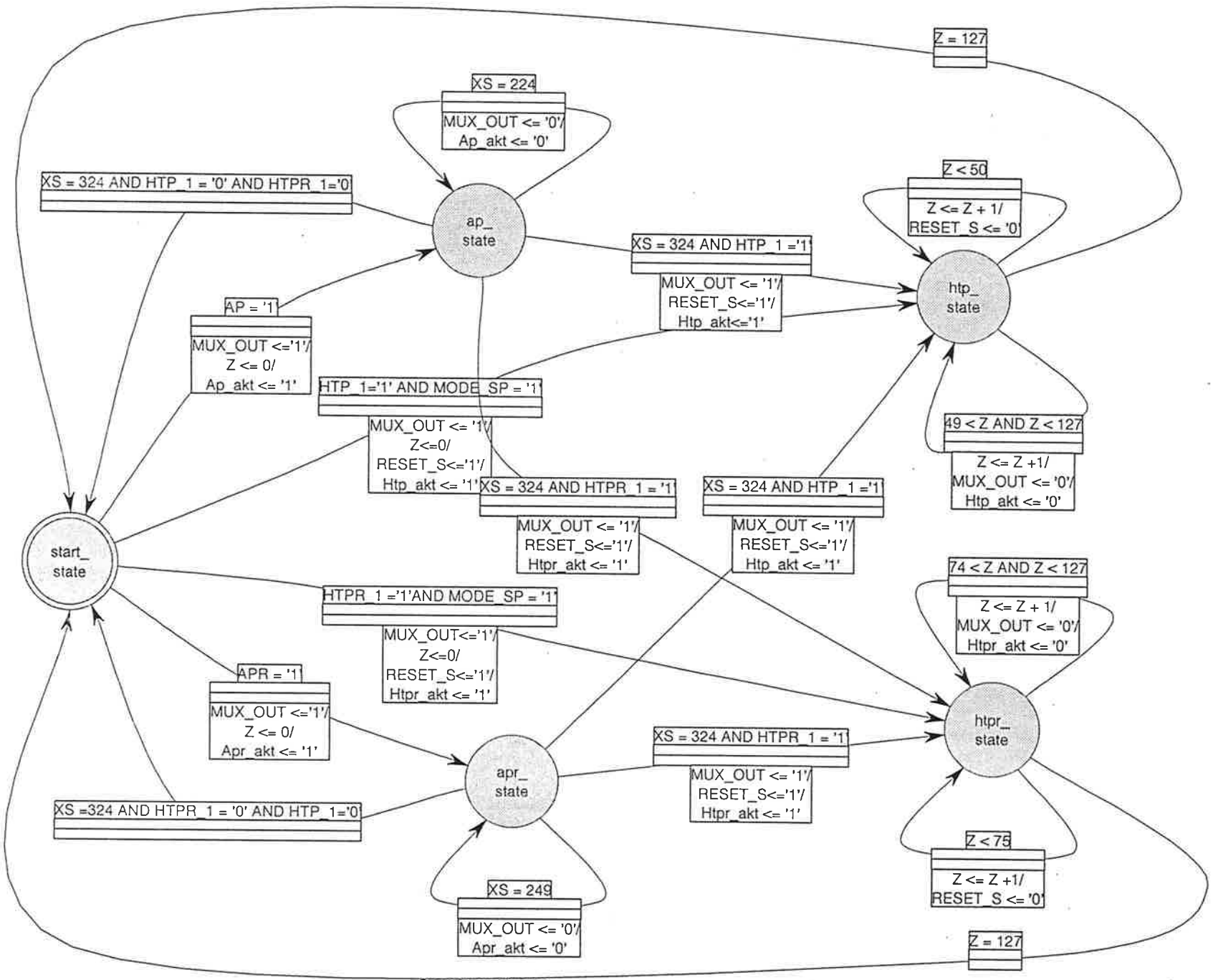
Cycle time: 650 (us)



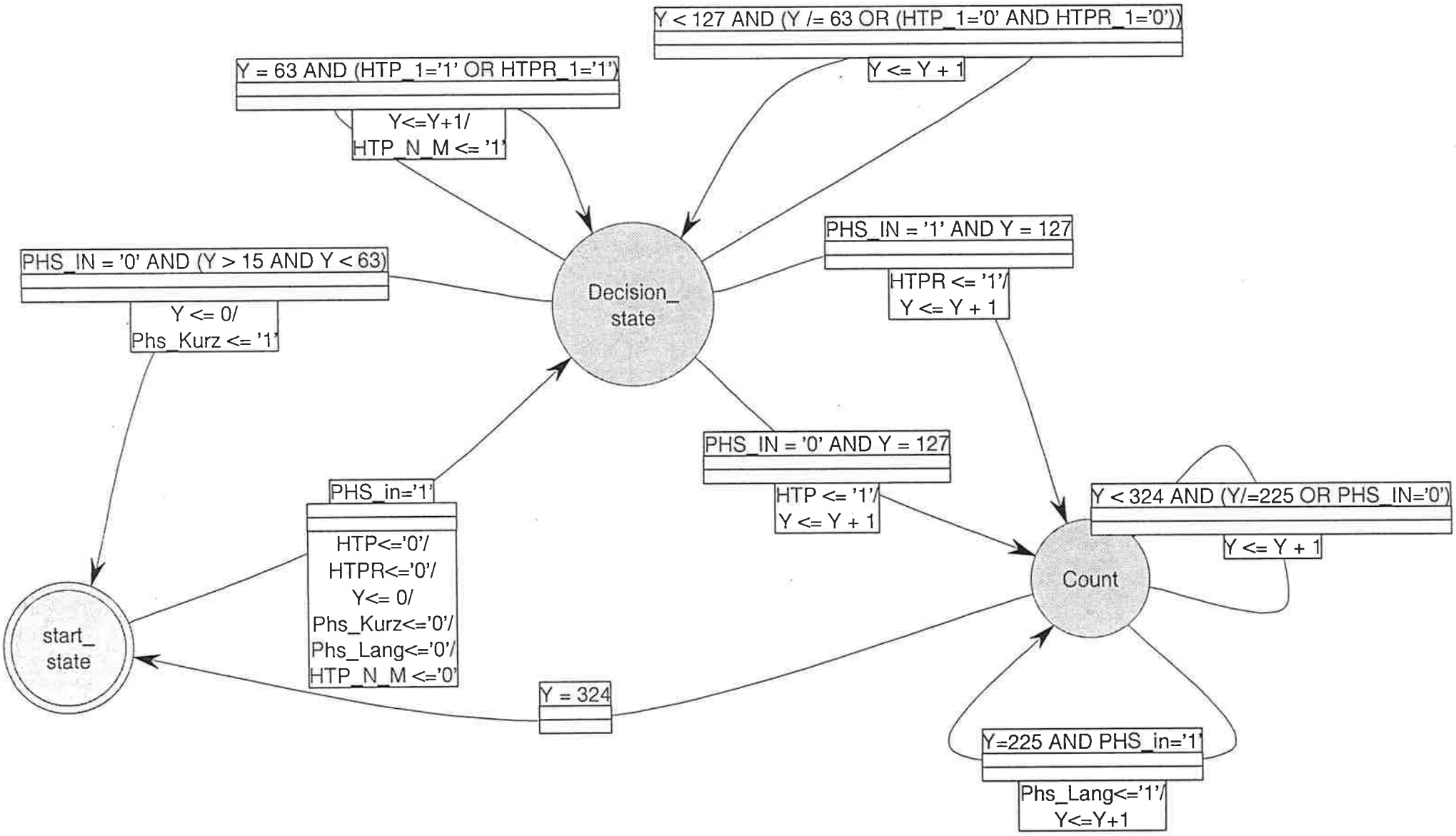
Context Diagram for w_fit



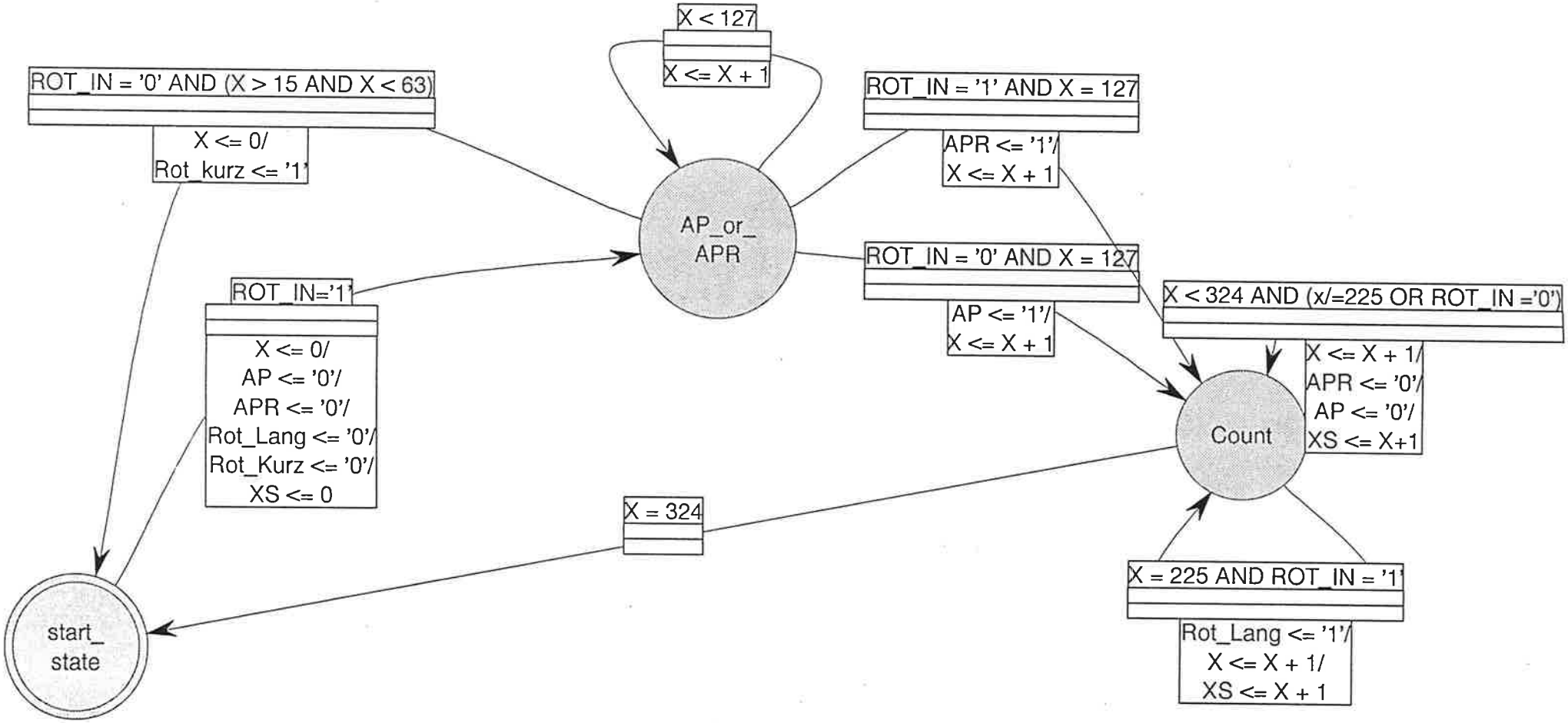
Data Flow Diagram for w_fit



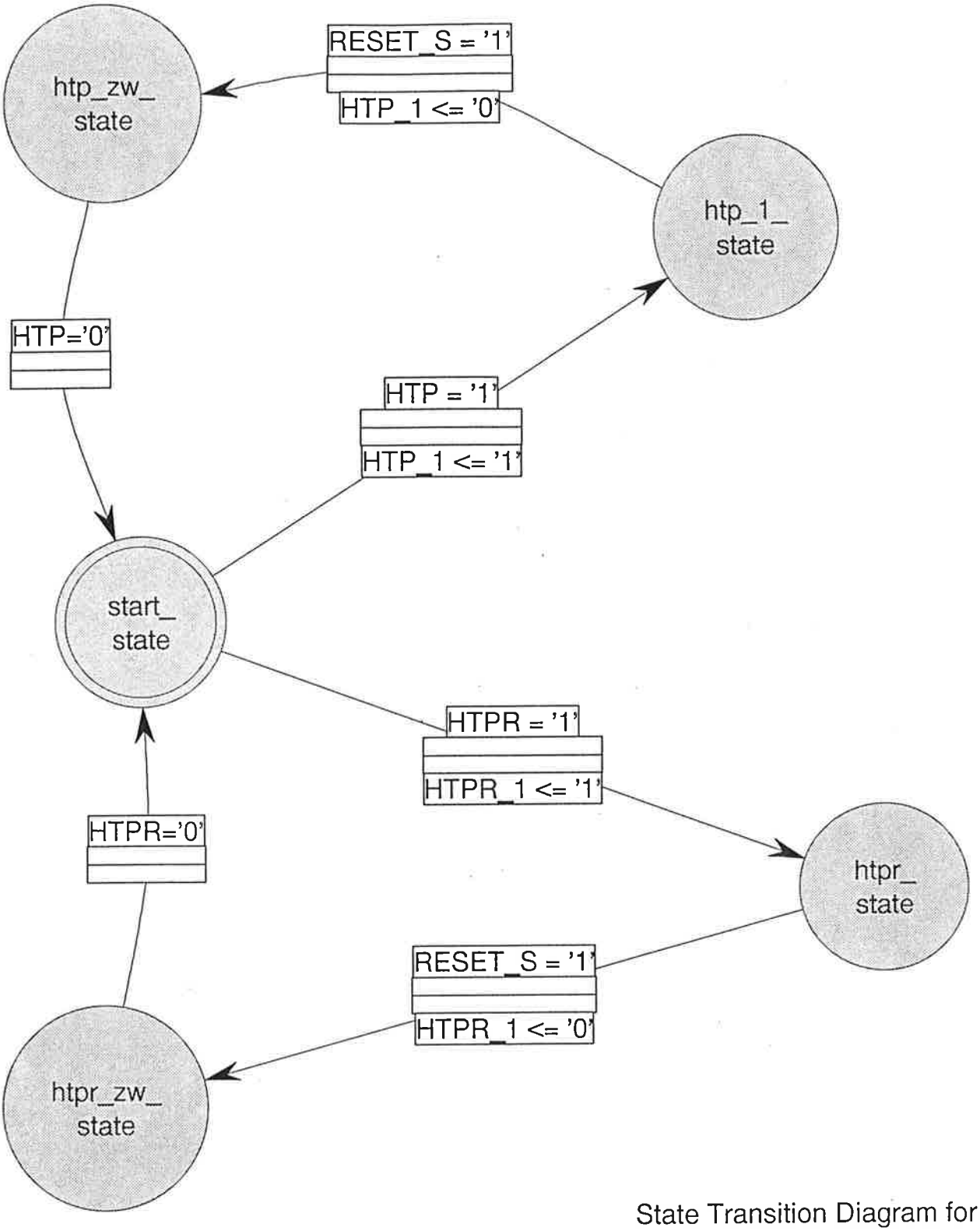
State Transition Diagram for controll



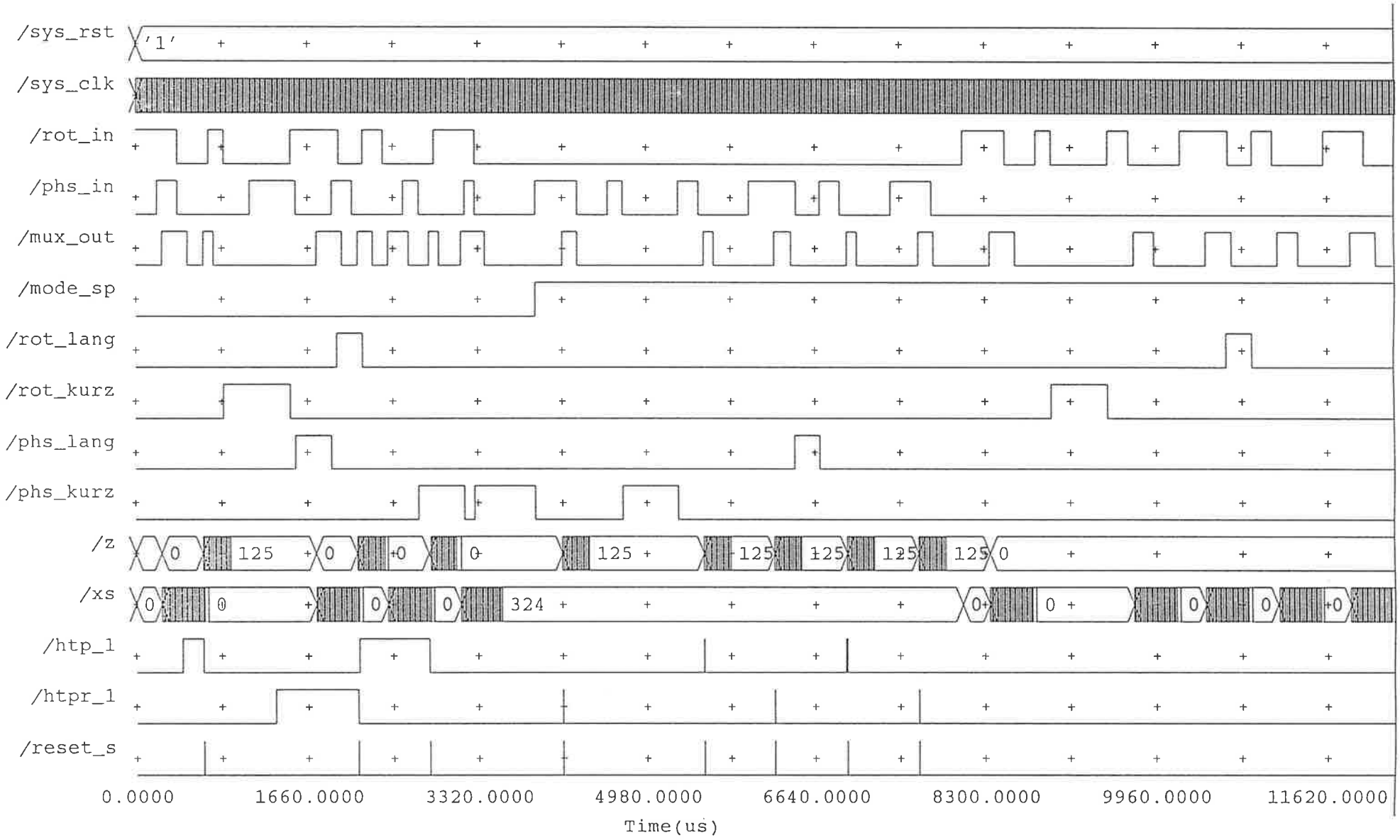
State Transition Diagram for phs

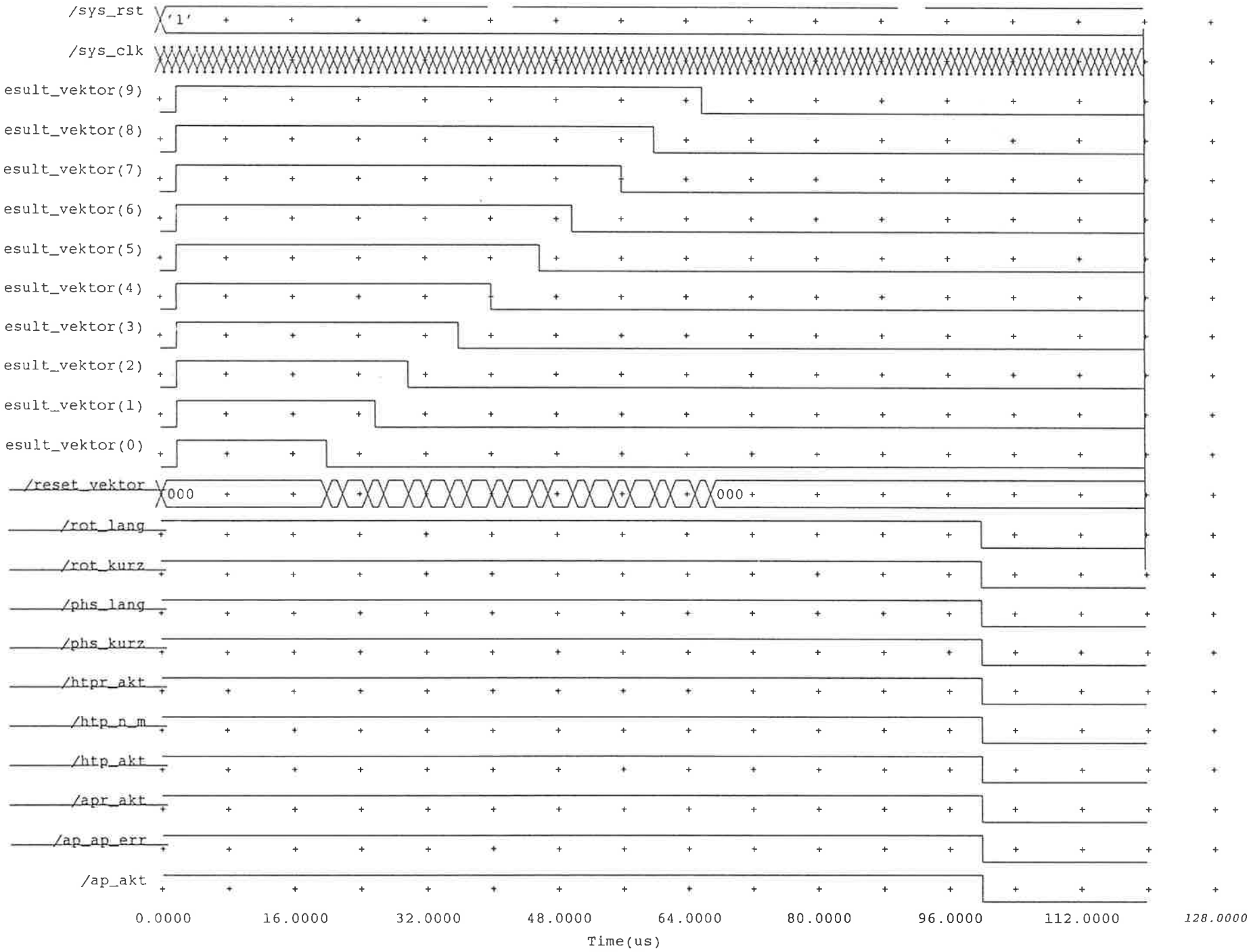


State Transition Diagram for rot

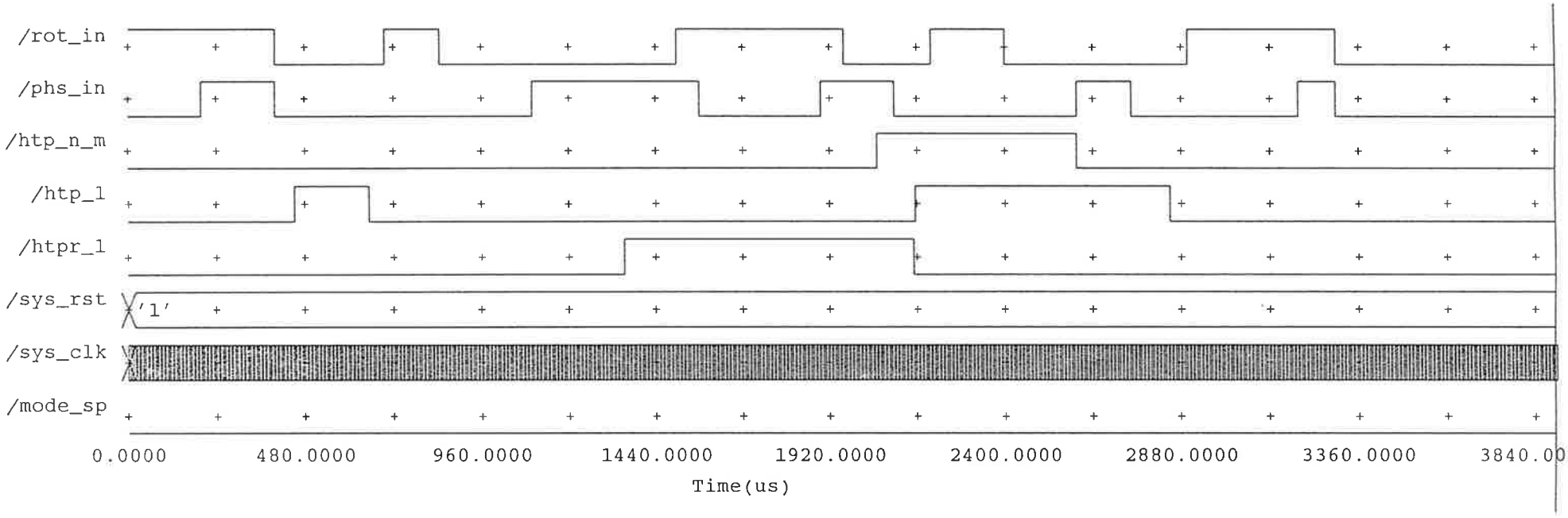


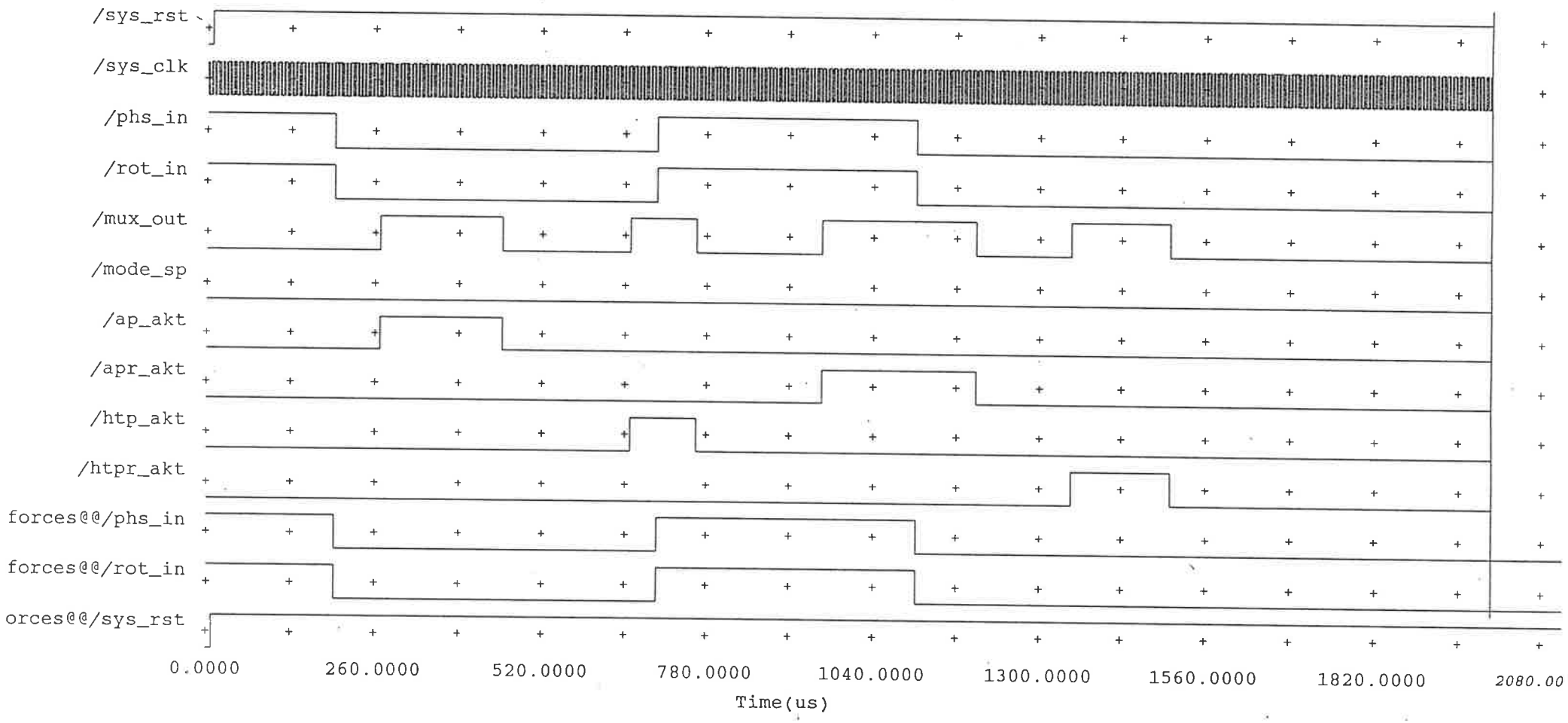
State Transition Diagram for speich


/sys_rst
'1'
/sys_clk
/rot_in
/phs_in
/mux_out
/mode_sp
/rot_lang
/rot_kurz
/phs_lang
/phs_kurz
/z
0
125
/xs
324
/htp_1
/htpr_1
/reset_s
0.0000
1660.0000
3320.0000
4980.0000
6640.0000
8300.0000
9960.0000
11620.0000
Time(us)


/sys_rst
'1'
/sys_clk
esult_vektor(9)
esult_vektor(8)
esult_vektor(7)
esult_vektor(6)
esult_vektor(5)
esult_vektor(4)
esult_vektor(3)
esult_vektor(2)
esult_vektor(1)
esult_vektor(0)
/reset_vektor
000
000
/rot_lang
/rot_kurz
/phs_lang
/phs_kurz
/htpr_akt
/htp_n_m
/htp_akt
/apr_akt
/ap_ap_err
/ap_akt
0.0000
16.0000
32.0000
48.0000
64.0000
80.0000
96.0000
112.0000
128.0000
Time(us)

htp_n_m reagerar med force Mode_Sp som det är tänkt




/sys_rst
/sys_clk
/phs_in
/rot_in
/mux_out
/mode_sp
/ap_akt
/apr_akt
/htp_akt
/htpr_akt
forces@@/phs_in
forces@@/rot_in
orces@@/sys_rst
0.0000
260.0000
520.0000
780.0000
1040.0000
1300.0000
1560.0000
1820.0000
2080.00
Time(us)