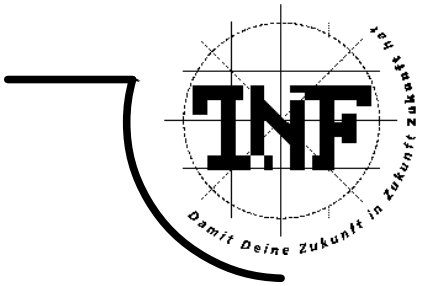
JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis



A hardware implementation of a FORTH-System with a Java compiler

DIPLOMA THESIS

to obtain the academic degree

GRADUATE ENGINEER

in the study

COMPUTER SCIENCE

Made at the Institute for integrated circuits

Contact:

O. Univ.-Prof. DI Dr. Richard Hagberg Lauer

Submitted by:

Gerhard Hohner

Linz, August 2008

Table of contents

Table of figures

Introduction

1 What is FORTH?

FORTH is versatile. It is a real-time operating system, an interpreter
or an interactive compiler, an extensible data structure, a software concept. In
FORTH is first and foremost but as a structured language to see which of their followers
the fourth generation is allocated, and with very few language elements
comes out, but the language scope what relation to commands and data structures,
is extensible. Each program, each variable is viewed as a language extension,
and managed by the runtime system in a dictionary as a token. The
means that even a loaded program system easily expanded and supplemented
can be without having to recompile the entire application and load. For
further, it means the program development that a gradual, based
Test, starting with the basic features of the application on the target system, possible
is.

The core of a FORTH system is a stack machine with a stack of data to the
Calculation of arithmetic expressions, parameter passing and return of result, and
an own stack of return addresses. The machine is not separate
Memory for data and programs. There must also be a run-time system, the
provides a user interface and manages the dictionary.

2 To implemented FORTH system

The core is to be implemented by a processor dedicated to developing, its
Instruction set consists of the basic commands of FORTH. As architecture is a
Stack machine is predetermined in an FPGA to implement with two stacks.
The runtime system (server) is also to accommodate to this module, and contains
a library of standard functions and services. The system should be minimal so
bloß from the Processor, one Memory and one serial Interface to the
There are communication with a client. The logic of UART is also on the FPGA
This integrated component is to implement, managed by the runtime system. The
Client runs on a standard PC, and provides the user interface, as well as service
programs (assembler, etc.) on. Finally, it should be possible with the client FORTH-
Translate programs, the object file it created to send to the server the
They invite, binds, initializes, enters in the dictionary and, if prescribed, will start. To
a date can run only one application on the minimal system, but
can be at any time terminated. It can, but multiple applications loaded
be, are also in interaction.

3 Java

It is desirable because a more sophisticated language available
FORTH is very simple and just taken just a SP Rathe the third
Generation is. Java offer. Two possibilities are imaginable, to the first
Just one Java virtual machine, JVM, which runs on the minimal system and the bytecode

the Java classes a compiler interprets, secondly, as the utility of
Available clients and Java converts to FORTH.

3.1 Java virtual machine

Arguments for a JVM:

- The classes of an application are always up to date. It meets the current JDK to download and to transfer the application to the minimal system. Bloß the JVM must be adapted to the current bytecode Java.

Arguments against:

- An emulator for the processor of the JDK platform must be implemented, to in Machine code written methods, characterized by the attribute native,. to run.
- A completely transparent, entirely independent of the JVM and the applications,. dynamic memory management and garbage collection must be implemented; about mark and sweep. The unpleasant fact is that no influence on the Garbage collection can be taken. During this process, a unpredictable time with unknown duration runs an application can their Unable to keep the tasks.
- All required classes in an application must be loaded, immediately, what a huge store of the minimal system requires. Assessments of the Typical memory size can not be made.

Because the disadvantages are weighty, is not a JVM.

3.2 Compiler

Advantages:

- The applications can be run in the machine code.
- The classes in the JDK, whose source code free of charge provides Sun, can the The minimum system requirements; the memory requirement can small be kept.
- *Reference counting can be used as dynamic memory management. The* Applications have full control over the garbage collection. Memory is only released when no longer needed and is immediately available.
- The minimal system must not be adapted for Java.

Disadvantages:

- The source classes have to be adapted.
- *Reflection is probably not possible.*
- You must commit themselves to a version of Java, the adaptation to the latest It is no minor effort.

The compiler is the more attractive option, especially since Java like a shell of the FORTH-

System is placed and even unrecognizable for the system.

4 The history of FORTH

The following chronology represents a summary of [17].

1968	Charles Moore begins with the development of FORTH on an IBM System
1970	Charles Moore published the first paper on FORTH
1971 to the	Charles Moore implemented FORTH on a minicomputer for the NRAO Control of radio telescope at the Kitt Peak
1973 FORTH	Edward K. Conklin, Charles Moore and Elizabeth D. Rather set up Inc.
1976 6800	Development of microFORTH for RCA the microprocessor 8080, Z80,.
1978	Develop a FORTH system for the 6502 Selzer, Ragsdale
1978	FIG (FORTH interest group) is founded
1980	FORTH also running other operating systems (Apple II, CP\/M)
1982	IBM PC FORTH running PC-DOS on the IBM PC
1985	the first FORTH processor offered by Harris
1987	Formation of a Committee for the standardization of FORTH
1994	Publication of standards ANSFORTH in DPANS'94 by ANSI
1998	The Committee adjusts his work

Hardware

1 Design of the processor

Here, only provisions with regard to the processor be taken, so detailed as necessary and as General as possible. There will be no information concerning the software made.

1.1 The basic FORTH commands

All basic commands [2, Chapter 6, page 21-46], grouped by fields, with Mnemonics and detailed description, are given. In addition, there are still complex commands, which are stated in the standard, they can be but by a Combination of the basic commands (macros) to imitate.

Some remarks on the representation of the stack in the action column in the following Tables:

- Stack are shown in the State before and after, and are by \"-\" separated. Multiple stacks are affected by a command the second person concerned receives a prefix. In the tables, this is merely the return address stack, with prefix p:.
- The right-most element in the stack is the topmost element. All previously mentioned are deeper.
- stands for any address in addr.
- n stands for a word smattering of sign.
- u stands for an unsigned word.
- x stands for a Word, whose actual Typ is insignificant.
- helped (offset) stands for a half word smattering of sign.
- char represents a single Characters, a 8-bit smattering of signed characters.
- *Italic mnemonics features commands that are not defined in the standard.*

1.1.1 Store commands

Since the standard does a word size of 16 bits, this but in terms of Java is is oblong,. have I realized a 32-bit architecture. This requires additional Commands for half words and pointer for double words are not necessary, they can be reproduced. The store commands are shown in table 1.

1.1.1 Store commands

Mnemonics	***action***
!	n addr- stores the second word n at memory address addr and eliminates the two words from the stack
@	addr - x. replace addr the addr stored contents x.
C!	char addr- stores the second word char as a single character at memory address addr and eliminated the two words of the stack
C @.	addr - char addr substitute the character stored in addr
H!	half addr- stores the second word half as double single-byte in memory address addr and eliminated the two words of the stack
H @.	addr - half replace addr with the double-byte at address addr
SP!	u addr- Initializes the stack pointer with address addr and sets the current Length of the overflow area on u words.
SP @.	-addr copies the stack pointer onto the stack
B!	u- the transfer is used, if u < > 0, otherwise deleted
B @.	-u u has the value - 1 if the transfer is otherwise set 0
32 @.	-u the next word - reads, starting with the most significant word - of Product of 64-bit multiplier.
64!	u u- loads the double word operands in the 64-bit multiplier

Table 1: Memory commands

1.1.2 Stack manipulation commands

Purpose This Commandsis only the Manipulation the Data or Return address stack. There are some standard commands but with double words operate,. the not led are,. they can patterned after be. The Manipulation commands are shown in table 2.

1.1.2 Stack manipulation commands

Mnemonics	***Action***
DEPTH	-u sets the length of the stack (in words) on the stack
DROP	x- Deletes the top word of the stack
2DROP	x 1 x 2-. Deletes the two Supreme words of the stack
NIP	x 1 x 2 - x 2 eliminate the second word in the stack
PICK	Xu... x 0 xu... u - x 0 xu copy the u-th element in the stack to the top of the stack
PUT	Xu xu-1... x 0 n u - n xu - 1... x 0 the second n replaced the word at the relative position of u in the stack in the stack and eliminated the two operands from the stack
[+ |-] n [VAL]	-n n pushes onto the stack
[+|-] half *[VALH]*	-half Converts helped in a word smattering of sign and pushes it onto the stack
[CONST] [+|-]x x Є[[0: 15]	-x Converts x in a word smattering of sign and pushes it onto the stack
DUP	N1 - n1 n1 duplicates the top word of the data stack
OVER	n1 n2 - n1 n2 n1 copied the second word of the data stack onto the stack
R @.	-n1 R: n1 - n1 copies the top word of the alternate stack onto the data stack
R1 @.	-n1 R: n1 n2 - n1 n2 copied the second word of the alternate stack onto the data stack

Mnemonics	***Action***
SAVE	... x -- Fully secures the cache of the stack in the overflow area

Table 2: Stack-manipulation commands

The subdivision of the numbers in 32-bit, 16-bit, 5-bit values increases the number of commands though, but greatly reduced the size of the compiled code.

1.1.3 Arithmetic and logical operations

The commands of the group are shown in table 3.

Mnemonics	***Action***
+	N1|U1 n2|u2 - n3|u3 adds two top words and replaces them with the sum of
-	N1|U1 n2|u2 - n3|u3 the two Supreme words replaced by the difference n1 - n2
CMP	N1|U1 n2|u2 - n1|u1 n2|u2 n3|u3 sets the difference n1 - n2 a n3 au f den StaPEL
H ***	uhalf1 uhalf2 - u the unsigned product of two half word replaced them on the stack
1 + --	N1|U1 - n2|u2 the top Word increments by 1
1 --	N1|U1 - n2|u2 decrements by one the top Word
2.	x 1 - x 2 the top Word is multiplied by 2
2 V. --	x 1 - x 2 the top floor is divided by 2
AND	x 1 x 2 - x 3 replace the two top words with their logical and

1.1.3 Arithmetic and logical operations

Mnemonics	***Action***
CELL +	Addr1 - addr2 added the length of a Word to addr1. The result replaces addr1
HALF +.	Addr1 - addr2 added the length of a half word to addr1. The result replaces addr1
INVERT	x 1 - x 2 inverts the top Word
INVERT	x 1 - x 2 inverts the top Word
LALT	x 1 u - x 2 sliding x 1 to u places to the left. The result replaces the two top words
NEGATE	x 1 - x 2 negates the Supreme Word
OR	x 1 x 2 - x 3 replace the two top words with their logical or
RSHIFT	x 1 u - x 2 sliding x 1 to u points to right, without taking into account the Sign. The result replaces the two top words
XOR	x 1 x 2 - x 3 replace the two top words or with their exclusive
+B	N1|U1 n2|u2 - n3|u3 Adds the top two words with the regrouping and replaced both Words by this sum
B	N1|U1 n2|u2 - n3|u3 replace the difference n1 - n2 - transfer two top words
LSHIFTC	x 1 u - x 2 sliding x 1 to u for bodies, with inclusion of carryover, left. The Result replaced two top

Mnemonics	***Action***
RSHIFTC	x 1 u - x 2 sliding x 1 to u points, with inclusion of carryover to the right,. without taking into account the sign. The result replaces the two top words

Table 3: The arithmetic logical commands

Extensions were to support double word and half word processing made, the standard does not exist.

1.1.4 Comparison operations

The commands of the group are shown in table 4.

Mnemonics	***Action***
0 <!	n n flag flag is set if n < 0, otherwise deleted. The flag is on the stack set
0 <	n - flag flag is set if n < 0, otherwise deleted. The flag replaced n on the Stack
U0 <	u - flag flag is set, if the transfer is set, otherwise deleted. The flag replaces u on the stack
U0 >	u - flag flag is set if the unsigned u < > 0 and the transfer is deleted, otherwise tagged. Das flag replaces u on the stack
0 >!	n n flag flag set, if n > 0, otherwise deleted. The flag is on the stack set
0 >	n - flag flag set, if n > 0, otherwise deleted. The flag replaced n on the Stack

1.1.4 Comparison operations

Mnemonics	***Action***
0 =!	n n flag flag is set if n == 0, otherwise deleted. The flag is on the Stack set
0 =.	n - flag flag is set if n == 0, otherwise deleted. The flag replaced n on the Stack
0 < >!	n n flag flag set, if n! = 0, otherwise deleted. The flag is on the stack set
0 < >	n - flag flag set, if n! = 0, otherwise deleted. The flag replaced n on the Stack

Table 4: Comparison operations

Extensions have been made to increase efficiency, the it standard not are. Relations that use both upper words with two are generally Commands implemented. First \"-\", if the operand is no longer needed, or *CMP, if the operands should be preserved to the formation of the difference. It follows 0 <, 0 >, 0 = 0 < >, U0 < or U0 > evaluation of the difference. The FORTH commands <, >, =, < >,. U >, U < are implemented as macros. In addition, to get the operand, nor the* Macros <!, >!, =!, < >!, U >!, U <!.

1.1.5 Program flow control commands

Mnemonics	***Action***
EXIT	R: addr-. Replace with the return address addr the program counter, and takes They launched
offset CALL	R: - PC A program is invoked. The current value of command count PC on the stack is placed, and added offset to PC

Mnemonics	***Action***
addr TRAP	R: - PC A program is called indirectly. The current value of Command counter filed PC on the stack, and PC by the contents of Replaces the word in addr
offset *BRANCH*	Unconditional jump to the PC is added in offset
offset *0BRANCH!*	flag - flag flag has a value of 0, is added to the PC offset
offset *0BRANCH*	flag- flag has a value of 0, is added to the PC offset
core *SETPC*	PC- Initializes the PC of the nucleus core. This command must precede SWITCH run
core *SWITCH*	-- Core, which means changes to the core that the current stack pair as well as PC be changed.
GETCOREID	- *coreno* The ID of the active nucleus pushes onto the stack

Table 5: The program flow control commands

None of the commands in table 5 is defined in the standard, but their existence is provided. Multiple cores in a CPU are not existential, but modern. Also multiple cores are to be supported.

1.1.6 Other commands

Mnemonics	***Action***
NOP	Producing an extension step (stable)
STOP	Stops command execution; can only by lifted an interrupt be
BREAK	Producing an extension step (stall); only for the test of the BIOS Software provided

Table 6: Other commands

1.1.6 Other commands

The commands listed in table 6 does not exist in the standard, but useful.

Overall, there are 99 commands! 33 of which are but redundant and serve only the Efficiency, which reduces the length of the compiled code by about 25%.

1.1.7 Findings

- Command needs no more than two operands
- A command produces exactly one or even no results
- No indication of the word width of the operands
- Only direct (immediate) addressing is used
- A load and store architecture is implied

1.1.8 Specifications

- Immediate operands should in Intel format (Little Endian - high byte on) exist (the high address low significant byte on the lower address)
- A byte-code is sufficient for the encoding of the commands
- The commands should be general so applicable, purpose, on each batch
- The word width shall be 32 bit
- The instruction set should be easily expandable

1.2. A characterization of stack machines

Stack machine is not equal to stack machine. The more accurate description is on distinctive properties referenced that allow a classification. Commonly used are following characteristics [1, Chapter 2.1, page 25-26]:

- Number of stack
- The stack buffer
- Number of operands of an instruction

A machine can be one or multiple stack v eRWenden. Please note that this Characteristic makes no indication of the realization of the stack. In the simplest case These are merely special register, the single stack in the address space locate. Your own machine commands support the use as a batch. It is only briefly noted that there were also computer or are, without stack coming out, about VAX or IBM 360 \/ 370.

To increase throughput, whereas the top stack entries in the Machine for buffers. This should be so, must not be but a representative buffered Stack would be the MC68000. The advantage of buffering is dar in to sehen, that the buffered entries without additional access to the stack is processed can. Only the writeback of modified entries must not only in the buffer, but also on the stack. A further improvement can use be obtained a cache between stack and buffer, the replacement strategy must but write through so immediate replacement not only in the cache, but also

in the stack. Alternatively to the cache, the use of local storage is possible
It is home to the top of the stack. Only if the size of the stack using the size
This Store addition grows, is Memory of the Stack in Claim
taken. This local memory must be directly accessible the CPU, without which
System buses in claiming to represent an improvement. Cache and
local stores are only additional features in a classification a
ArchiteKTur nicHT except eight should be allowed.

It makes a difference whether the machine is a 0, 1, or 2-Adressmaschine
represents. While a 0 Adressmaschine all operands implicitly on the stack
lie,. and not explicitly specified be must,. can at 1 and 2
Adressmaschinen additional operand, such register or memory contents in a
Operation are introduced. Additional operands must be provided, which
be not timeless, can immediately start the during 0-Adressmaschinen
Command execute. All machines have in common the ERGbnis on the stack to
set. The top element is implicitly as the second operand at 1-Adressmaschinen of
Stack used if the command requires an additional operand.

It is worth noting, that literals and constants, the operation code of load commands
or subroutine call are attached, are possible even at 0 Adressmaschinen.

1 and 2 Adressmaschinen need in the General less Commands for one
Calculation as a 0 Adressmaschine, the encoding of the instructions is but longer
a 0 Adressmaschine comes from a byte code.

1.3 Setting the type (2, 2, 0)

A 0 Adressmaschine is sufficient as the engine of choice. By definition, FORTH expected
two stack - the data stack for calculations and parameter passing, and one
Stack for the return addresses of called subroutines.

Commands use not more than the two Supreme Elements, Thus is sufficient
The length of two for each of the two stack stack buffer.

A stack itself should consist of a tip that should be in a local store.
In addition, a registry that refers to a memory location that the the overflow
absorbs the tip, or at lower reaches this invite to the top. These operations
modify the registry automatically. Is alternative to use a cache,
excluded because the replacement strategy should be write through, and so on
Definitely a memory access via the system bus needed would make. Local
Memory not needed one of those.

Because at a time only with a stack - work FORTH
knows No Command the at the same time on two Stack works, is sufficient one
Stack management to manage both stack. The information on which stack
worked is, must come from the command decoder.

1.4 The stack organization

A local store is used which is organized as a ring buffer, a stack
houses [1, Chapter 3.2.1.2, page 33-34]. Parameters of the stack are the position
the first and the last item in the store. Should this stack overflow, which is

final element in a range of rescue in the main memory to swap out, vice versa in one Underflow from the Recovery to store. Performance indicator the Recovery is a pointer that contains a defined main memory address. This Pointer must be modified on each read or write operation, so that he always Displays the last saved date. The rescue area is thus itself again a Stack.

1.4.1 The local memory

Preferably, its size is a power of 2 - you will save a Modulooperation When modifying the stack parameters. With simulations [1, Chapter 6.4.1, page 139-] [143] it was found that a size of 64 for the execution of arithmetic Expressions is perfectly adequate. As the stack of data not only to the evaluation of Express, but also to r PARAmeterübergabe is used to suggest I as minimum size 64 the number of excess and underflows in any case before, so low. keep can be.

Indexf points to the topmost element in the stack. Many of the commands require the second element, which is achieved by Indexs. In order to rapidly rescue a Overflow, must be the lowest item always available indexed by Indexl-. With *Indexr should be also possible optional each element in the stack reading access. Indexi identifies a freely selectable position in the stack for write operations. Unless* It was only mentioned that Indexi requires a clearance signal for the purpose of clarity omitted. Optional reading via the content as address the PICK command, of the top stack element used. The contents of the addressed item replaced top element of the stack. The size of the stack does not change so, there may be no Overflow occur. Therefore, the indices of Indexl and Indexr become a statistic summarized, it is but a control signal necessary to the currently desired index to select. Indexf, Indexs need no control signals, they are always active.

The read out Data *First, Second, Random* form Input values for the Stack buffer, load is in overflow immediately in the main memory written, otherwise ignored.

The local memory must have so four independent ports, three to the one concurrent read, concurrent write. There is a store two cycles are necessary to read the addresses are created in the first, initial the data is provided in the second measure. Write only one measure, needed to keep in mind, that successfully read the new value until the next clock can be. These boundary conditions must be observed in the design of the stack buffer.

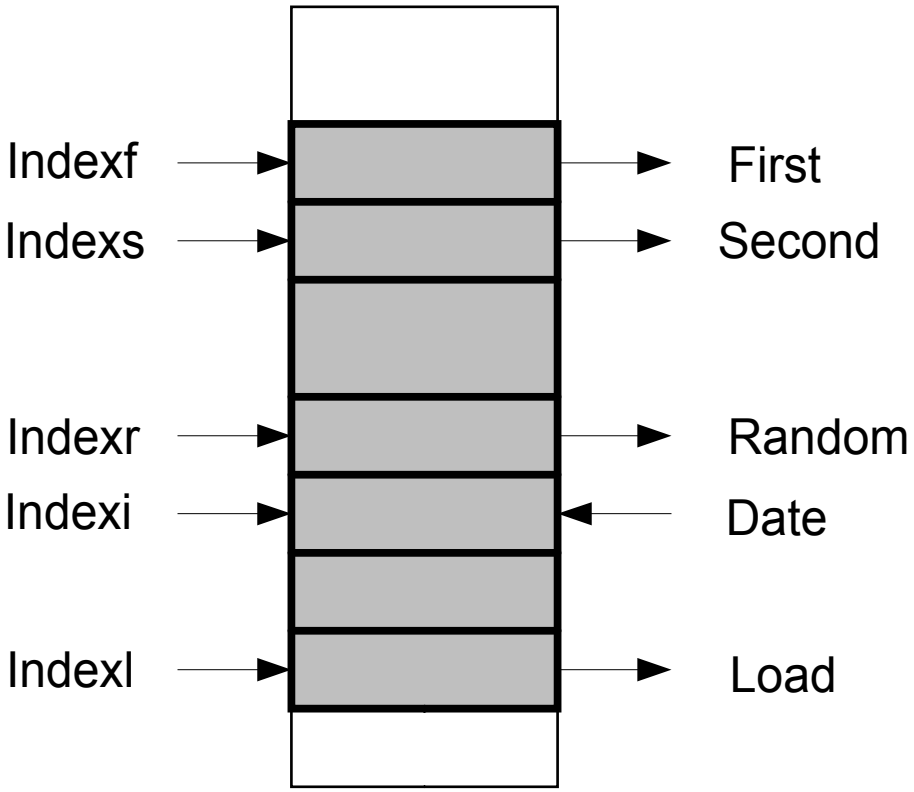


Figure 1: The stack

1.4.2 The stack buffer

The following arrangements apply:

- t the date to which the parameters are updated
- t + 1 the time at which the stack is indexed
- t + 2 the time at which the values buffered and processed
- t + 3 the time that new values are written on, and the stack buffer is updated
- top the active stack buffer
- buffer(2,2) the matrix of buffered entries
- p(2,2) the shade store of the stack buffer
- Select the number of the selected stack
- date a new value
- random Optional read stack entry
- first top stack entry
- second second batch entry
- last the value of the lowest stack entry
- fast(2) is true if a new date for the transfer is ready
- early(3) is true if the last stored date the most recent value is
- CUR(3) is true if the last stored date is the latest value

Next is \"|\" separator of alternative values, the left alternative has ever-higher Priority over the rights.

Then applies to the stack buffer that currently being edited:

$$top_{t+1} \leftarrow buffer_{t+1}(select)_{t+2}$$

In other words, there is work on the stack, whose last parameters were modified to its current data values are available in the shadow memory p.

The last element written last the represented either just on this spot Or the already existing value.

$$last_{t+3} \leftarrow date_{t+2,\ cur(2)=true} \mid date_{t+1,\ early(2)=true} \mid random_{t+2}$$

Similarly with the stack buffers, as well as the shadow storage.

$$buffer_{t+3}(select)_{t+3}\ 0) \leftarrow date_{t+3,\ fast(0)=true} \mid date_{t+2,\ cur(0)=true} \mid date_{t+1\ early(0)=true} \mid first_{t+1}$$

$$buffer_{t+3}(select)_{t+3}\ 1) \leftarrow date_{t+3,\ fast(1)=true} \mid date_{t+2\ cur(1)=true} \mid date_{t+1\ early(1)=true} \mid second_{t+1}$$

$$buffer_{t+3}(select)_{t+3} + 1) \leftarrow p_{t+2}(select)_{t+3}(+1).$$

$p_{t+3}(select)_{t+3} \leftarrow buffer_{t+3}(select)_{t+3}$

Take the above calculations of *fast and early their values at time t + 2 the* Stack buffer are thus specified that almost functions and early are:

$early_{t+1}(0) \leftarrow$ true if indexf $_{t+1} = indexi_{t+1}$

$early_{t+1} \leftarrow$ true (1) if indexs $_{t+1} = indexi_{t+1}$

$early_{t+1} \leftarrow$ true (2) if indexl $_{t+1} = indexi_{t+1}$

$cur_{t+2}(0) \leftarrow$ true if indexf $_{t+2} = indexi_{t+2}$

$cur_{t+2} \leftarrow$ true (1) if indexs $_{t+2} = indexi_{t+2}$

$cur_{t+2} \leftarrow$ true (2) if indexl $_{t+2} = indexi_{t+2}$

$fast_{t+3}(0) \leftarrow$ true if indexf $_{t+2} = indexi_{t+3}$

$fast_{t+3} \leftarrow$ true (1) if indexs $_{t+2} = indexi_{t+3}$

In practice, the values for almost all arise along the way. Each operation, a Result sets onto the stack, due to implicitly \"indexf\" $_{t+2} = indexi_{t+3}$". To explain: to the Time t is the position indexf $_{t+1}$ of the top stack entry be carried out. The Result of the operation at the time of t + 2 is the new value date $_{t+3}$ on the stack to the place indexi $_{t+3}$ at the time of t + 3 should be written. It must therefore be the Equality \"indexf\" $_{t+2} = indexi_{t+3}$\"apply.\" A restoration operation, by storage the first or second element from below, updated from the overflow, for the first element \"fast(0) = true\" or the second sets \"fast(1) = true\".

The physical implementation of the stack buffer is the shadow memory p stack buffer, even a priority function, is the the most recent values of shadow memory, current Selects the contents of the stack and last results.

1.5 Definition of processor core

A command by lAEft folgende phases indicated in table 7.

Bezeichnu NG	***Unit***	***Description***
Download	Prefetch queue	Removes the next command and the may. Hung immediate value
decode	Befehlsdekoder	Is that necessary for the command Tax information ready
Update stack	administrations g	Updated the parameters of selected stack
prepare	Stack buffer	Updated the selected stack buffer
process	Stack buffer, ALU, memory	Evaluates data and tax information
To write	Stack administrations g	Inserts eventueLL generated date onto the stack and the stack buffer

Table 7: The phases of a command

Each of these stages is overlap-free with the other phases, each phase should come with one-step operation. Phasenpipelining is possible, the means that a command is fully worked with every cycle, although each command at least six Operation steps required. Each Phase can additional Operation steps force, should inform all other phases in that This may be an extension step (stall) must run.

1.5.1 Prefetch queue

This unit reads from getting whole words from memory in a queue which they the commands with immediate value, if required, sequentially removes and forwards it to the decoder. Advantage of this strategy is to minimize Memory accesses, as do most of the commands with a single byte, and with an access up to four commands are read. The queue can more than one Word record, are also long commands, TRAP about five bytes, usually without required Extension step for uploading the additional byte executable, since this unit the Queue is always filled. The command counter by a jump command is or Program new set is required also the queue new what as Disadvantage is because extension steps are necessary.

This unit is an own data and address bus.

1.5.2 Command decoder

He is just an implementer, its description after the implementation of stack management is possible.

1.5.3 Stack management

Updating and writing were already set, remain still provisions to the
Processing to meet. The ALU is alone the stack management available, she will
fully utilised by this. Furthermore it has an own data and address bus for
Memory accesses of the processing type, these are to be eine höHere priority than that of the
Prefetch queue have.

1.6 The processor

It should be the core and the following function groups with a
Bus Manager synchronize and integrate, and the connection of external units, as well as
like the store, allow.

1.6.1 UART

Should communicate with one other computers via a serial interface
(RS232) to be possible, the UART is the bridge between the interface and the processor.
Properties of UART:

- separate transmitter and receiver, able to produce both a break signal
- Data Word 7 or 8 bits wide
- Stop bits 1, 1.5 or 2
- Parity none, even or odd
- Baud rate selectable

The ports are memory mapped

1.6.2 Counter and timer

Four independent backward counter, each 24-bit wide, each with an external count input,.
also able to produce a break, should be available. The ports are
memory mapped.

1.6.3 Interrupt controller

He will manage 16 Maskable Interrupt signals and, where appropriate, one
Send the interrupt vector to the core. The vectors given in table 8 show
the first 16 words of the address space (0-63), a vector containing the address of its
Interrupt handler.

1.6.3 Interrupt controller

Break assignment	
0	Undefined command
1	Software Interruption
2	Ext. 0
3	Ext. 1
4	Ext. 2
5	Ext. 3
6	Ext. 4
7	Ext. 5
8	Ext. 6
9	Ext. 7
10	Counter 0
11	Counter 1
12	Counter 2
13	Recipient UART
14	Transmitter UART
15	Counter 3

Table 8: Interrupt signals

The ports are also memory mapped.

1.6.4 ROME

It should absorb the processor system's BIOS and a size of 4096 words have. The Rome always whole words, can be accessed read only. His Adresslage is to directly connect to the address range of RAM.

1.6.5 Interface to the outside world

The processor is to control an asynchronous SRAM or a synchronous SDRAM can.

1.6.6 The block diagram of the processor

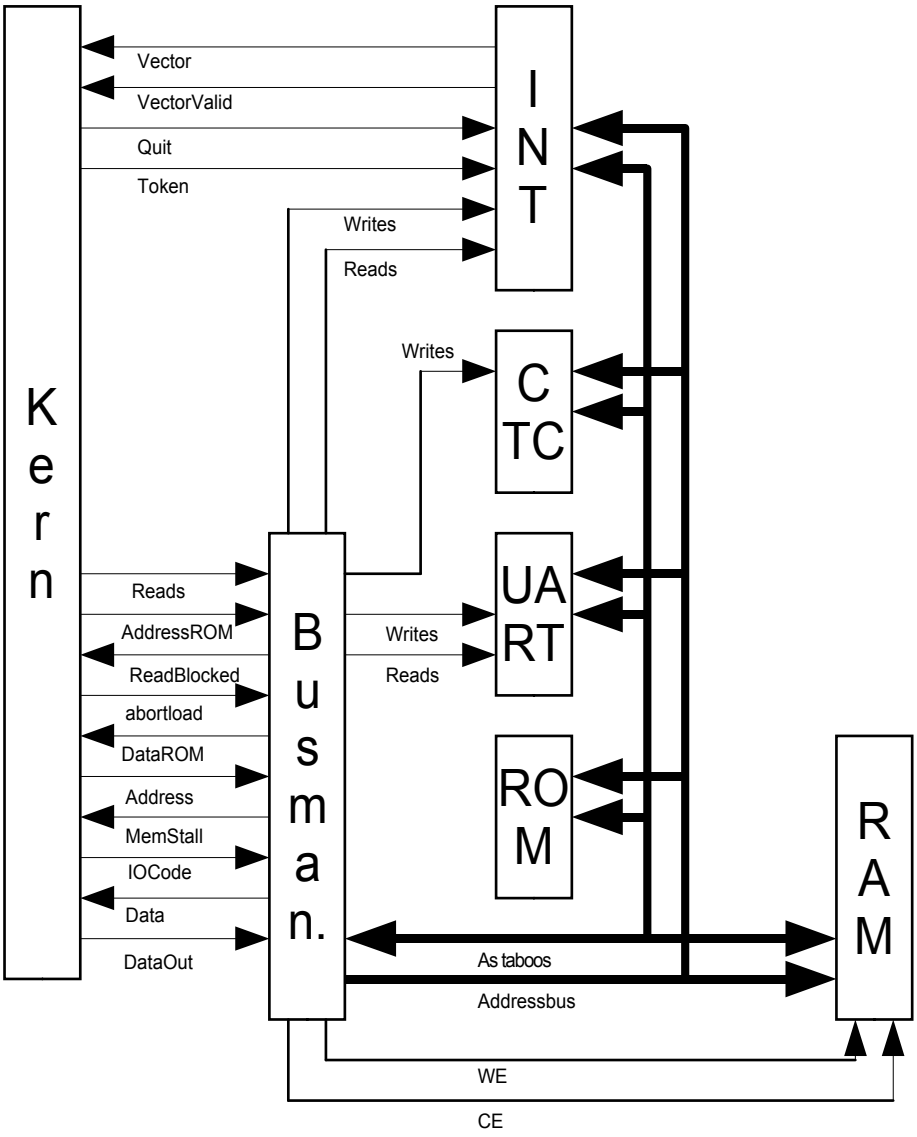


Figure 2: The simplified block diagram of the processor

The interruption of signals were not taken into account. The block RAM represented a static SRAM. The Rome decodes its allocated address space independently and needs therefore no selection signal CE. The data bus is a gross simplification. In the data lines of the components are not merged, the implementation but each component is connected directly to the bus Manager. Same applies for the Addressbus.

1.6.7 Final remarks

The Processor should a Proof for this be,. that RISC architectures not on

1.6.7 Final remarks

Register machines are limited, also stack machines are definitely eligible.
In contrast to the classical Harvard architecture, where program and data memory are strictly separated, should there be only a memory address, as in Neumann-Common architectures. To note also the command processing, is the an of Neumann cycle corresponds to.

2 Implementation of the processor core

The interface:

```
entity theCore is
  generic (constant CacheIndexBitWidth: integer: = 10;-ld(Cachesize))
           constant TableBitWidth: integer: = 4);     -ld(Interrupttable)
  port (nReset: in std_ulogic;)                       -system reset
        Clock: in std_ulogic;                         -system clock
        abortload: out std_ulogic;                    -abort memory cycle
        MemStall: in std_ulogic;                      -memory stall
        Data: In DataVec;                             -incoming data
        ReadBlocked: in std_ulogic;                   -blocked memory fetch
        ROM code incomming DataROM: in std_ulogic_vector(DataVec'range);--
        AddressROM: out std_ulogic_vector(RAMrange'high_-_1_downto_0); -ROME
                                                                        address
        Reads: out std_ulogic;                        -read ROM
        Address: out std_ulogic_vector(RAMrange'high_+_1_downto_0);    -memory
                                                                        address
        DataOut: out DataVec;                         -data to memory
        IOCode: out std_ulogic_vector(2_downto_0);    -memory operation
        illegal: out std_ulogic;                      -illegal opcode
        Token: out std_ulogic;                        -service routine running
        Quit: out std_ulogic;                         -vector processed
        Vector: in std_ulogic_vector(TableBitWidth_-_1_downto_0);-interrupt
                                                                  vector
        VectorValid: in std_ulogic);                  -vector valid
end theCore;
```

Source	***Comment***
theCore.vhd	Integrated all the following sources and implemented the command decoder
ALU.vhd	Implemented the ALU
theStacks.vhd	Implements stack management and integrated ALU
ProgramCounter.vhd	Implements the prefetch queue
Global.vhd	Global declarations

Table 9: The source files of the core

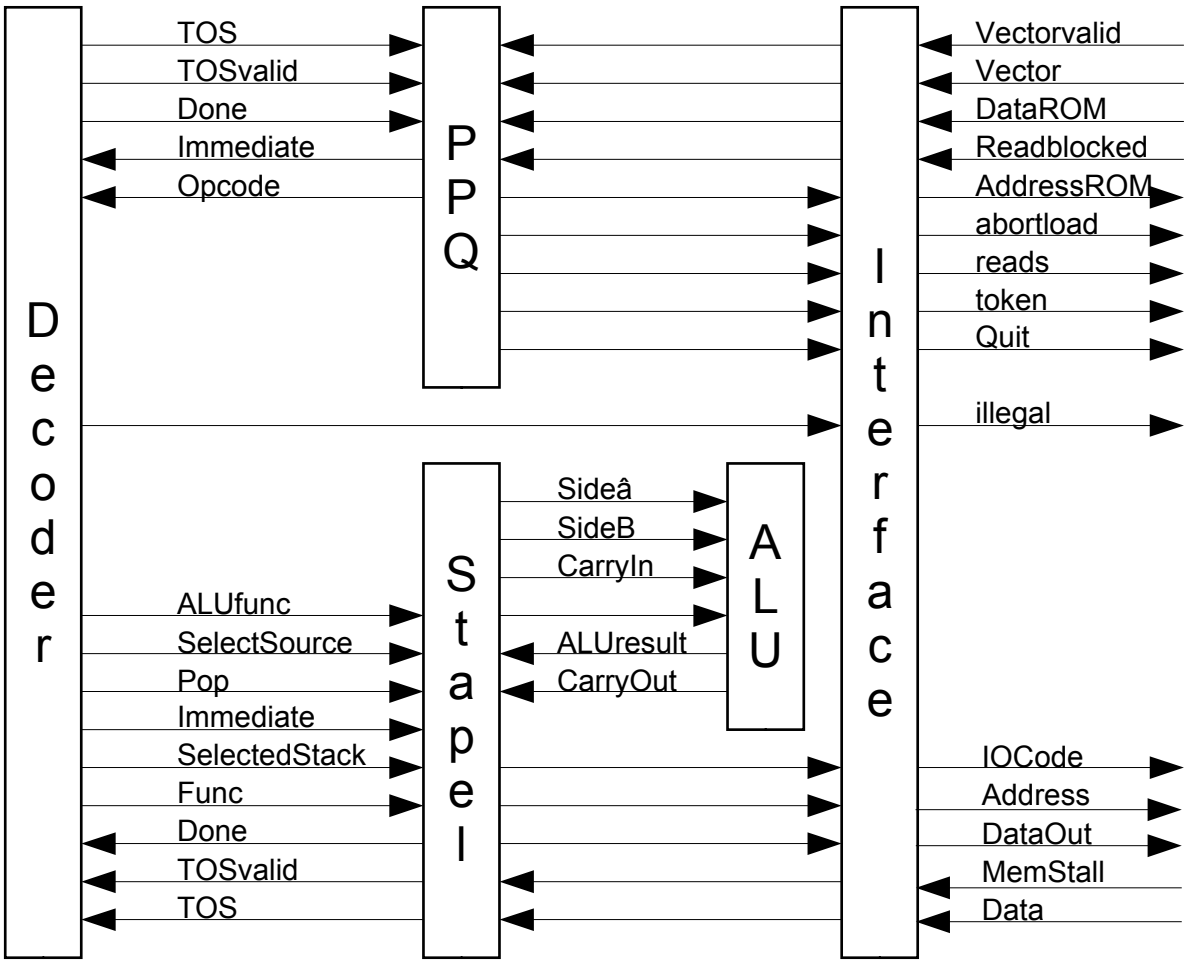


Figure 3: Block diagram of the core

The following introduces the various function groups of the core.

2.1 The ALU

The interface:

```
entity ALU is
      port (nReset: in std_ulogic;)                    -reset
            Sideâ: In DataVec;                         -first operand
            SideB: In DataVec;                         -second operand
            AluResult: out DataVec;                    -result
            CarryIn: in std_ulogic;                    -Carry in
            CarryOut: out std_ulogic;                  -Carry out
            AluFunc: in AluFuncType);                  -Opcode
```

```
end entity ALU;
```

The incoming operands, as well as the incoming transfer, and tax information
Afunc simultaneously evaluate four processes. The two most significant bit
the tax information select from the partial results, table 10, the fact
desired result.

Process task	
Arith	Addition or subtraction
relop	the Relation von *Sideâ, CarryIn on 0*
shifts	Sideâ to SideB job moves to the left or right
logic	Performs Boolean operations from or a Multiplication

Table 10: The function groups of ALU

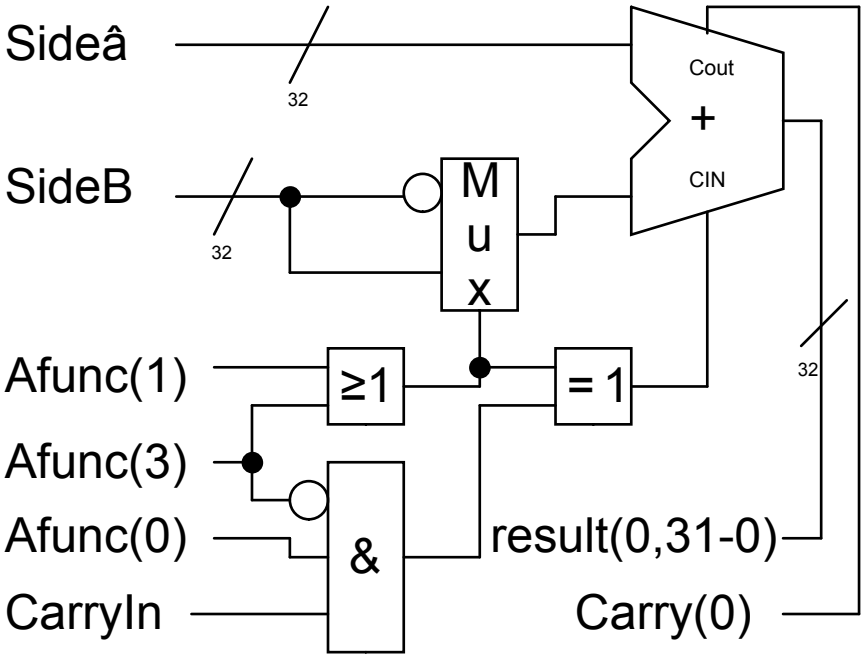


Figure 4: The Adder\/Subtrahierer

The Adder\/Subtrahierer is shown in the image. Afunc(1) and Afunc(3) indicate whether it relates to an addition or subtraction; Subtraction is the second
Operand negates added. Afunc(3) and Afunc(0) specify whether CarryIn is used or not. The obtained input transfer for the Adder is a subtraction
inverted created. The results are result(0) and Carry(0).

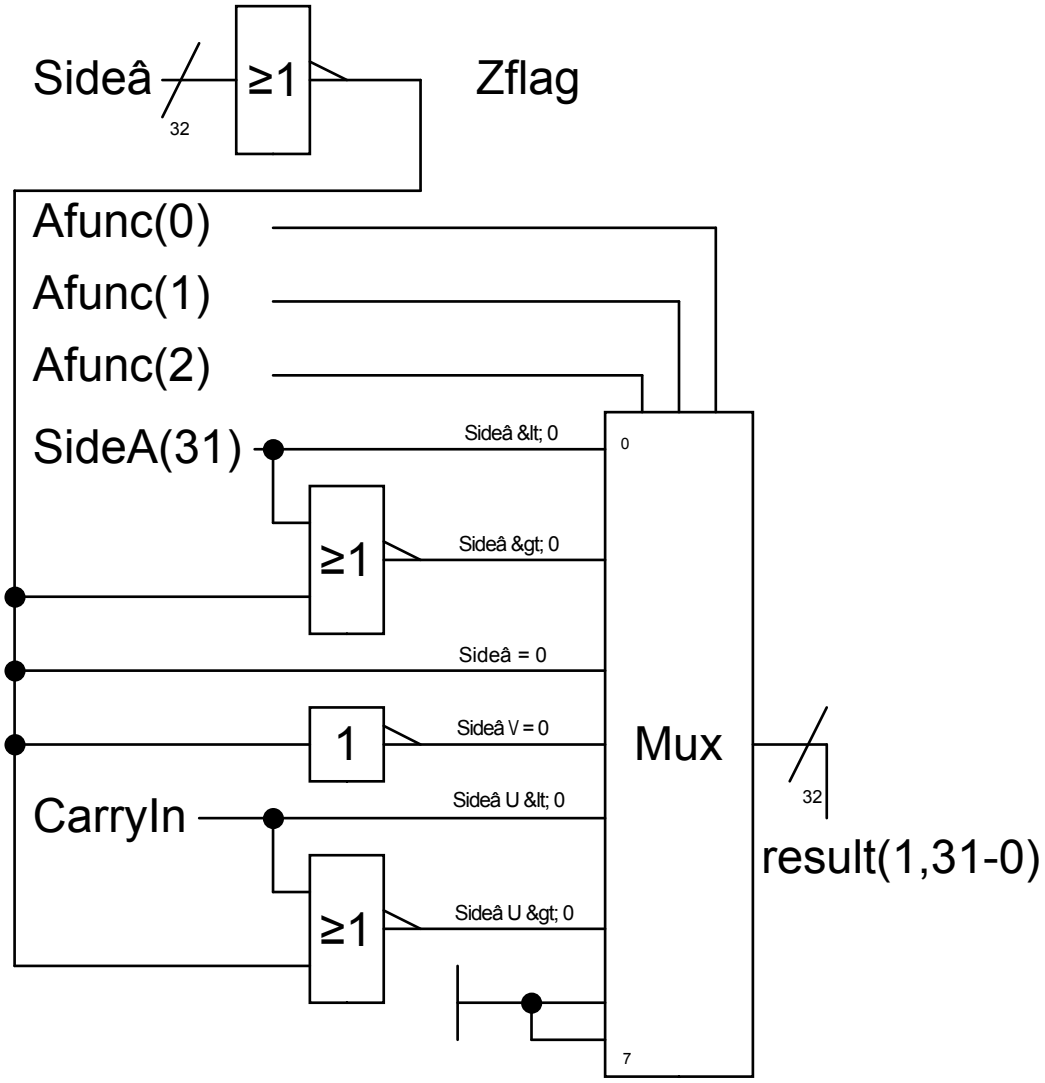


Figure 5: shipping stabilizer relop

In comparison operations, Zflag is from the MSB Sideâ, CarryIn and help size (true if Sideâ = 0), the comparison result determined,-1 for true, 0 for false.

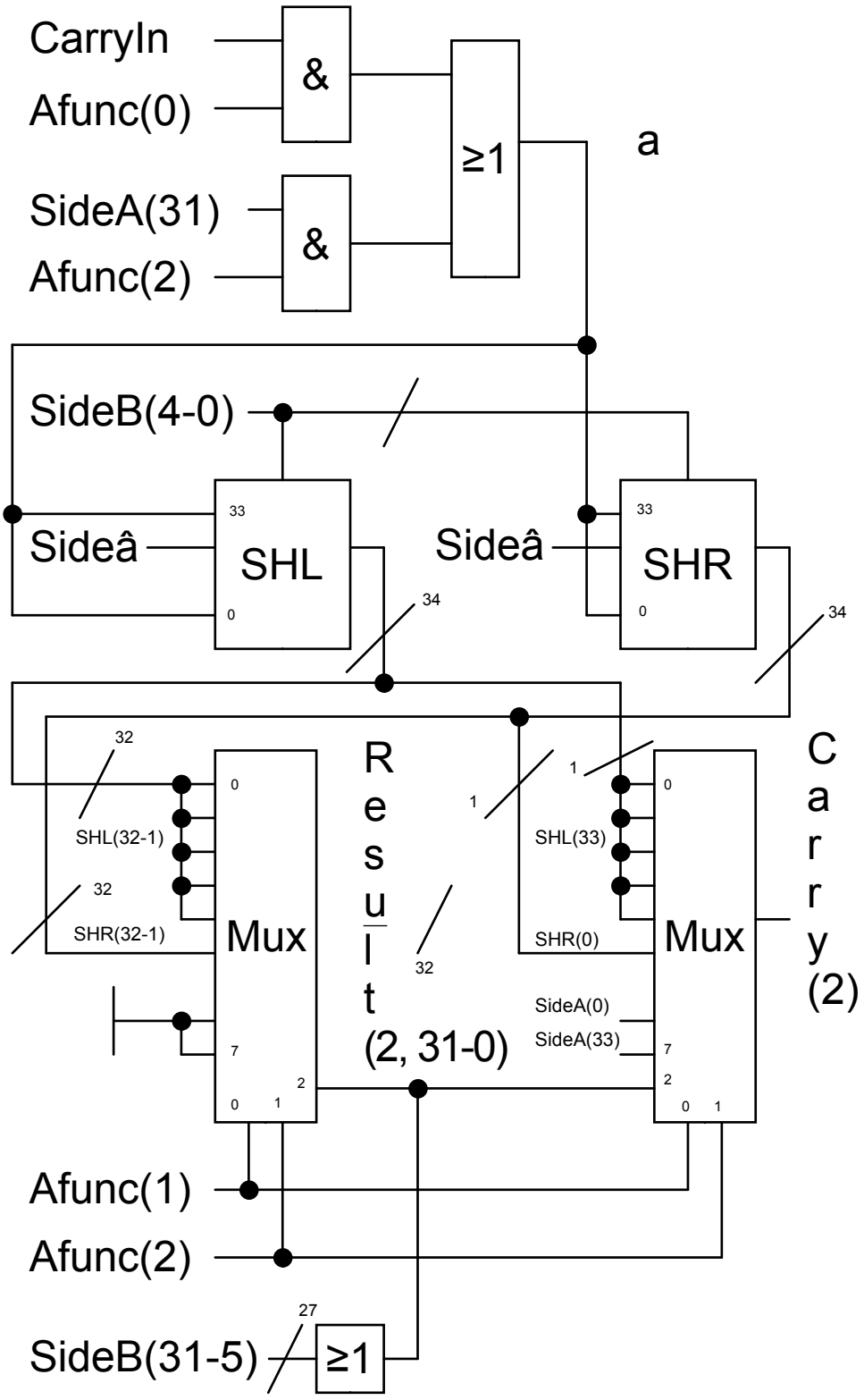


Figure 6: Shift register

The shift operation used a rechtsseitiges and a ridging shift register.

As a CarryOut that bit stems most recently from the selected shift register, was pushed.

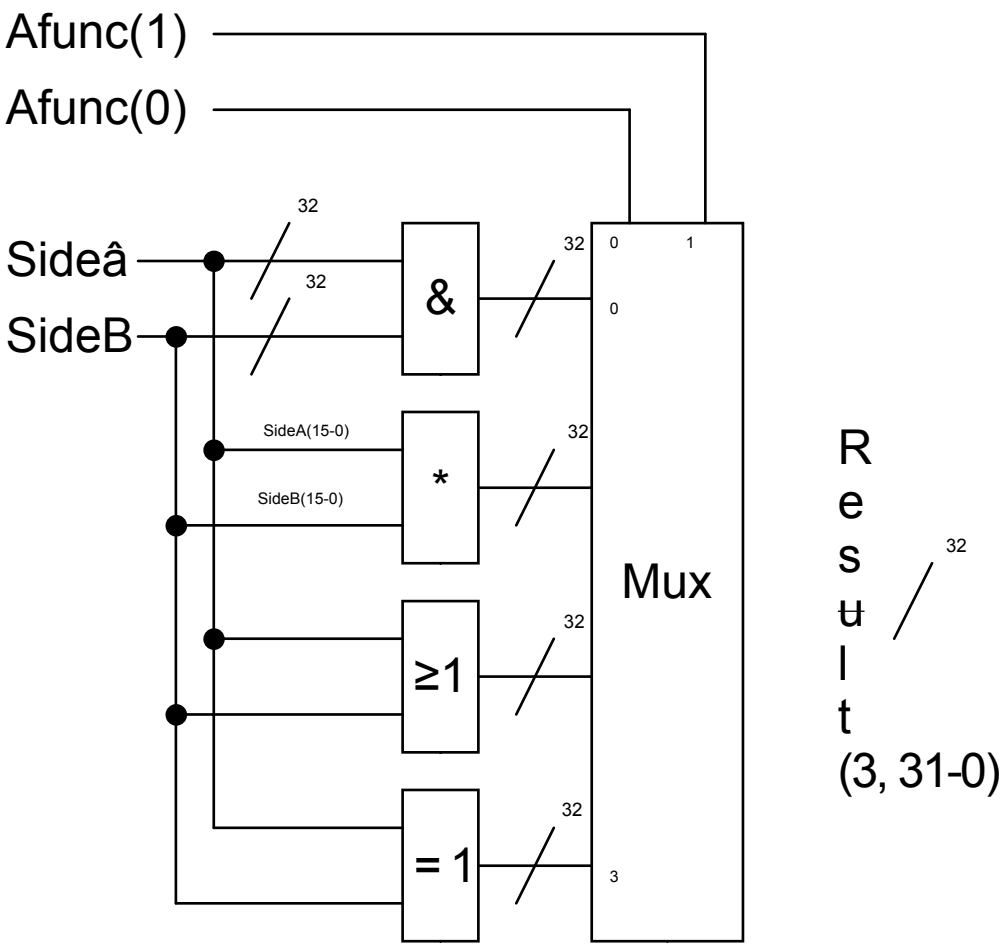


Figure 7: Logic and multipliers

The logical operations are self explanatory. The operation was not not extra realized, it is equivalent to

not Sideâ Sideâ of xor-1 =

The unsigned multiplication process 16-bit operands and returns a 32-bit Product. You just needed a clock.

The provided operations are listed in the following tables 11 to 14.

Adder \/ Subtrahier operation er		***Transmis g***	***comment***
AddALU (00100)	Sideâ + SideB	calculate t	
ArgumentException: The incoming token has e	Argument Exception: The incoming token has e	Argument Exception: The incoming token has e	Argument Exception: The incoming token has e ArgumentException: The incoming token has e
ArgumentException: The incoming token has e	Sideâ + SideB + CarryIn	calculate t	Get a new access token from the Author
SubcALU (00111)	Sideâ - SideB - CarryIn	calculate t	
AddALUnC (00000)	Sideâ + SideB	CarryIn	Adressrechnun g
SubALUnC (00010)	Sideâ - SideB	CarryIn	Adressrechnun g

Table 11: Addition\/subtraction

Ver stabilizer	***Essence n***	***Transmis g***	***comment***
LtALU (01000)	Sideâ < 0	CarryIn	MSB
GtALU (01001)	Sideâ > 0	CarryIn	MSB nor Zflag
EquALU (01010)	Sideâ	CarryIn = 0	Zflag
NequALU (01011)	Sideâ \/ 0	= CarryIn	Emergency Zflag
UltALU (01100)	Sideâ < 0	CarryIn	Carry
UgtALU (01101)	Sideâ > 0	CarryIn	Carry nor Zflag

Table 12: Relations

Shift operation operation		***Transmis g***	***comment***
LshiftALU (10000)	Sideâ < SideB	calculate t	
LshiftCALU (10001)	Sideâ, CarryIn << SideB	calculate with excess tag t	
RshiftALU (10010)	Sideâ > SideB	calculate t	
RshiftCALU (10011)	CarryIn, Sideâ >>> SideB	calculate with excess tag t	
RshiftAALU (10100)	Sideâ > SideB (: = 1).	calculate t	arithmetic Shift

Table 13: Schiebeoperationen

Logical Operation	***Operation***	***Transmis g***	***Komment AR***
AndALU (11000)	Sideâ & SideB	deleted	
MultALU (11001)	Sideâ born sieve	deleted	
OrALU (11010)	Sideâ | SideB	deleted	
XorALU (11011)	Sideâ ^ SideB	deleted	

Table 14: Logical operations and multiplication

2.2 The stack management

The interface:

```
entity theStacks is
   generic (constant CacheIndexBitWidth: integer: = 10);    -log. of depth of
                                                               cache
   port (nReset: in std_ulogic;)                            -Reset
         Clock: in std_ulogic;                              -clock signal
         MemStall: in std_ulogic;                           -memory stall
         Pop: in std_ulogic_vector(1_downto_0);             -pop count
         SelectSource: in std_ulogic_vector(3_downto_0);    -select a visible
                                                           element of the stack
         SelectedStack: In BinRange;                        -selects the
                                                               stack target
         Data: In DataVec;                                  -date entering
         DataOut: out DataVec;                              -date leaving
         Immediate: In DataVec;                             -immediate value
         Target: out std_ulogic_vector(RAMrange'high_+_1_downto_0); -memory
                                                                      address
         IOCode: out std_ulogic_vector(2_downto_0);         -memory operation
         Tosen valid: out std_ulogic;                       -return address is
                                                               on top of stack
         Ready: out std_ulogic;                             -stack ready for
                                                               next operation
         Func: In StackFuncType;                            -Opcode stack
```

```
        First: out DataVec;                                  -first operand
                                                                on stack
        AFunc: In AluFuncType;                               -Opcode ALU
        SelectALU: in InputOrder_t);                         -input selection
                                                                for ALU
end entity theStacks;
```

2.2.1 Required parameters of a stack

Several parameters describe a stack, are necessary to his referencing, or allow a conclusion on its current state.

- Top — the index of the top stack element (corresponds with indexf)
- Tail — the index of the lowest stack element (corresponds to indexl)
- Append — the index in the free area after tail
- cachelength — the current length of the stack in the local store
- reloadstate — current condition of overflow machines
- stackptr — points to the latest element in the overflow area
- stacklength — the current length of the overflow area

This variable is new in the first operation step (update of the stack) calculated in the second operation step (processing of the stack buffer), where appropriate, bloß read. More Operation steps required the Stack not. Both Operation steps can extension steps (stalls) but independently of each other force.

2.2.2 Update of the stack

Following details are expected from the command decoder:

- $Select_{t-1}$ — the number of the selected stack
- $stackfunc_{t-1}$ — the operation to be applied onto the stack
- pop_{t-1} — the number of top entries, which are to delete off the stack
- $selectalu_{t-1}$ — Information, which values to which inputs of the ALU are laying
- $afunc_{t-1}$ — the operation that you want to perform the ALU

Passed the modified parameters, as well as the information, whether a Extension step (stall1) is necessary.

The Update is only carried out,. If the Processing No Extension step required. Should be the desired stack operation SetSPStack, is forces initialization of the stack. The address is in the top element of the stack the Overflow area expected in thesecond Element the current Length the The stack in the local store overflow area is set to empty. For everyone else Stack operations is cachelength modified and carried out top. The upper end of The stack would be on State, lacking even the lower end.

An overflow occurs or the stack operation SaveStack, a saving of the stack forces in the overflow area, reduces stackptr and stacklength increases, the Letter of last requested. The characteristics of tail, append, cachelength be modified and the State of the reloadstate inhibit set, in the case of SaveStack can additional extension steps be enforced.

No overflow occurs, the machine, table 15, on a state change is to check.

State action	
check	just no extension step runs, the overflow is not empty and at least two entries in the local store are free, force one Extension step and change the State of load
load	is reading the latest element of the overflow area an and modified the affected parameters. Contains the local memory of at least two Entries, check is changed in the State , ansonSten is another Extension step enforced. The local buffer is not empty, is in the State delay changed
delay	a delay step, the new State will check
inhibit	is the current command DROP or 2DROP, will check in the State substitute. This behavior should the programmer a trouble-free change logical stack allow

Table 15: The States of overflow machines

Finally the new tax data be passed ALU that, but only if no Extension step runs. Enforced an extension step (stall1) If stackfunc has the values GetPut, LoadStore, or SetSPStack.

2.2.3 Processing

All parameters calculated at time t, be used, also.

- Select$_t$ the number of the selected stack
- stackfunc$_t$ the operation to be applied onto the stack
- source$_t$ Citing the source, the value date $_{t+1}$supplies (ALU, immediate) stackptr, stacklength +. CAchelength, buffer(source(1), source(0)))

Only if processing does not extension step, is stackfunc, as in Table 16 led, evaluated.

2.2.3 Processing

Operation action	
pushstack	the value of the source that is specified by source is date $_{t+1}$
getput	optional reading of a stack element, or optional override a Stack element. The distinction allows source(2). This operation expects the address - relative to the top element of the stack - in the top Stack element. The second stack element replaces the value of the addressed Element when writing. The indexing of the stack starts at the first Element, which is a parameter for the operation, whose index is 0 This operation automatically recognizes whether the source or the target in the local store or in the overflow. It is accordingly indexr or *indexi or an Adressregister target for memory access. As well* provided the appropriate control signals
loadstore	equivalent to getput, source or Target is the memory
Switchcore,	switch to a different core
readpop	compulsory reading of the latest value of the overflow, the address is the operation step update
writepop	forced writing of last in the overflow, the address is also from the operation step update

Table 16: All operation cycles of the phase processing

The new tax values for memory access are in IOCode for accessing stack in *fetch - optional reading of the stack, and store - for optional write - filed.* All operations that read access on the stack or memory - stackfunc has the Value *getput* or *loadstore* –, need at least one additional Operation step (stall2) to take over the value in the stack.

2.2.4 Other tasks

The program required for conditional jumps and the return from subroutines *prefetch queue the value of top (0),t as well as a guarantee of validity*

$$TOSvalid_{t+1} \leftarrow source_t(3)$$

The Command decodermust informed be,. whether wait must, because Extension step is running, or can continue

$$Ready_{t+1} \leftarrow stall1_t \text{ nor } stall2_t$$

Next are laying the desired data sources to the inputs of the ALU, the Information is available in selectalu, table 17.

selectalu$_t$	***Entrance Sideâ***	***Entrance SideB***
ImIm	Immediate$_t$	Immediate$_t$
FIm	top$_t$(0)	Immediate$_t$
ImF	Immediate$_t$	top$_t$(0)
SF	top$_t$(1)	top$_t$(0)

Table 17: The operand assignments of ALU inputs

2.3 The command decoder

The decoder, table 18, assumes a single program prefetch queue complete Instruktion – the Operation code and one possibly angehängten immediate value - and translates these into a fixed set of control statements for their processing . This phrase and (assumed) direct values are to the Stack management weiteRGTo give. The decoder is ready with the stack management synchronized and only then delivers a new set, if the batch administration is ready. This synchronization signal is program prefetch queue on the redirected, TOSvalid and top are also (0$_t$) - the top$_t$ element of the stack and be Confirmation signal - transmitted unchanged through.

2.3 The command decoder

ID		***Set***						
Opcode	Select Mnemnonik		P o p	source	stackfunc	Select-ALU	Afunc Operand	
0000000	NOP	Opcode () (7)	0	0100	NopStack	ImIm	0	AddALUnC
0000001	!	Opcode () (7)	2	0111	LoadStore	ImIm	0	AddALUnC
0000010	@.	Opcode () (7)	0	0011	LoadStore	ImIm	0	AddALUnC
0000011	C!	Opcode () (7)	2	0101	LoadStore	ImIm	0	AddALUnC
0000100	C@.	Opcode () (7)	0	0001	LoadStore	ImIm	0	AddALUnC
0000101	H!	Opcode () (7)	2	0110	LoadStore	ImIm	0	AddALUnC
0000110	H@.	Opcode () (7)	0	0010	LoadStore	ImIm	0	AddALUnC
0000111	DEPTH	Opcode () (7)	0	0100	PushStack	ImIm	0	AddALUnC
0001000	DROP	Opcode () (7)	1	0100	NopStack	ImIm	0	AddALUnC
0001001	2DROP	Opcode () (7)	2	0100	NopStack	ImIm	0	AddALUnC

ID		***Set***						
Opcode	Mnemnonik	Select	Pop	source	stackfunc	Select-ALU	Afunc Operand	
0001010	NIP	Opcode(7)	2	00 Opcode (7)) 0	PushStack	ImIm	0	AddALUnC
0001011	PICK	Opcode(7)	0	0011	GetPut	ImIm	0	AddALUnC
0001100	PUT	Opcode(7)	2	0111	GetPut	ImIm	0	AddALUnC
0001101	VAL	Opcode(7)	0	0110	PushStack	ImIm	PPQ	AddALUnC
0001110	DUP	Opcode(7)		00 Opcode (7)) 0	PushStack	ImIm	0	AddALUnC
0001111	OVER	Opcode(7)	0	00 Opcode (7)) 1	PushStack	ImIm	0	AddALUnC
0010000	R @.	Opcode(7)	0	00 not Opcode (7)) 0	PushStack	ImIm	0	AddALUnC
0010001	R1 @.	Opcode(7)	0	00 not Opcode (7)) 1	PushStack	ImIm	0	AddALUnC
0010010	SAVE	Opcode(7)	0	0100	SaveStack	ImIm	0	AddALUn ArgumentException: T
ArgumentException: Th[illegible]		ArgumentException[illegible]		0100 [illegible]	NopStack [illegible]	ImIm [illegible]	0 [illegible]	AddALUnC [illegible]
0010100	SP!	Opcode(7)	0	0100	SetSPStack	ImIm	0	AddALUnC
0010101	SP @.	Opcode(7)	0	0101	PushStack	ImIm	0	AddALUnC

2.3 The command decoder

ID		***Set***						
0010110	+	Opcode(7)	2	0111	PushStack	SF	0	AddALU
0010111	-	Opcode(7)	2	0111	PushStack	SF	0	SubALU
0011000	1 +	Opcode(7)	1	0111	PushStack	FIm	1	AddALUnC
0011001	1	Opcode(7)	1	0111	PushStack	FIm	1	SubALUnC

ID		***Set***						
Opcode	Mnemnonik	Select	Pop	source	stackfunc	Select-ALU	Afunc	Operand
0011010	2.	Opcode(7)	1	0111	PushStack	FIm	1	LshiftALU
0011011	2/.	Opcode(7)	1	0111	PushStack	FIm	1	RshiftAALU
0011100	AND	Opcode(7)	2	0111	PushStack	SF	0	AndALU
0011101	CELL +	Opcode(7)	1	0111	PushStack	FIm	4	AddALUnC
0011110	HALF +.	Opcode(7)	1	0111	PushStack	FIm	2	AddALUnC
0011111	INVERT	Opcode(7)	1	0111	PushStack	FIm	-1	XorALU
0100000	LALT	Opcode(7)	2	0111	PushStack	SF	0	LshiftALU
0100001	NEGATE	Opcode(7)	1	0111	PushStack	ImF	0	SubALU
0100010	OR	Opcode(7)	2	0111	PushStack	SF	0	OrALU
0100011	RSHIFT	Opcode(7)	2	0111	PushStack	SF	0	RshiftALU
0100100	XOR	Opcode(7)	2	0111	PushStack	SF	0	XorALU
0100101	+B	Opcode(7)	2	0111	PushStack	SF	0	AddCALU
0100110	B	Opcode(7)	2	0111	PushStack	SF	0	SubCALU
0100111	LSHIFTC	Opcode(7)	2	0111	PushStack	SF	0	LshiftCALU
0101000	RSHIFTC	Opcode(7)	2	0111	PushStack	SF	0	RshiftCALU
0101001	0<!	Opcode(7)	0	0111	PushStack	FIm	0	LtALU
0101010	0<	Opcode(7)	1	0111	PushStack	FIm	0	LtALU
0101011	0=!	Opcode(7)	0	0111	PushStack	FIm	0	EquALU
0101100	0=.	Opcode(7)	1	0111	PushStack	FIm	0	EquALU
0101101	CMP	Opcode(7)	0	0111	PushStack	SF	0	SubALU

2.3 The command decoder

ID		***Set***						
0101110	CONST-10	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC

ID		***Set***						
Opcode	Mnemnonik	Select	Pop	source	stackfunc	Select-ALU	Afunc	Operand
0101111	CONST-15	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0110000	CONST-5	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0110001	CONST-14	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0110010	CONST-9	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0110011	CONST-13	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0110100	CONST-8	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0110101	0 < >!	Opcode(7)	0	0111	PushStack	FIm	0	nEquALU
0110110	0 < >	Opcode(7)	2	0111	PushStack	SF	0	nEquALU
0110111	0 >!	Opcode(7)	0	0111	PushStack	FIm	0	GtALU
0111000	0 >	Opcode(7)	1	0111	PushStack	FIm	0	GtALU
0111001	CONST-12	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0111010	CONST-7	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0111011	CONST-11	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0111100	CONST-6	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
0111101	EXIT	Emergency Opcode(7)	1	1000	NopStack	ImIm	0	AddALUnC
0111110	CALL	Emergency Opcode(7)	0	0110	PushStack	ImIm	PPQ	AddALUnC

2.3 The command decoder

ID		***Set***						
0111111	TRAP	Emergency Opcode(7)	0	0110	PushStack	ImIm	PPQ	AddALUnC
1000000	BRANCH	Opcode(7)	0	0100	NopStack	ImIm	0	AddALUnC
1000001	0BRANCH!	Opcode(7)	0	1000	NopStack	ImIm	0	AddALUnC
1000010	0BRANCH	Opcode(7)	1	1000	NopStack	ImIm	0	AddALUnC

ID		***Set***						
Opcode	Mnemnonik	Select	Pop	source	stackfunc	Select-ALU	Afunc	Operand
1000011	B@.	Opcode(7)	0	0111	PushStack	ImIm	0	SubCALU
1000100	B!	Opcode(7)	1	0111	NopStack	ImF	0	SubALU
1000101	BREAK	Opcode(7)	0	0100	NopStack	ImIm	0	AddALUnC
1000110	VALH	Opcode(7)	0	0100	PushStack	ImIm	PPQ	AddALUnC
1000111	CONST15	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001000	CONST14	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001001	CONST13	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001010	CONST12	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001011	CONST11	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001100	CONST10	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001101	CONST9	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001110	CONST8	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1001111	CONST7	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1010000	CONST6	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1010001	CONST5	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1010010	CONST4	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUn

2.3 The command decoder

ID		***Set***						
								C
1010011	CONST3	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC
1010100	CONST2	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC
1010101	CONST1	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC
1010110	CONST0	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC
1010111	CONST-1	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC
1011000	CONST-2	Opcode(7)	0	0100	PushStack	mIm	0	AddALUnC

ID		***Set***						
Opcode	Mnemnonik	Select	Pop	source	stackfunc	Select-alu	Afunc	Operand
1011001	CONST-3	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1011010	CONST-4	Opcode(7)	0	0100	PushStack	ImIm	0	AddALUnC
1011011	U0 <	Opcode(7)	1	0111	PushStack	FIm	0	ULtALU
1011100	U0 >	Opcode(7)	1	0111	PushStack	FIm	0	UGtALU
1011101	H.	Opcode(7)	2	0111	PushStack	SF	0	MultALU
1011110	SETPC	Opcode(7)	1	1000	NopStack	ImIm	0	AddALUnC
1011111	SWITCH	Opcode(7)	0	0100	SwitchCore	ImIm	PPQ	AddALUnC
1100000	GETCOREID	Opcode(7)	0	0110	PushSta		CK	ImIm PPQ
AddALUn		C		other	Opcode(7)	0100	NopStack	ImIm 0

AddALUn

C

Table 18: All commands with the necessary tax information
(PPQ is the immediate operand of the prefetch queue program)
97 Instructions are so far forgiven, 31 possible further still undefined. Is one
undefined instruction, is it like a NOP command, in addition, it activ[illegible] i
predefined stack used

should be or - 1 - the alternative stack, it can by programmer
the prefix \"A:\", which deliver the command forward is, are enforced.
The importance of source - table is 19, information to the data source - dependent
stackfunc. source(3) is always a requirement of the top stack element, the

stackfunc	***source(0)***	***source(1)***	***source(2)***	***comment***
PushStack	1	1	1	ALU
PushStack	0	1	1	Immediate operand
PushStack	1	0	1	stackptr
PushStack	0	0	1	Stacklength + cachelength
PushStack	element	Stack	0	A Element from the Stack buffer
GetPut	--	--	0	Optional Stack element read
GetPut	--	--	1	Optional Stack element To write
LoadStore-		--	0	Read random memory cell
LoadStore-		--	1	Optional Memory cell To write

Table 19: The meanings of the tax information source for processing cycles

The allocation of codes obeys no system, but took place arbitrarily.

2.4 Program prefetch queue

The interface:

```
entity ProgramCounter is
  generic (constant TableBitWidth: integer: = 4);
  port (nReset: in std_ulogic;)                                    -system reset
        Clock: in std_ulogic;                                      -system clock
        abortload: out std_ulogic;                                 -abort current
                                                                      fetch cycle
        ReadBlocked: in std_ulogic;                                -fetch cycle
                                                                      delayed
        Opcode: out std_ulogic_vector(7_downto_0);                 -coded operation
        Data: In DataVec;                                          -incoming data
        Address: out std_ulogic_vector(RAMrange'high_-_1_downto_0);-memory
                                                                           address
        Immediate: out DataVec;                                    -immediate operand
        Reads: out std_ulogic;                                     -memory operation
        Done: in std_ulogic;                                       -core ready
```

```
        TOS: in DataVec;                                    -return address
        Tosen valid: in std_ulogic;                         -return address
                                                               available
        Token: out std_ulogic;                              -service routine
                                                               running
        Quit: out std_ulogic;                               -vector processed
        Vector: in std_ulogic_vector(TableBitWidth_-_1_downto_0);
                                                            -interrupt vector
        VectorValid: in std_ulogic);                        -interrupt vector
                                                               present
end ProgramCounter;
```

This unit delivers the next command for the command decoder. Since the operation codes have a fixed length of a byte, but some one operand with a length attached two or four bytes is two memory accesses may be necessary to the Command to read completely. This means an additional extension step, the -can be avoided with a prefetch queue. Not only the word, the the The next command operation code contains, is read, but at least the immediately following Word. These advance read words form the prefetch queue, the next command is removed from the. The number of reads is this Method somewhat larger than with an unbuffered method, as for example in the case of a The contents of the buffer is invalid jump statement, therefore, too much has been read, on the other hand need no additional extension step long commands. A buffer of Length two brings none in the case of two successive long commands Improvement, he must at least s dIE length three are. A further enlargement bRbegins no benefits, but merely increases the number of anticipated on the store. A optimum effect to e rzielen, it is assumed that the commands without gaps each other follow, are aligned so not on word boundaries.

The unit has several interfaces, including one to the store.

- Data contaAEs a read Word (32 bit)
- abortload the most recently launched read request is immediately canceled
- Reads the read request is active, if the buffer is not full
- ReadBlocked is active, if still no word be read despite read request could
- Address the address of the word to be read is an independent, after each Read operation, increasing pointer which should empty the buffer be on the Is to be put on the value of the program counter.

Interruptions may be requested on the following interface.

- Vector the Address one The word, in which the Address the Instant routine is stored
- VectorValid signals a valid value of the vector
- ToKen signalisiert the controller that the service routine is started

2.4 Die program prefetch queue

- Quit VectorValid signals the controller which you can adoption of the vector, reset

Remains still the interface command decoder and stack management.

- Opcode an 8-bit operation code
- Immediate the 32-bit operand of command
- Done the decoder is ready to take over
- TOS the topmost element of the current stack
- TOSValid signals a valid value of TOS

The basic process is to first update, then the next command the buffer extract, the program counter increase, where appropriate, another word for the buffer request, and most recently the operation code and its operand to the decoder to pass on. Should an extension step be necessary - because the buffer is empty or maintained on the stack is — the NOP command to the decoder is sent, so that the following phases can continue unhindered.

Some commands, table 20, are at least partly processed in this unit or influence on the prefetch queue or the program counter specifically, what a delete causes of the buffer.

Command	***Action***

EXIT	While (!)(TOSValid);
	PC = TOS;
	empty(Puffer);

Offset CALL	Immediate = PC; PC += offset; empty(Puffer);
addr TRAP	Immediate = PC; PC = born addr; empty(Puffer);
vector *Vector-valid*	Immediate = PC; PC = b. vector; empty(Puffer);
offset BRANCH	PC += offset; empty(Puffer);
offset *0BRANCH!*	While (!)(TOSValid); If (TOS) == (0) { PC += Offset; empty(Puffer); }

2.4 Program prefetch queue

Command	***Action***
offset 0BRANCH	while (!)(TOSValid); If (TOS) == (0) { PC += Offset; empty(Puffer); }
STOP	While (!)(Vektorvalid); PC++;

Table 20: Influence growing commands

Table 21 provides commands that need more than five steps of operation.

Command	***stall***	***Comment***
EXIT	3	The prefetch queue must 3 Bars wait,. until the Return adress in TOS appears
Offset CALL	1	The prefetch queue requires one additional measure to \"PC + offset\" to calculate
addr TRAP	1	The prefetch queue requires one additional measure to to read the vector in addr
offset BRANCH	1	The prefetch queue requires one additional measure to \"PC + offset\" to calculate
offset *0BRANCH!*	3	The prefetch queue must wait 3 cycles, until in the TOS Test size appears
offset *0BRANCH*	3	The prefetch queue must wait 3 cycles, until in the TOS Test size appears
PICK	2	Update and Processing need one additional clock for the addressing of the stack and Acquisition of date
PUT	1	The update requires an additional clock
[C | [H] @.	> = 2	The update requires an additional clock and the Processing at least one further to the Acquisition of date
[C | [H]!	> = 1	The update requires one additional measure, the Any further processing

Command	***stall***	***Comment***
SAVE	*cachelength*	Outsourcing in the store requires additional Bars
SP!	> = 1	The storage from the storage needs may additional bars to fill the stack buffer

Table 21: Commands that force stalls

3 The processor

Source	***Comment***
mycpu.vhd	The processor
theCore.vhd	The processor core
UART.vhd	The UART
FiFo.vhd	The FiFo of UART
counter.vhd	Counter and timer
IntVectors.vhd	The interrupt Controller
ROMcode.vhd	The Rome
vhdl_syn_bl4.VHd	the RAM interface
Global.vhd	Global Declarations

Table 22: The source files of the processor

The following introduces the integrated units.

3.1 UART

The interface:

```
UART is entity
      port (nReset: in std_ulogic;)                    -reset
            CLK: in std_ulogic;                        -system clock
            RxD: in std_ulogic;                        -UART input
            TxD: out std_ulogic;                       -UART output
```

```
            RTS: out std_ulogic;                          -request to send
            CToS: in std_ulogic;                          -clear to send
            ReadyR: out std_ulogic;                       -receiver ready
            ReadyT: out std_ulogic;                       -ready transmiter
            Address: in std_ulogic_vector(2_downto_0);      -port address
            Data: in std_ulogic_vector(23_downto_0);        -parallel input
            DataOut: out std_ulogic_vector(23_downto_0);    -parallel output
            Writes: in std_ulogic;                        -write to UART
            Reads: in std_ulogic);                        -read from UART
end UART;
```

The UART contains a reception and a transmitter, which work both parts of each other regardless. A filter is the recipient, the one sent bit correctly detects and synchronizes the receiver clock with the received signal. The Sender is a clock generator, whose clock the length of time of a sent bit pretending. Both units have ever a parallel port, to be sent on the or data received can be transported for further processing.

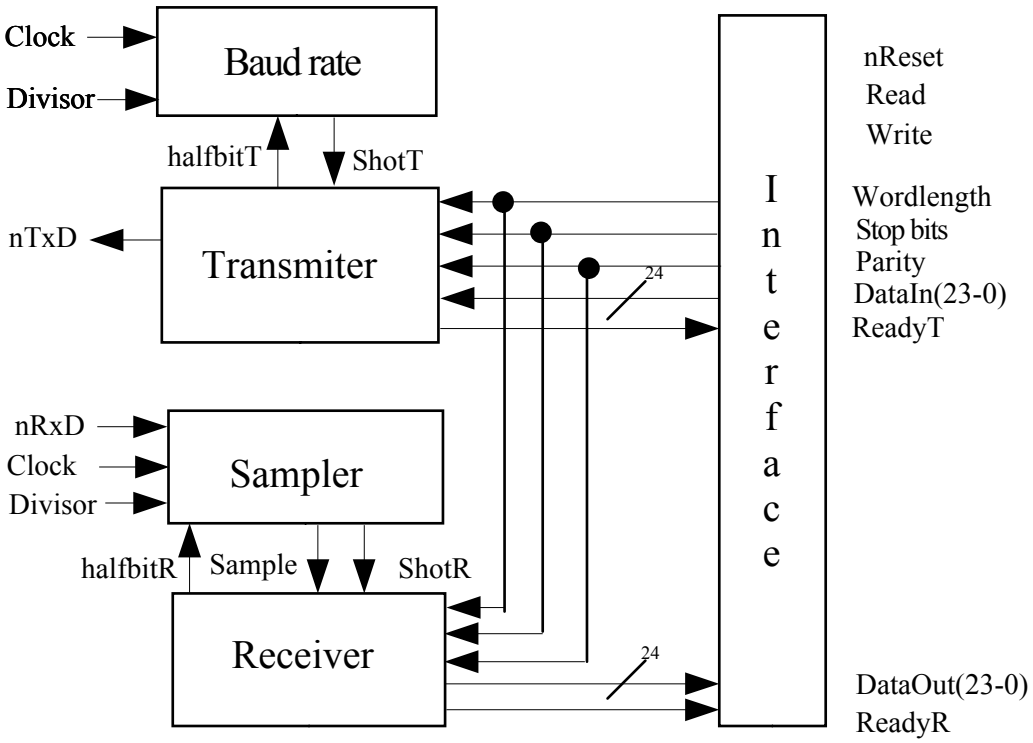


Figure 8: block diagram of UART

3.1.1 The interface

The 9 Bit date on *Data* is by Enable by *Writes* of the Transmiter taken over. The ninth bit indicates whether a masking character advance sent should be, because the remaining bits represent a control character. Once the transmiter is again ready to send, he enabled ReadyT. This signal can be used as break request used. It is automatically disabled by writes.

DataOut the received date can be acquired, the 7 or 8 bits long is. The ninth bit must be reset, otherwise it shows a bad reception the date. A reading of the DataOut signal reads is enabling initiated, which automatically the signal ReadyR is back, which in turn one complete empfangenes Word in the Receiver informs. *ReadyR* can as Used interrupt request.

The signals listed now, table 23, concerning both the reception and the transmitter.

Signal	***Importance***
Wordlengt h	the length of a to be sent or received date. 1 8 Bit date 0 7 Bit date
Parity	Indicates whether a test bit is necessary and its parity 00 use no parity bit 10 Parity bit add on even parity 01 Parity bit add on odd parity 11 no significance
Stop bits	Specifies how many stop bits exit a sign 00 2 Stop bits 01 1 Stop bits 10 1.5 Stop bits 11 no significance
Divisor	A freely selectable 24-bit constant with the system clock is to share, to the double baud rate to generate.
Clock	The system clock
nReset	The General reset signal for all units valid

Table 23: The parameters of the interface

3.1.2 The transmitter

3.1.2.1 The baud rate generator

He is no more than a programmable divider, which runs synchronously with the transmiter. As long as the transmiter is waiting for a write command (signal writes), he is inactive. The Generator starts, once the Datum zum sending is ready, and then runs synchronously until the date is completely transferred. Then he is again inactive.

It is possible to halve the amount of time for a bit during operation. That serves Signal halfbitT, which is provided by the transmiter. This possibility is necessary to send 1.5 stop bit. Every time when the work counter to download has, will be assessed this signal, and the counter appropriately initialized either with the double value of the divisor for a whole bit, or divisor for a half a bit.

3.1.2.2 Of the transmiter

He is realized by a finite state machine. In the initial quiescing idle is. checks whether the recipient requires the broadcast of XON or XOFF, and sends it If necessary. Otherwise wait for a date. An is whether is checked, an Escape character is required, and this, where appropriate, that sent immediately Date.

The send operation is initiated, where the 11-bit wide transmit buffer with the date,. Start, Parität-and first stop bit initialized sending, change to the State will. This condition will leave only if the send buffer is empty, so only more Contains zeros. Result State is SendingDone, if no further stop bit is necessary, otherwise, stopping is interposed in the an another half or whole Stop bit is sent. In SendingDone, only the end of the transmission is. waits and changed initial quiescing idle.

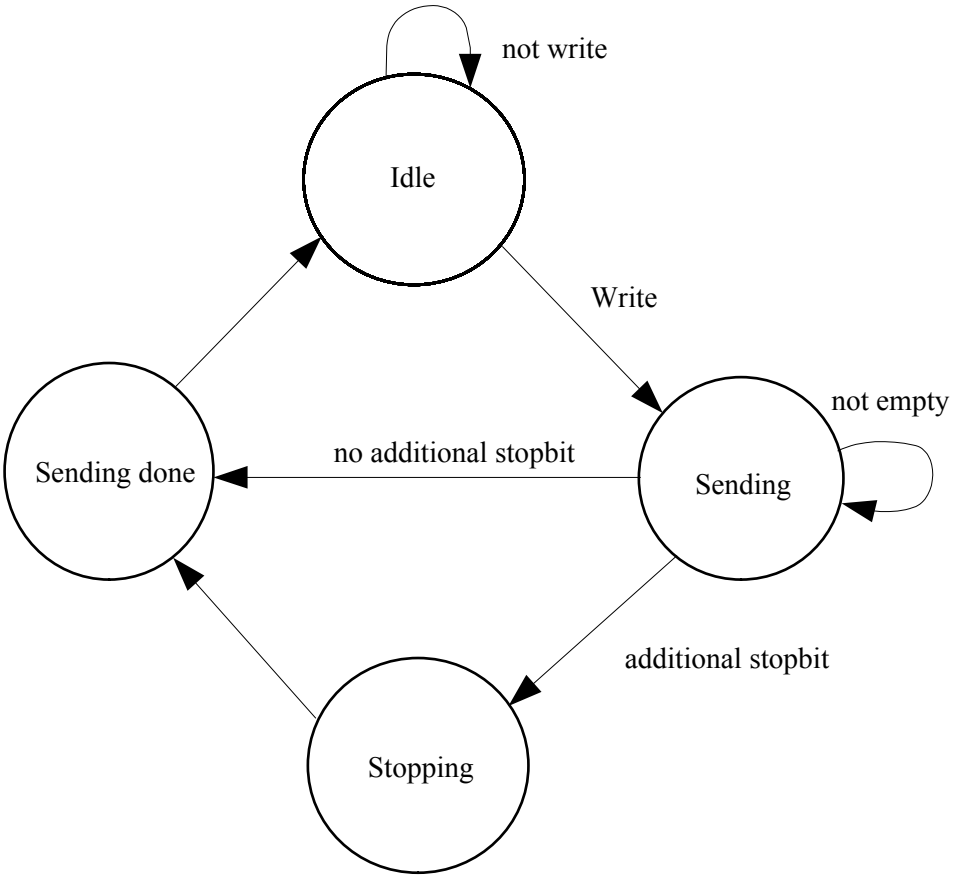


Figure 9: Automatic of the Transmiters

3.1.3 The tuner

3.1.3.1 Of the filter

The filter starts the detection and clock generation, once one the receiver input logical 0 appears. These must be at least for the half time of a bar, to clearly recognize a start bit. This is not the case, the filter waits on a It produces a stable clock for the receiver start bit, when recognized start bit. The Filtering of the input signal starts again at the beginning of each bar. Is During this time, at least for the half period detected a logical 0,. is one 0, otherwise one 1, on the Receiver redirected. The continuous Clock generation stops when the receiver is in the idle state.

3.1.3.2 Of the receiver

In the start quiescing idle, it checks whether or not the receive buffer is free and forces, where appropriate, the consignment of XOFF or XON. Otherwise, it waits

3.1.3 The tuner

The proper detection of the start bit through the filter, automatic and is only in the State ReceiveData on reception of the date. There is checked whether all data bit received and decided whether an additional parity bit or one or two Stop bits follow. In the State of ReceiveParity is the correct reception by one logical 0 in the bit marked 8 of the receive buffer, in case of failure, it is to 1 used. Then, it is checked whether one or two stop bits are outstanding. In the State *ReceiveStopbit* is the Reception the first Stop bit wait. In the State *ReceiveStopbits the reception of the last Stopbit will wait the full* Reception of the date is indicated by ReadyR and quiescing idle substitute.

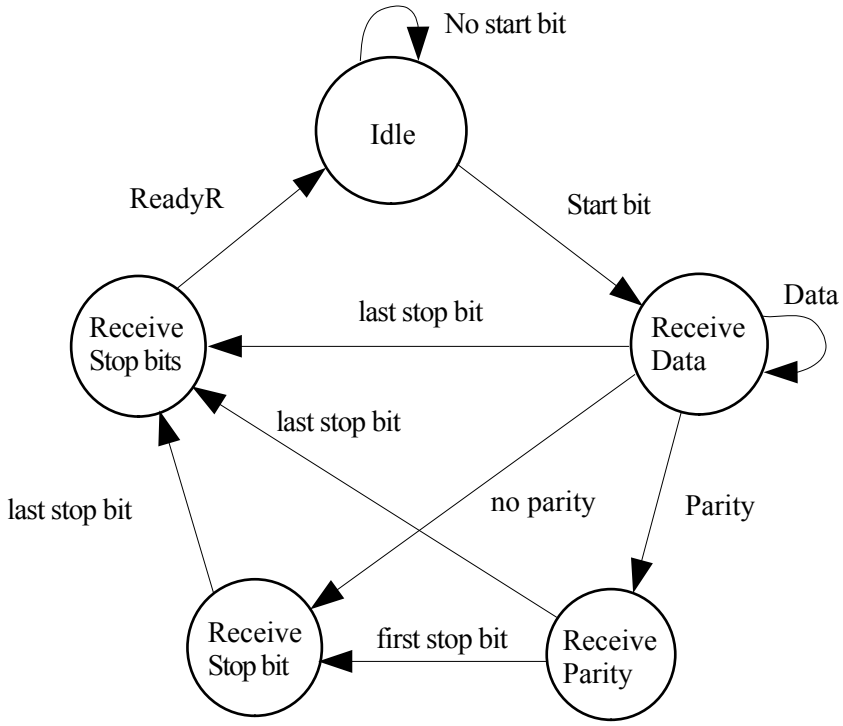


Figure 10: Slot of the receiver

This unit shows the following addresses listed in table 24.

Address e	***Comment***
ffffffe0H	Data port
ffffffe4H	Definitionsport
ffffffe8H	Prescaler of the baud rate generator, 24 Bit
fffffecH	Decode
fffffff0H	The escape character escape
fffffff4H	The XON character
fffffff8H	The XOFF character

Table 24: The ports of UART

Table 25 defines the allocation of data port.

Bit	***significance***
7-0	The date
8	Is it set, it forces the broadcast of an escape character, at one Write access It is set, it indicates a faulty reception when a read access

Table 25: Importance of bits of writing and reading ports

Table 26 defines the usage of definition or decode.

3.1.3 The tuner

Bit	***Values***
1 - 0	Parity: 0 none 1 odd 2 even
3-2	Stop bit: 1 one stop bit 0 two Stop bit 2 1.5 Stop bit
4	Word length: 0 7 Bit 1 8 Bit

Table 26: Meaning of the bits of the definition or status ports

Table 27 defines the allocation of ports escape character, XON, XOFF

Bit significance	
7-0	Bit pattern of the character
8	Is It used,. are the made progress Bit without Importance

Table 27: Meaning of the bits of the port of escape characters

Without FiFo occasionally loss of data. To avoid this, were
Transmitter and receiver are equipped with ever a FiFo.

3.1.4 Of the FiFo

The interface:

```
entity is FiFo
  generic(constant_WordLength:_integer_:=_9;)          -size of word
          constant ldFiFoSize: integer: = 3);          -default FiFo size 8
  port (nReset: in std_ulogic;)                         -reset
        Clock: in std_ulogic;                           -system clock
```

```
        put: in std_ulogic;                          -write into FiFo
        get: in std_ulogic;                          -read from FiFo
        Data: in std_ulogic_vector(WordLength_-_1_downto_0);-incoming data
        DataOut: out std_ulogic_vector(WordLength_-_1_downto_0);-outgoing data
        ArgumentException: The incoming token has expired. Get a new access token from the A
        AlmostFull: out std_ulogic;                  -only last half free
        Empty: out std_ulogic);                      -FiFo empty
end FiFo;
```

As parameters, the addresses of the first entry will be first, of directly on the Last following last and the current number of free entries, free, used. In one single process the Input signals evaluated and the parameters updated, as well as the output data generated.

3.2 Counters and timers

The interface:

```
entity counter is
  port (nReset: in std_ulogic;)                      -system reset
        Clock: in std_ulogic;                        -system clock
        Address: in std_ulogic_vector(1_downto_0);   -address bus
        Source: in std_ulogic_vector(0_to_3);        -counter input
        Data: in std_ulogic_vector(26_downto_0);     -incomming data
        DataOut: out std_ulogic_vector(26_downto_0); -outgoing data
        Writes: in std_ulogic;                  -write operation (active high)
        Tint: out std_ulogic_vector(3_downto_0));    -interrupt request
end counter;
```

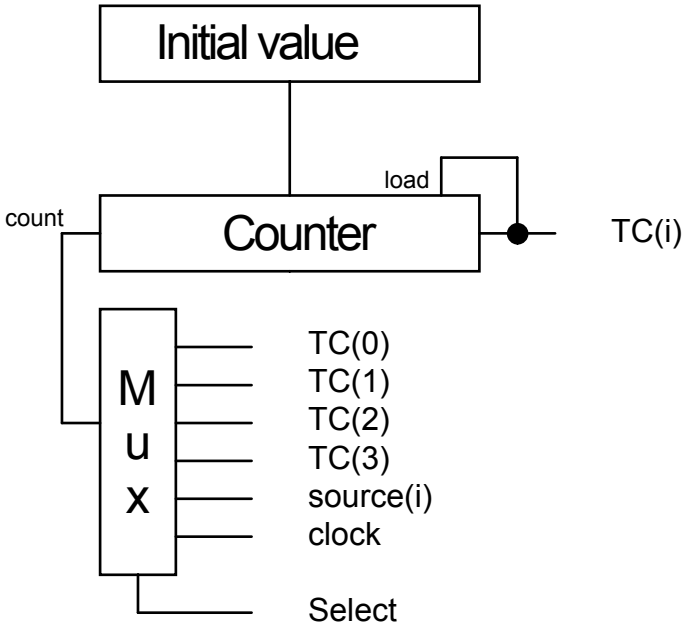


Figure 11: A single backward counter

The principal structure of one of the four implemented 24-bit counter is shown in the picture. It is a backward counter, the automatically with the initial value initialized as soon as he has counted to zero down. A pulse generated that this tc signal, by a system clock, duration, is also used as a break signal may continue it be an another counter counting event. Alternative counting events are the external Input signal source and the system clock clock.

The counters work in parallel. The counting events can be combined with maximum half System clock arrive, since the input count not on the level, but on the Change of the level of \"inactive\" to \"active\" responds. Is the counting event of the system clock clock, goes there twice as fast, since no change must be evaluated.

The counters are by writing a definition on their base addresses (ffffffc0H) define (ffffffc4H, ffffffc8H, or ffffffccH), the counter takes over the new definition, table 28, immediately, not suspends the current counter but. Only if 0 is reached, he will loaded new initial value in the numerator.

Bit	***Date***
2 - 0	Select, serves to the Selection the Counting event
26-3	The initial value

Table 28: Assignment of Definitionsport

The selection of the counting event is defined in table 29.

Select input		***Program for the production of Select***
000	TC(0), Counter 0 is the Prescaler	0 TRIGGER PRESCALER
001	TC(1), Counter 1 is the Prescaler	1 TRIGGER PRESCALER
010	TC(2), Counter 2 is the Prescaler	2 TRIGGER PRESCALER
011	TC(3), Counter 3 is the Prescaler	3 TRIGGER PRESCALER
100	The external Entrance *source*	TRIGGER INPUT
101	The system clock clock	TRIGGER SYSCLOCK
110	The system clock clock	TRIGGER SYSCLOCK
111	The meter is locked	LOCK COUNTER

Table 29: The meaning of the parameter select

The data in the following format, table 30 appear when reading the base address.

Bit	***Date***
2 - 0	Select
26-3	The current Counter

Table 30: Usage of Leseport

3.3 The interrupt controller

The interface:

3.3 The interrupt controller

```
entity IntVectors is
  generic (constant TableBitWidth: integer: = 4);
  port (nReset: in std_ulogic;)                          -system reset
        Clock: in std_ulogic;                            -system clock
        Quit: in std_ulogic;                             -interrupt quitted
        Token: in std_ulogic;                            -update pending
        IntSignal: in std_ulogic_vector(2_**_TableBitWidth_-_1_downto_0);
                                                         -input
        Writes: in std_ulogic;                           -write port
        Reads: in std_ulogic;                            -read port
        Address: in std_ulogic_vector(1_downto_0);       -port
        Data: in std_ulogic_vector(2_**_TableBitWidth_-_1_downto_0);
                                                         -incomming data
        DataOut: out std_ulogic_vector(2_**_TableBitWidth_-_1_downto_0);
                                                         -outgoing data
        Vector: out std_ulogic_vector(TableBitWidth_-_1_downto_0);
                                                         -vector number
        VectorValid: out std_ulogic);                    -vector valid
end IntVectors;
```

A processor should take into account also unforeseen external events can. He needed to a unit the such events realizes several concurrent events that selects with the highest urgency, and finally the processor core tells how he has to do with this event. Such Called interruption (interrupts), and the processing unit events *Interrupt controller.*

There are a number of species as the unit to respond to an interruption, the Most flexible is in the form of an interrupt vector. Thus, the controller sends a Address of the address of an interrupt handler is stored in the Processor core, this interrupts the current processing and leads instead Interrupt handler, then he continues the interrupted program.

This implementation is the number (0-15) of incompatibly signal to the *sends program prefetch queue that they multiplied by 4 and the product as* Address of the vector is interpreted. For vectors are therefore the first 64 bytes of Address space to allocate where the addresses of interrupt routines to store are. You can change therefore arbitrarily, making a maximum of Flexibility is guaranteed.

The synchronization with the core is done through the call signal VektorValid - a valid vector is available - which acknowledged with quit when the vector is accepted. Only the - the service routine is executed by the core, allows the signal token Controller to create a new vector.

This unit is 16 asynchronous, pulse or level-driven, inputs, which at Each clock be read - therefore a pulse duration of at least one clock. The

Input 0 has the highest priority, the entrance of 15 the lowest, higher quality Interruptions can conversely not interrupt Niederwertigere. Each input is maskierbar, can thus systematically be excluded from processing. The Inputs are not directly evaluated but accumulates, the battery is but evaluated in every cycle, this means it is the break with the highest Determines priority and, if allowed, their vector generates. The battery ensures that non-prioritized Interruptions get remain. Interrupt routines must her akkumuliertes signal by reading or writing to a reserved address, targeted reset. Preferably this is done immediately before the command EXIT, it is guarantees, that None low-order Interruptions the Interrupt handler can interrupt.

Further, the possibility exists on the controller or switch off, what the Programmer of interest may be, also provides the assembler macros, table 31, ready.

Macro	***Inbound outbound address***			***Comment***
EGG	0|1		ffffffd4H	parameter 1 is the controller with 4 Cycles Delay turned on, with 0 Right away switched off. The Macro provides no value back
DI		0|1	ffffffd4H	the Controller is unconditionally turned off,. are but his previously State back
QI			ffffffd8H	Receipt the interruption - sets the accumulated back - signal and allows Thus one new Evaluation of input
PI		pending	ffffffd8H	Is a 16-bit vector back, at which each set bit indicates whether be corresponding Entrance one Interrupt requests
SETIMASK	mask		ffffffd0H	The corresponding input each set bit is locked.
GETIMASK		Mask	ffffffd0H	Read the mask

Table 31: Macros for the integration of the controller in FORTH

3.4 The Rome

The interface:

```
entity ROMcode is
  port (clock: in std_ulogic;)                          -system clock
        Address: in std_ulogic_vector(ROMrange);        -address bus
        Data: out DataVec);                             -outgoing data
end ROMcode;
```

The capacity is defined by ROMrange, global.vhd, file. Currently, one has Capacity of 4096 words to 32 bit. Its size can be in 2048 byte increments enhanced be if file global.vhd, ROMrange' is placed high on new and IndexBitWidth is reduced, where appropriate, accordingly. The local memory of the stack has then a smaller capacity than 1024 words, so 2 stack with each half capacity. In Rome is the BIOS stored. His home address is situated directly to the RAM. The size the RAM is defined in RAMrange (file global.vhd). The BIOS itself is in the ABSCHnitt *Software in the first chapter explains.*

3.5 The CPU

The interface:

```
entity CPU is
  generic (constant TableBitWidth: integer: = 4;)     -ld (number of interrupts)
           constant IndexBitWidth: integer: = IndexBitWidth);
                                        -ld (stack size of dedicated memory)
  port (nReset: in std_ulogic;)                        -system reset
        SysClock: in std_ulogic;                       -system clock
        RxD2: in std_ulogic;                           -UART serial input
        TxD2: out std_ulogic;                          -UART serial output
        RxD1: in std_ulogic;                           -UART serial input
        TxD1: out std_ulogic;                          -UART serial output
        Source: in std_ulogic_vector(0_to_3);          -CTC input signal
        SD_A: out std_logic_vector(12_downto_0);       -memory address
        SD_BA: out std_logic_vector(1_downto_0);       -bank address
        SD_DQ: inout std_logic_vector(15_downto_0);    -databus
        SD_RAS: out std_logic;                         -RAS of DDR2 memory
        SD_CAS: out std_logic;                         -CAS of DDR2 memory
        SD_CK_N: out std_logic;
        SD_CK_P: out std_logic;
        SD_CKE: out std_logic;
        SD_ODT: out std_logic;
        SD_CS: out std_logic;
        SD_WE: out std_logic;                          -WE of DDR2 memory
        SD_LDM: out std_logic;
        SD_LOOP_IN: in std_logic;                      -SDRAM calibration loop
        SD_LOOP_OUT: out std_logic;
        SD_UDM: out std_logic;
        SD_LDQS_N: out std_logic;
        SD_LDQS_P: out std_logic;
        SD_UDQS_N: out std_logic;
        SD_UDQS_P: out std_logic;
        Interrupt: in std_ulogic_vector(0_to_7));  -external interrupt sources
end CPU;
```

The CPU implements the bus Manager. Here is the Adressdekodierung integrated Units in accordance with Table 32, the Interrupt signals be connected,. the Ports be ausschließlich the Stack administrations accessible made,. the *prefetch program queue* can ROM and RAM access only.
Stack management and program prefetch queue can at the same time on ROM and RAM access, i.e. a unit on the RAM, the other on the ROM. Both on
RAM or ROM is not possible, in such a case, the stack management takes precedence.

Address range	***Unit***
0 - 3ffffffH	RAM
4000000H - 4003fffH	Rome
fffffc0H - fffffcfH	Counter and timer
fffffd0H - fffffdfH	Interrupt controller
fffffe0H - ffffffbH	UART
ffffffcH - fffffffH	Software Interruption (Write access)
ffffffcH - fffffffH	System clock frequency in Hz (Read access)

Table 32: Adressaufteilung of the processor

3.5.1 Of the RAM controller

```
entity vhdl_syn_bl4 is
  port ()
    cntrl0_DDR2_DQ: inout std_logic_vector(15_downto_0);
    cntrl0_DDR2_A: out std_logic_vector(12_downto_0);
    cntrl0_DDR2_BA: out std_logic_vector(1_downto_0);
    cntrl0_DDR2_CK: out std_logic;
    cntrl0_DDR2_CK_N: out std_logic;
    cntrl0_DDR2_CKE: out std_logic;
    cntrl0_DDR2_CS_N: out std_logic;
    cntrl0_DDR2_RAS_N: out std_logic;
    cntrl0_DDR2_CAS_N: out std_logic;
    cntrl0_DDR2_WE_N: out std_logic;
    cntrl0_DDR2_ODT: out std_logic;
    cntrl0_DDR2_DM: out std_logic_vector(1_downto_0);
    cntrl0_rst_dqs_div_in: in std_logic;
    cntrl0_rst_dqs_div_out: out std_logic;
    SYS_CLK: in std_logic;
    reset_in_n: in std_logic;
    cntrl0_burst_done: in std_logic;
    cntrl0_init_val: out std_logic;
```

```
    cntrl0_ar_done: out std_logic;
    cntrl0_user_data_valid: out std_logic;
    cntrl0_auto_ref_req: out std_logic;
    cntrl0_user_cmd_ack: out std_logic;
    cntrl0_user_command_register: in std_logic_vector(3_downto_0);
    cntrl0_clk_tb: out std_logic;
    cntrl0_clk90_tb: out std_logic;
    cntrl0_sys_rst_tb: out std_logic;
    cntrl0_sys_rst90_tb: out std_logic;
    cntrl0_sys_rst180_tb: out std_logic;
    cntrl0_user_output_data: out std_logic_vector(31_downto_0);
    cntrl0_user_input_data: in std_logic_vector(31_downto_0);
    cntrl0_user_data_mask: in std_logic_vector(3_downto_0);
    cntrl0_user_input_address: in std_logic_vector(25_downto_0);
    cntrl0_DDR2_DQS: inout std_logic_vector(1_downto_0);
    cntrl0_DDR2_DQS_N: inout std_logic_vector(1_downto_0)
    );
end vhdl_syn_bl4;
```

The controller for DDR2 RAM, matrix, as 16x32MB was organized by the Xilinx Spartan3A-. Starter Kit [16] adopted and adapted. The data can only in burst mode (length four) (16-Bit words) read or write. The adaptation supports only single 16, or 8-bit access, but no direct 32-bit access. Reason is a faulty reading or letter - high and low Word are occasionally swapped depending on of the Adresslage-, the cause of which I could not locate. The made Adaptations are described in the annex, Chapter 2.

3.5.2 The StateMachine of RAM

The controller is controlled by the StateMachine and in the following diagrams represented.

3.5.2 The StateMachine of RAM

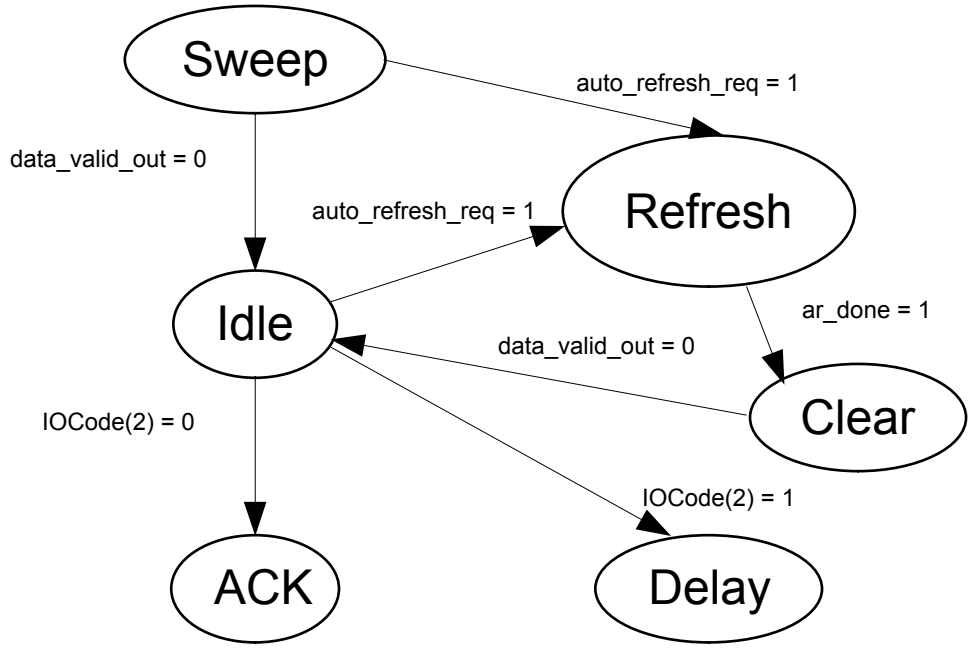


Figure 12: starting node idle and refresh cycle

From the idle state, it is with IOCode(2) = 0 in the State of ACK, reading the RAM, with *IOCode(2) = 1 in the State delay, write to the RAM, is active in the auto_refresh_req* State refresh. Refresh takes precedence over IOCode(2). IOCode(2) is only evaluated, if IOCode(1) \/ 0 and IOCode(0) = \/ = 0 are. Includes State sweep Read \/ write off. First when data_valid_out = 0 is a new can Memory access be initiated, is active before auto_refresh_req, it goes directly in the refresh State. Refresh will only leave if the controller by its end *ar_done 1 = signaled. In the State of clear is on the readiness of the controller* maintained and changed = 0 after idle with data_valid_out.

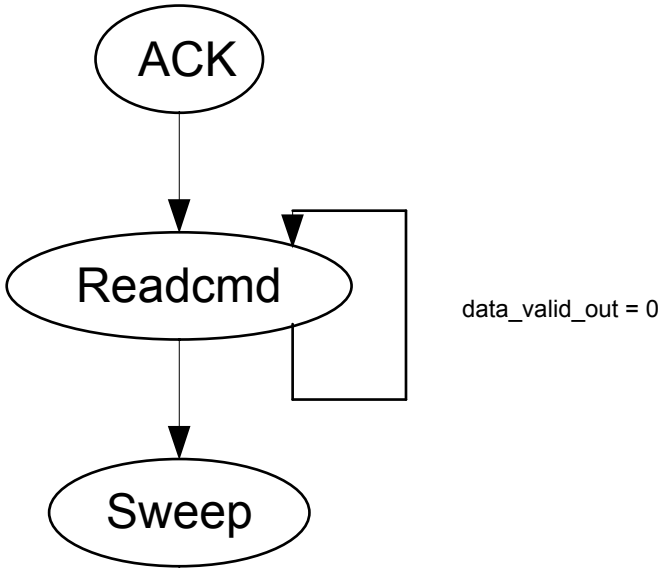


Figure 13: The read cycle

From the ACK State it goes directly in the State of Readcmd, who will leave only
If data_valid_out = 1 is, only then will the data from user_output_data
taken over and stall the signal = 0. This shows the stack management or the
program prefetch queue, that the data in the next clock is taken
can. The end of access will wait in sweep.

3.5.2 The StateMachine of RAM

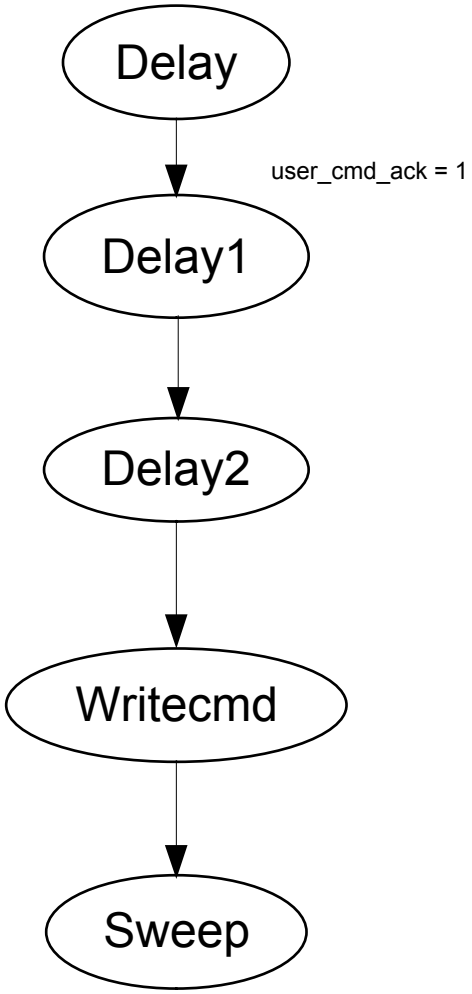


Figure 14: The write cycle

As soon as user_cmd_ack = 1 cycle so initiated, it enters the State
Delay1, and it immediately in the State Delay2, where user_data_mask with loud
Filled ones, to prevent a letter of the last 2 words of burst access.
In the State of Writecmd, the acknowledgement signal is stable = 0 set and thus the
Stack management end of the operation signals. In sweep is the end of
Wait for access.

The StateMachine has been simplified, only for 16 or 8-bit access wiederg ege en. implementation regulated was the 32-bit access, the by two consecutive 16-bit access
was realized. For store operations additional operation steps arise, how
recorded in table 33.

Unit	***32-bit To write***	***8 or 16-bit To write***	***32-bit read***	***8 or 16-bit read***
RAM	14	7	14	7
ROME	-	-	1	1
Integrated Units	0	0	0	0

Table 33: Number of additional bars for accesses to RAM or ROM

4 Test and verification

Source	***Comments***
Countertest.vhd	Testbench for the counter module
Inttest.vhd	Testbench for the interrupt controller
Programcountertest.vhd	Testbench for the program of prefetch queue
Uarttest.vhd	Testbench for the UART
cpu1sim.vhd	Testbench for the processor (verification)
Cpu1.vhd	The processor with SRAM interface
TheCore1.vhd	The processor core
SRAM.vhd	A static RAM
UART1.vhd	A second UART
cpu2.vhd	Testbench for the Processor (Post simulation)

Table 34: The source files for the Verikation and test

Indicated programs were created to verify or test, the in
own ISE project (FORTHS) are. This version of the CPU used an SRAM,
which easier to simulate lets than a DDR2 RAM.

4.1 Test of individual components

4.1.1 The counter module

The counter 0 is initialized to 1 and is the system clock - timer - timer

operated. As soon as the associated break signal is active, the test run is ends. The signal must be exactly one measure.

4.1.2 Of the interrupt controller

Break input line 4 becomes active first, after 200 ns line 5 on *Vector should be 4 be with active VectorValid. After further 200 ns* both lines down and quit, as well as token to acknowledge the suspension 4. Now should *Vectorvalid be inactive. After another 100 ns quit and token are inactive and management* 3 active. The vector should be issued and VectorValid be active 3.

4.1.3 The program prefetch queue

Checks will CALL, TRAP and EXIT behavior. The correct function can to the waveforms of address, opcode, and immediate in the WAVE window by ModelSim read be. The program is in *myromtable* the Finding testbench.

4.1.4 Of the UART

Transmitter and receiver are shorted. Is checked, whether individual words (all 0) louder 1, 1 0 (alternating) including Parity correct received on be. At a message is displayed faulty reception.

4.1.5 The stack management

A testbench was used for the stack management also, but no longer exists. She was built, but just as easily, as the remaining benches. The correct function could be detected with the testbenches anyway.

4.2 Test of the processor

4.2.1 Behavioral simulation of code

An assembler program that object in the variable is specified in cpu1sim.vhd is sent to the server (processor and BIOS), there bound and executed. The all communication between server and Simulator is as a dump on the console display of the Simulator. The purpose of this testbench is the verification of the processor, the BIOS and the assembler to allow. Some files had this modified be.

- cpu1.vhd Contains a SRAM interface and one additional Output Mnemnonic to display the current command in the command decoder.
- theCore1.vhd Provides in addition to Mnemnonic.
- SRAM.vhd Contains a 32x64k static RAM. Dumps are possible in the man in the highest address of RAM the starting address of Dump writes.

- UART1.vhd An additional UART to communicate with the server.

The UART will work with 4.5 Mbaud. In bios.fs, the macro SIMULATION to-1 is to do this setZen. Mlt This testbench has been successfully verified the BIOS. Errors in the processor not occurred, would have been - recognized but he had given the wrong results, what would be noticed. Has been tested with the following assembler programs.

- countt.FS Check and test the counter module
- juchu.txt Test the Consolenausgabe
- memorytest.FS Test of dynamic memory
- mirror.fs Flipping the bits of the word f0f0f0f0
- shift.FS Test the Schiebeoperationen
- test.FS Test the operation \/ MOD
- testf.FS Test the math library

4.2.2 Post map simulation

cpu2.vhd works cpu1sim.vhd analog, it serves only the simulation on RTL Level.

- mycpu.vhd Contains an SRAM interface and the processor.
- SRAM.vhd Contains a 32x64k static RAM. Dumps are possible in the man in the highest address of RAM the starting address of Dump writes.
- UART1.vhd An additional UART to communicate with the server.

This test was a formality. All waveforms were smooth, without peaks and spikes, a review of the sources is therefore unnecessary.

4.2.3 Post place and route simulation

This was also performed with cpu2.vhd and was the final test of Processor, before it has been tested on the Development Board. A SImulationslauf is very time consuming and with program juchu.obj, lasted about 4 days, ran but immediately successful.

Four prints of the SRAM memory access are listed below. In the first -2 as a 16-bit word is written, read it in the next. The third is-2 as a 8 - bit Written word, and in the last read.

4.2.3 Post place and route simulation

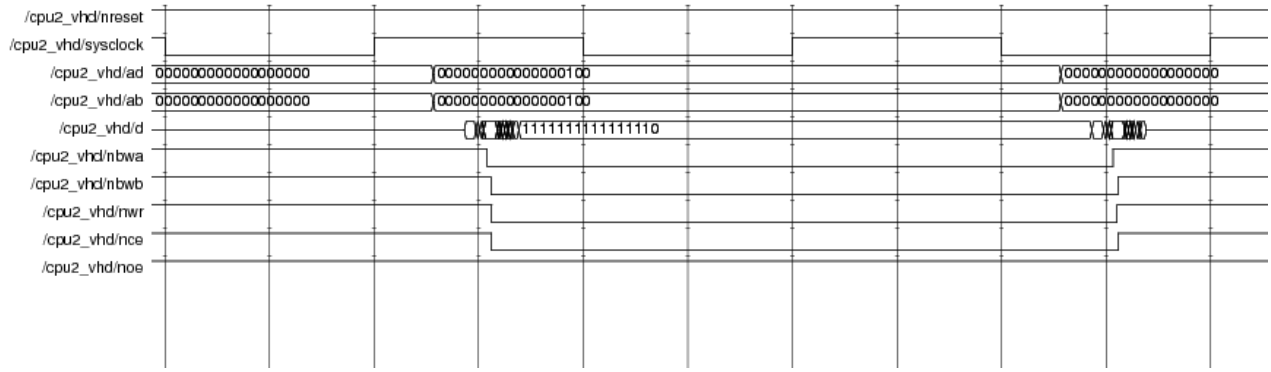


Figure 15: 16-bit write access to the SRAM

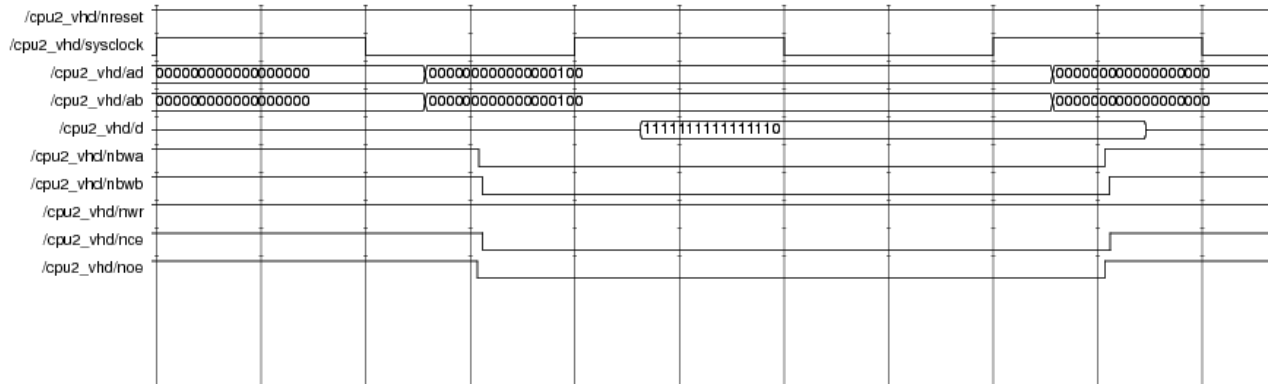


Figure 16: 16-bit read access to the SRAM

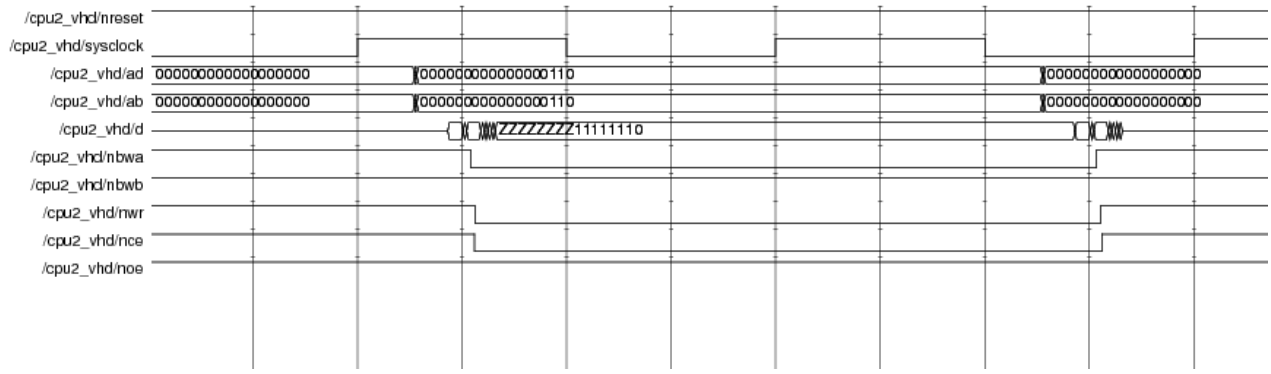


Figure 17: 8-bit write access to the SRAM

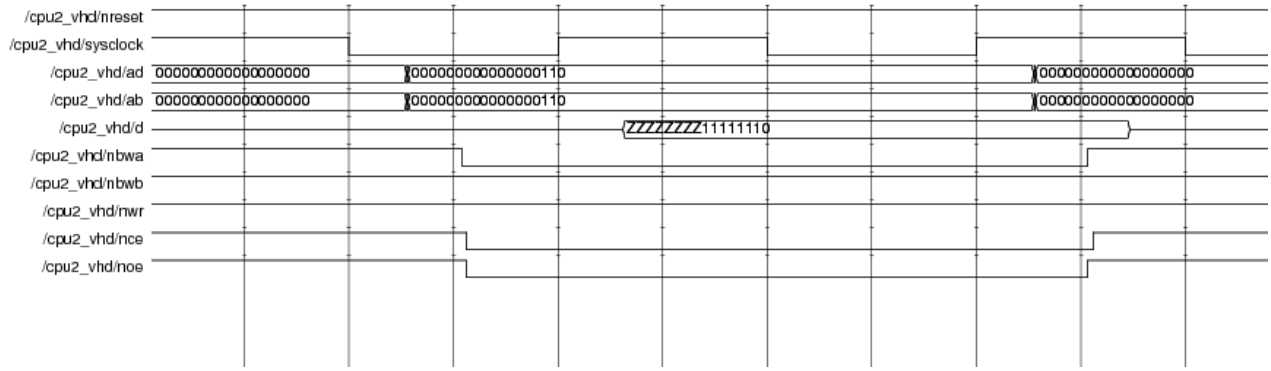


Figure 18: 8-bit read access to the SRAM

Software

Are used in EBNF grammars – the notation is known as
-provided, as well as for their Nonterminalsymbole regex expressions - be read in *JavaDoc from Sun, class pattern [20].*

1 The BIOS

It includes several source files, table 35.

Source	***Comment***
BIOS.FS	Initializes and starts the server
Dictionary.FS	Interface to the FORTH dictionary
memorymanagement.fs	dynamic memory management
SIO.fs	Service routines of the serial interface
counter.FS	Support for counter and timer
interruptservice.FS	Support for the Interruption management
consoleIn.fs	The standard input
consoleOut.fs	The standard output
fileIO.fs	The file management
Java.txt	Support for Java
Math.FS	Integer arithmetic nd Logik
fmath.FS	Floating-point numbers
FormatNumber.FS	Formatting integer values
linker.FS	Loading, bind and execute by Applications

Table 35: The source files of BIOS

The memory is, split as defined in table 36.

Beginning	***End***	***Comment***
0000000	000003F	Table the Interrupt routines
0000040	000083F	Read or Read buffer the Console
0000840	000093F	PAD, a predefined Scratchbuffer
0000940	0000A3F	16 Bufferhandle
0000A40	2FFFFFF	heap
2FFFFFF	0000A40	Dictionary
3000000	3FFFFFF	applications
3FFF000		Stack of return addresses
4000000		Data stack

Table 36: The allocation of RAM

After a reset the dictionary is built on the stack and its overflow area set the table of the interrupt routines, according to table 37, initialized, and the Client about the readiness of the Server - service 2 52 – informiert.

Interruption routine	
15	SERIALOUT - the sender of UART
14	SERIALIN - the recipient of UART
13	TIMER3 – on count the System time
0	ILLEGAL – undefined Command code
1-12	masked

Table 37: Used interruptions

Then the server constantly checks whether a service is requested and leads him where appropriate, out. The implemented services are shown in table 38.

Code	***Service***
255	Software reset - causes a restart of the server
254	Loading, bind and execute an application
253	Completely read the dictionary and send it to the client
252	The length of the free standard input to the receive buffer Send client
251	Immediate termination of an application
250	Redefine the serial interface
249	Querying the size of heap and dynamic storage
248	The dynamic memory create and initialize
247	The dynamic memory freed
246	Re-initialize the dynamic memory
245	Remove a module
244	Late binding of last loaded modules
243	Loading, bind and execute multiple modules of an application
242-241	undefined

Table 38: The services of BIOS

Newly defined services require a modification of bios.fs and the service routine *SERIALIN in SIO.fs.*

Following default variable [2] are defined by the BIOS: PAD, HLD, TIB, #TIB, > IN,. SOURCE ID, SPAN, BASE, PRECISION, SYSCLOCK, MILLISECONDS. Their importance is, with the exception of SYSCLOCK, the system clock frequency in Hz, and MILLISECONDS, the time that has elapsed, to read in the standard [2] since the last hardware reset and will not discussed here. The standard features are listed below and also not explained, with the exception of extensions.

1.1 The Protocol

The 8-bit ASCII code is used. The character ranges are 0-27 and 241-255 reserved control characters, to be used it as data, rollout with a the character 27 (escape) must be masked.

The construction of a set is set out in table 39.

Byte	***Importance***
0	A handle value range 0-22 0 ... Standard input or output 1-14... File handle 15... reserved for the log 16. for \"free receive buffer\" standard input 17-22 reserved 23... reserved for XON 24... reserved for XOFF
1	Specifies the action
From 2	Data
last	Sentence: 26 for correct data, 25 for feh lerhafte Data

Table 39: The layout of a frame that is used by the services

The services operate according to following protocols of table 40. It is left to right to read where each new line represents a step forward in the process. A Service can be initiated only by the client by sending the number of the service, the server responds with a result or asks the client to send data,
etc.

1.1 The Protocol

Client	***Server***
255	16 freeinputbufferlength 26
254 or 243 15 deck 26 - send of Deck	15 READ 240-1 26 - read request 15 MODULPROCESSED 4-byte-errorcount (26 | 25)-end the (successful = 26) communicated binding
253	Send 15 LISTDICT dictionary 26 - dictionary
252	16 freeinputbufferlength 26
251	

Client	***Server***
250 15 7-Byte 26	15 READBUFFERED 240 7 26 - read request the Parameter
249	15 MEMORYUNUSED freeheapsize dynamicmemorysize freeprogramstorage 26
248 15 4-Byte 26	15 READBUFFERED 240 4 26 - request size
247	--
246	--
245 15 Module name 26	15 READBUFFERED 240-1 26 - read request of NAmen
244 15 List 2 6	15 READBUFFERED 240-1 26 - request the list of to bind references

Table 40: Protocols of the services

1.2 File operations

The file system [2, section 11.6.1, page 66-70] allows the server data from the client to read or to sch on it ReBen. As a whole can at the same time 16 files used be the of handle are managed. Reserved are 0, the standard input handle or output, and handle 15 for the Protocol of the communication between server and Client that handle 1 to 14 are freely available, but from the server when you open or Creating a file automatically, and can user does not affect be. A handle is represented by a single byte, defined in table 41.

1.2 File operations

Bit	***Importance***
7-4	File or Pufferhandle
3	Flag directory
2	File writable
1	File readable
0	Binary data

Table 41: Meaning of bits of filehandles

The attributes one handle can with you, am, W\O RIO, R\W are defined, handle be awarded by OPEN-FILE, CREATE FILE and shared with CLOSE.

A Pufferhandle defined in tab elle 42, is a Quadtupel of words with may refer to Importance.

Word	***meaning***
0	Address of the ring buffer
1	Length of the ring buffer
2	Starting index of the logical buffer
3	Current Length the logical Buffer

Table 42: The layout of a Pufferhandle

The remaining commands specified in fileIO.fs are: DELETE FILE, FILE heading; FILE-SIZE, INCLUDE FILE, INCLUDED, READ FILE, READ-LINE, REPOSITION FILE, RESIZE FILE, SET FILE DATE, WRITE FILE, WRITE-LINE, FILE STATUS, GET FILE DATE, FLUSH FILE, RENAME FILE, *PAGE, AT-XY.* In addition defined Operations are shown in table 43.

Command	***Importance***
ABSOLUTE FILE	buffer bufferlength name namelength - buffer bufferlength success(=0) Determine the absolute file name in the file system of the client. Returned is a string
ENTRIES FILE	buffer bufferlength name namelength - buffer bufferlength success(=0) Read the entries of a directory. Returned is a Vector of character strings
CHANGECHARSET flag-	Sets the character set of the output console on the client with true on ASCII, otherwise to Unicode. Start default is ASCII.

Table 43: Additional operations on files

A code is generally associated with any command that needs the help of the client, He specified the service of client. About 15 handle records of following structure are passed.

Action [handle] {parameter} 26

Action ← OPEN | CREATEF | READ | LINE | LISTDICT | ATXY |
GETFILEPOS | SET FILE POS | GETFILESIZE | SET FILE SIZE |
INCLUDED | CLEAR | STATUS | RENAME | DELETE | CLOSE |
ABSOLUTE | RELATIVE | SET DATE | GETDATE | READBUFFERED |
ASCII | *ERRORREC*

The parameters are the ones expected for the command, and end with code 26. Then, the server waits for a response to the client. The client responds to Execution the Remote procedure overnight Handle 15. Exception are the *italic* written commands that need no acknowledgement.

{result} (26 | 25)

1.3 The dictionary

A single entry is in FORTH token [2, section 3.3, page 13], it is defined by a triple by words:

- Name the address of a FORTH string one 2 byte length followed by the string. This name can Token be referenced.
- Type the type of the token
- Data field an a constant value or the address of a function or

1.3 The dictionary

Variable

FolgeNDE types are shown in table 44, possible:

Value	***Importance***
1	CONSTANT - a constant value
2	VARIABLE - the address of a variable
3	PROCEDURE - the address of a function (not standard)
4	VALUE - a value of a
5	MODULE - base address of an application (no default)
6	MARKER - saves the current starting address of the heap free for subsequent recovery
7	2 VARIABLE - the address of a double word
otherwise	undefined

Table 44: data types of FORTH

The following commands are in the source file dictionary.fs, standard commands are: FIND, EXECUTE. Additional macros, table 45, are defined in the Assembly.

Befe HL	***Importance***
create	datafield type adr- The parameters define a token that is to enter. This function should not directly, but called only through the macro CREATE, be.
forget	drobnicy- Parameter is the name of the entry to be deleted, in the form of a FORTH-String. This function should not directly but only through the macro FORGET, are called.

Table 45: Macros for the dictionary

The dictionary is an unsorted list of consecutive merely, at one end entries be appended. To delete entries are not necessary immediately way deleted but marked as undefined. Should all entries on the current to delete follow, be undefined, the list will be shortened.

A token is executed modules of the type, the list shortens immediately, that is to say, all the module following entries are deleted immediately.

The BIOS allocates a shared memory block for the heap and the dictionary.
The heap starts at the low-order block address and direction grows higher
Block address, the dictionary starts at the higher value block address and grows
Towards lower-order block address, which a best possible exploitation of
allocated block is possible.

1.4 Loading, binding and initialize applications

After receiving an object file, uploading, these must be made executable.
This is done in three steps. First is the initial part of declarations, usually by
Variables and registration of the new commands in the dictionary by dissolving the
References runs made and executed. The references are then the
resolved resident part of the deck, and released the initial part and reference list.

Grammar of a deck:

← length resident length initial flag deck references

flag... only execute 0, -1 execute and keep resident

length... the length of the following block, an integer

resident ← modulename [code]

initial ← code

moduleName ← forthstring

Code ← (any | unresolved) [code]

unresolved ← local | Dictionary

local ← null

dictionary ← stack zero zero

zero ← zero zero zero zero

zero ← (byte) 0

stack ← zero | (byte) 0 x 80

any... any B ytewe RT

Re ← differences ← {record}

record ← addr [forthstring align length] {offset}

offset ← positive | negative

addr... forthstring the relative address of the identifier in the deck or - 1, if
He referred to a Dictionaryeintrag

forthstring... the identifier - 2 byte length followed by the string

align... Padding guarantee a word boundary as position by length

positive... the relative address of an unresolved in the deck, case local exists

negative... the relative address of an unresolved in the deck, case dictionary exists

Is not 1, addr is definitely local and by the absolute address of referenced identifier to override. Exists case negative for offset - only followed by a name (forthstring) addr - dictionary to according to table 46 override.

Type	***Replacement***
CONSTANT	(stack |) (VAL), data field
VARIABLE	(st)aCK | (VAL), data field
PROCEDURE	(stack |) (TRAP), address of the data field
VALUE	(stack |) (VAL), address of the data field (stack |) (VAL)
MODULES	(stack |) (VAL), address of the data field
MARKER	(stack |) (VAL), address of the data field
2 VARIABLE	(stack |) (VAL), data field

Table 46: Substitutions for each data type

(\"|\" stack is a concatenation operator, (0 or 1) the stack indicates for VAL or TRAP) (the corresponding command code to use.)

In the case of positive is to override local by the address of the data field.

For a self-contained application, so a single deck, this strategy is sufficient and is initiated by the code 254. An application consists of a Series by Decks, Code 243, can individual References may not to be resolved immediately. These references are together with the start address, the deck in the store to the client with an ID of ERRORREC sent back and there in a L iSt e gArgumentException: The incoming token has expired. Get a new access token from the subsequent binding of this list is initiated by sending code 244 and attempts to resolve the outstanding references.

1.5 Memory management with reference counting

Standard are: UNUSED, modified standards are ALLOCATE, FREE, resize [2,] [Section 14.6.1, page 87]. The modification is that the memory management the User provides a block of memory not directly, but via a Handle. Establishing one handle is defined in table 47.

Word	***meaning***
0	The reference counter
1	Starting address of the allocated memory the data area block

Table 47: Establishing a memory handle

This solution allows a memory compaction only! Should an allocation request not immediately satisfied are can, the compression is automatically called. This compresses the occupied blocks by moving at the end of dynamic Store. So a single large free formed by the beginning of the dynamic memory Block that can probably meet the requirements.

Java expects a memory management, which automatically detects when an occupied block can be actually released. This is a very old method that supports it, Counting the references [8, Chapter 2.1, page 19-25], a task by the Application must - be done the memory management but must support. In short, a block is only in the list of free blocks transferred when be Reference count reaches zero, otherwise the counter is simply humiliated. Advantage of this method is the distribution of administrative work, it is only If it is really necessary. Other procedures require a own periodic garbage collection process which will not be interrupted may what is not desirable.

Reference counting also has a disadvantage. A circle in a graph can not self-lubricating constantly detected and resolved. The application programmer needed at Use of graphs in the concerned classes per a method dispose, the the Pointer of the nodes of the graph is set to null, which guarantees any circles to be resolved. Nevertheless I consider with this method for the best beneficial a minimum of implementation effort, and most economically in terms of Storage - a block is no longer needed, it is released and is immediately available.

The dynamic memory is a closed block of memory via the services, can be created, shared, and initializes. It is only a dynamic storage possible. In the first word of the block is the number of the created handle in the second word is the head of the list of free blocks. From the third word, it is consecutive list handle of which can grow any. The rest of the block disintegrates in documented, non-contiguous areas that handle that are accessible. The blank areas are organized in a doubly linked list. Has a free block following structure, table 48.

Word	***use***
0	Address of the next free block
1	Length of blo Ck-es
2	AdUnitholdens the previous free BloCk-es
AB 3	unused

Table 48: Establishing a free block of memory

An occupied block is defined in table 49.

Word	***use***
0	Address the Handle
1	Length the Block
From 2	Data region

Table 49: The layout of a busy block of memory

1.5.1 Allocation

First is a free handle searched or created and a best matching free Block searched. None exists is searched a garbage collection again, aborts on failure. Otherwise, the block is from chained, adapted, initialized, the List the free Blocks modified and the Address the Handle returned.

1.5.2 Free

The counter will be humbled, and only if it is 0, the list blocks free, to possible merger with neighbouring countries, registered. The handle is not released, can be reused but.

1.5.3 Resize

The data area of the block is copied to the heap, and go-ahead given for the block. A new block of the desired size is allocated, the data field of the current Copied blocks from the heap, and the old handle, which now contains the new data range, returned.

Additional commands are recorded in table 50.

Command	*Importance*
MALLOC	Handle size - flag This function is identical to ALLOCATE, but specifically for Java adapted
ALLOCATED-SIZE	handle - size Returns the size of the data area, without administrative data,
HANDLEVALID	handle - flag If it is a valid address, - 1, otherwise 0 is returned
INCREFERENCE	handle- increments the reference counter

Command	*Importance*
DECREFERENCE	handle- humiliated reference counter. If 0 is reached, the block is released

Table 50: Additional functions for memory management

1.5.4 Times for INCREFERENCE, MALLOC and DECREFERENCE

The following abbreviations apply:

- N the amount of dynamic memory in words
- h the number of applied handle
- m the number of occupied blocks
- b the memory of all allocated blocks
- f the number of free blocks

Also applies the Anna hme, daSSebeen processed in command at intervals.

The constants represent the counted out number of the commands of a sequence, the The number of iterations of a sequence represent variables.

The number of commands for the hidden procedure for memory compaction:

COMPACT POOL 56 + 45 = born f + 11 b. b + 15 born m

The number of commands for the hidden function to the Handlesuche:

$GETFREEHANDLE_{usually}$ = 88 + 8 b. h

$GETFREEHANDLE_{worst}$ = 88 + 8 born h + COMPACT POOL

1.5.4 Times for INCREFERENCE, MALLOC and DECREFERENCE

The number of commands for the hidden function to determine of the last best free Blocks:

SEARCHLAST = 13 + 23 born f

The number of commands for the hidden function to determine of the best free block:

$BESTFREEBLOCK_{usually}$ = 53 + SEARCHLAST

$BESTFREEBLOCK_{worst}$ = 53 + SEARCHLAST + COMPACT POOL + SEARCHLAST

The number of commands for the function MALLOC:

MALLOC (n) = 70 + GETFREEHANDLE + BESTFREEBLOCK + (n + 3) \/ 4 born 11

The number of commands to the function DECREFERENCE:

$DECREFERENCE_{decrement}$ = 48

$DECREFERENCE_{destroy}$ = 48 + 188 + 32 born f

The number of commands for the procedure INCREFERENCE:

INCREFERENCE = 38

1.5.5 Final remark

The implemented storage allocation is based on MS-DOS. With regard to the Storage utilization is optimized the implemented strategy! She can however be easily be replaced by a different memory allocation. The implementation starts with MOVEBLOCKUP including and ends at UNUSED only and takes 1193
A byte of the BIOS. For a different implementation are just under 2 kB in Rome, with a Capacity of 4096 words, available. Will need more space, is the capacity, as in Chapter 3.4 described to expand...

1.6 Standard input and standard output

In consoleIn.fs following commands are: KEY?, KEY WORD, PARSE, restore INPUT, SAVE INPUT, QUERY, REFILL, EXPECT, OR ACCEPT.

In consoleOut.fs following commands are: EMIT?, EMIT, TYPE, SPACES, SPACE, CR. One Extension is recorded in table 51.

Command meaning	
PRINT addr length-	The sequence of bytes, starting at addr of length length, consists of characters in the Unicode, and appears on the console.

Table 51: Additional functions for the console

1.7 Serial interface

The local service routines SERIALIN and SERIALOUT serve the interruptions

the UART. The send routine removes the send buffer only the next character if one exists and transmits it over the UART. The reception routine detects that Request services and tax data evaluates all other data are unmasked in the ring buffer in the current Pufferhandle - default is 0, the Consoleneingabe. The integration of UART is documented in table 52.

Command	***Importance***
SETSIO	Stop bits parity wordlength baudrate- defines the mode of UART, where: baudrate the baud rate wordlength 7 or 8, the word length, in bits parity 0 for no parity bit, 1 for odd parity 2 for even parity Stop bits 4 for 2 stop bits, 3 for 1.5 stop bit, 2 for 1 stop bit
GETSIO	-Stop bits parity wordlength baudrate Returns the current setting of the UART back
REDEFINESIO	stop bits parity wordlength baudrate- Reinitializing the interface

Table 52: Additional functions for the integration of the UART in FORTH

1.8 Meter and timer

The procedures for the integration of the counter is documented in table 52.

1.8 Meter and timer

Command	***Importance***
TRIGGER SYSCLOCK	-fashion The counter is to divide the system clock
TRIGGER PRESCALER	prescaler - fashion The counter uses the counter prescaler as prescaler
TRIGGER INPUT	-fashion The timer used for external counting input
LOCK COUNTER	-fashion The counter should be locked

Command	***Importance***
SET COUNTER	counter mode reloadvalue- Starts the counter mode and the counter mode Reloadvalue initial value
READCOUNTER	counter - count Reads the current counter counter counter
SET-COUNTER-SERVICE	serviceroutine counter- Counter assigns a service routine to the counter and en-masked his break signal

Table 53: Additional functions for the integration of the counter module in FORTH

1.9 Interruptions

Provides the standard commands ABORT and QUIT, and in addition as shown in table 54.

Command	*Importance*
SETVECTOR	serviceroutine number- Associates the service routine interrupt number to
GETVECTOR	number - serviceroutine Are the service routine interrupt number back
SOFTINT	-- forces a software interrupt (1 service routine).

Table 54: Additional functions for integration of interrupt controller in FORTH

1.10 Java

Some routines that every Java application needs are given in table 55.

Command	*Importance*
INSTANCEOF	object forthstring - castedobj Should an object in his by forthstring designated super class be cast and if successful the Super object you want return, otherwise 0.

Command	*Importance*
CASTTO	object forthstring - castedobj Should an object in his, by forthstring known, super class or derived Klapasses gewandelt. On success, the desired Object returned, otherwise 0.
EXECUTE-NEW	table hash code - obj The constructor of the method table identified by hash code one Class should called be and one new Instance return. The table is a list of tuples of the form (hash code, routine).
EXECUTE METHOD	object hashcode polymorphically - result of method The method identified by the hash code of the object or its Super objects to run if desired, what most of the Case, should be taken into account the polymorphism.

Table 55: Additional functions to support Java

For the top 30 bit of hash code, there are no requirements, what they contain and how they are determined, thing of the compiler is. For the two lowest bit is
Set default table 56.

Bit	***Default***
0	1 for PRIVATE, otherwise 0
1	1 for PROTECTED, else 0

Table 56: Predefined bit of hash codes for Java

1.11 Mathematics

1.11.1 Simply integer

*, V MOD, V, MOD, born V MOD, born V [2] process simply integral operands, and deliver a fach ganzzahlige results, M born delivers a double integer result.

1.11.2 Double integer

>A;character string in a double integral convert CROAKER, CONVERT Value.

TO V MOD,. SMVREM, FMVMOD deliver simple integer Results. Additional Functions are documented in table 57.

Command	*Importance*
UDVDMOD	uddividend uddivisor - udremainder udquotient divides unsigned double integer operand
UDVMOD	uddividend udivisor - uremainder udquotient by dividing an unsigned double integer operand a simple integer operand
ASHIFT	d shiftfactor - d a double integer arithmetic to the right moves, the Schiebewert is signed
SHIFTL	UD shiftfactor - ud a double integer value is moved to the left, the Schiebewert is signed
SHIFTR	UD shiftfactor - ud a double integer value is shifted to the right, the Schiebewert is signed
D BORN.	D1-d2 - d3 Multiplies two double integral operands
D V.	ddividend ddivisor - dquotient divides two double integral operands
DMOD	ddividend ddivisor - dremainder modulo two double integer operands

Table 57: Additional arithmetish logical functions

1.11.3 Formatting and output

< #, HOLD, SIGN, #, #S, # >, U.R, U.,.R,., D., D.R are standard features. Additional Functions are documented in table 58.

1.11.3 Formatting and output

Command meaning	
..R	width value - addr length such as.R, only the length and the address of the converted presentation is here returned
U..R	uvalue width - addr length like U.R, only the length and the address of the converted presentation is here returned
D..R	dvalue width - addr length such as D.R, only the length and the address of the converted presentation is here returned
UD...R	udvalue width - addr length like U.R, only the length and the address of the converted presentation is here returned

Table 58: Additional functions for formatting numbers

1.11.4 Double precision floating-point functions

The standard ANSI 754 [12] is used as the format. The four basic computer operation deliver results with an error of not more than half bit. Break all higher functions at the latest After 50 Iterations off,. or earlier,. If more Iterations None Change in the result can bring.

The basic operations F born, F \/ square algorithms, implement alternative comes for the multiplication algorithm of schönhage streets [14, theorem 7.8, page] [273] in question. The best method is to select. It applies:

n, the size of the problem: n = 53, the length of the mantissa bits

a, the number of 32-bit words for the mantissa: a = ceil(n\/32) = 2

Q (n), the complexity of the quadratic algorithm: Q (n) = 3 born a born n

S (n), the complexity of schönhage streets: S (n) = O (n born ld (n) b. ld(ld(n))).)

Q(53) is < S(53), then is the square process better

3 born a born 53 < 53 born ld(53) born ld(ld(53))

318 < 53 born 5,72792 born 2,518

318 < 764,41738

Thus the square algorithms in a 53-bit mantissa length are more efficient as schönhage streets, the only starting a mantissa length > 8 born 32 = 256 bit be better can.

The following, listed in the standard functions, can be found in the source file fmath.fs: F ***,

F, F +, f, FNEGATE, FMAX, FMIN, FLOOR, FROUND, F b., FACOS [= π \/ 2 - FASIN(x)],] FACOSH [13, Chapter 4.6.21, page 87], FASIN [= FATAN (x \/ FSQRT(1_–_x²)], FASINH [13, Chapter 4.6.20, page 87], FALOG, FATAN [13, Chapter 4.4.42, page 81], FATAN2,. FATANH [13, Chapter 4.6.22, page 87], FCOS [13, Chapter 4.3.66, page 74], FCOSH [13] [Chapter 4.5.2, page 83] FEXP [13, Chapter 4.2.1, page 69], FEXPM1, FLN [13, chapter] [4.1.25, 68], FLNP1, flew, FSIN [13, Chapter 4.3.65, page 74], FSINH [13, Chapter 4.5.1,] [Page 83], FSINCOS, FSQRT [18, Cape 1.3, page 11-15], FTAN [13, Chapter 4.3.67, page] [75], FTANH, F ~, F. FS., FE, > FLOAT, D > F, F > additional functions are in table D.
59 documents.

Command	***Importance***
RECIPROCAL	df - df calculates the inverse (1\/x) [18, Chapter 1.2, page 2-10] after Newton's Method and uses only addition, subtraction, and multiplication. No Division
F..	DF - addr length like F., only the length and the address of the converted is here Representation returned
FS...	DF - addr length like FS., only the length and the address of the converted is here Representation returned
FE...	DF - addr length like FE., only the length and the address of the converted is here Representation returned

Table 59: Additional functions for the treatment of floating-point numbers

1.11.5 An improvement of the square division algorithm

It is assumed that an n-bit addition or subtraction in one step
can be performed. Dan n iSt für the multiplication algorithm for the number of Operations of ≤ 3 n, the division algorithm for unsigned integers the number the operations ≤ 4 n + ld (n), which is significantly worse. It is an improvement but possible to ≤ 3 n + ld (n).

Dividend, divisor, m, be n any natural numbers, then holds:

$$divisor * 2^{n} > dividend \geq divisor * 2^{m} \wedge divisor * 2^{m+1} > dividend \Rightarrow$$
$$dividend - divisor * 2^{n} + \sum_{i=m}^{n-1} divisor * 2^{i} = dividend - divisor * 2^{m}$$

In other words, you subtract the divisor in the n-th step, notes, the difference is less 0, added the divisor in the following steps as long as the amount greater than zero

is making the largest limiting m is determined. The quotient bits n up to m + 1 then 0, the bit of m 1. It is eliminated so the otherwise necessary but unpleasant perceived, addition of correction in an unsuccessful attempt of subtraction, this reduces the number of operations of an iteration step from 4 to 3. You need therefore in one step either a subtraction or addition, and each one Shift operation for the divisor and quotient.

My searches regarding this modification on the Internet were unsuccessful. I conclude that this simple fact is not well-known knowledge! The square division algorithm is not significantly worse than the multiplication.

2 The client

It implements the Protocol to communicate with the server, as well as the remote Procedures that allow the server a file management - everything in the class serial communicate in the package. In addition one assembler, a Java compiler and a user interface.

2.1 The assembler

The assembler [2, Chapter 6-14, page 21-87] a deck generated from source files means one or more Objekdateien - which meets the requirements of BIOS. The Operation details of translation

Source file of scanner → → → deck parser → code generation

The parser is the controlling part. He calls the scanner on the token from the To extract source power, recognizes the correctness or error, which causes Code generation, and writes at the end of the translation of the code into a destination file.

2.1.1 Of the scanner assembler

The source files are communicated in an initialization step the scanner with which it initializes the stack of the source files. A source row has the following structure.

sourceLine ← {whitespace | include | comment | undefine |} Macro {\"A:\"} | {{\"A:\"} token}

[define | XmlWriter::endcomment create] \"\\n\"

where whitespace defines the separators. Note that terminal symbols by whitespace separated must be, because it otherwise as identifier ident recognized be. Terminal symbols are identifiers with a specific meaning in FORTH.

whitespace ← [\\0-] -all control characters and the space

Include a more source file can directly inserted in the current source file be. This directive sets a new file onto the stack of the source files. Read is always from the top file. Only when this has been read completely, it is taken from the stack.

ident include ← \"INCLUDE\"

← ident [!-~] +. ArgumentException: The incoming token has expired. Ge
Spaces

Comments may be present any source code, but cause no action.

Comment ← (\"charsequence\")

XmlWriter::endcomment-create ← \"\\\" [^\n]*[to the end of the line

← CharSequence [-~\\t\\n]+] -any string of printable characters

-Bem: always a limiting sign has
for example \")\".

-Comes this character detached in

2.1.1 Of the scanner assembler

Sequence above,.

-It is with a \"\\\" to mask.

It is to define possible your own macros and delete again. A macro is a named text replacement, which once defined, any inserted into the source code can and of the scanner is replaced with the associated text. Macros can call other macros that are but non-recursively. The naming is not restricted, therefore everything through a macro can be overridden really. Macros are not the standard defined in!

define ← \"define\" userdefined expansion

undefine ← \"UNDEFINE\" userdefined

UserDefined ← ident -the identifier of the text replacement

expansion ← [^ \\n]*] -the rest of the source line

In addition, there are predefined macros, but can not be deleted. All Macros, including those defined, can the alternative stack with an odd Select number of preceding prefixes \"A:\". An even number, including zero, selects the default stack.

macro ← \"EI\" | \"DI\" | \"QI\" | \"PI\" | \"SETIMASK\" | \"GETIMASK\" | \"SP!\" | \",\" | \"> BODY\" |
\"+!\" | \"C+!\" | \"H+!\" | \"2!\" | \"2@\" | \"2DUP\" | \"2OVER\" | \"2SWAP\" | \">R\" | \"?\"DUP\" |
\"ABS\" | \"ALIGN\" | \"ALIGNED\" | \"ALLOT\" | \"C,\" | \"BL\" | \"CELLS\" | \"CHAR +\" |
\"CHARS\" | \"HALVES\" | \"COUNT\" | \"DECIMAL\" | \"ERASE\" | \"FILL\" | \"HEX\" | \"I\" |
\"J\" | \"MAX\" | \"MIN\" | \"MOVE\" | \"R>\" | \"ROLL\" | \"RED\" | \"S > D\" | \"SWAP\" | \"2 > R\" |
\"UNLOOP\" | \"2R >\" | \"2R@\" | \" \"FALSE\" | \"TRUE\" | \"TUCK\" | \"WITHIN\" | \"D+\" |
\"D-\" | \"D0 <\" | \"D0 =\" | \"D2 BORN\" | \"D2 \A\" | \"D<\" | \"D>\" | \"D=\" | \"D > S\" | \"DABS\" |
\"DMAX\" | \"DMIN\" | \"2 RED\" | \"\" YOU < \"|\" YOU > \"|\"F!\" | \"F@\" | \"FALIGNED\" |
\"FDEPTH\" | \"FDROP\" | \"FDUP\" | \"FLOAT +\" | \"FLOATS\" | \"F0 <\" | \"F<\" | \"F0 =\" |
\"FOVER\" | \"FSWAP\" | \"DF!\" | \"DF @\" | \"DFALIGNED\" | \"SFALIGNED\" | \"FABS\" |
\"SFLOAT +\" | \"DFLOAT +\" | \"DFLOATS\" | \"SFLOATS\" | \"TIB\" | \"#TIB\" | \"> IN\" |
\"<\" | \">\" | \"=\" | \"<>\" | \"<!\" | \">!\" | \"=!\" | \"<>!\" | \"U<\" | \"U>\" | \"U<!\" | \"U>!\" |
\"SOURCE\" |
\"ABORT\" \"charsequence\" \"\" |
\".(\"charsequence\") \"|\"
Charsequence [CHAR] |
\"[']\" charsequence |
\"CONSTANT\" ident |
\"CREATE\" ident |

'VARIABLE' ident |

'2 VARIABLE' ident |

\"PROCEDURE\" ident |

(\"LITERAL\" |) (\"VALUE\") ident |

\"Modules\" ident |

\"MARKER\" ident |

\"FORGET\" ident |

UserDefined

Is a CONSTANT, VARIABLE, CREATE, 2 VARIABLE, PROCEDURE, MODULE, or MARKER is encountered in the source code, the scanner identifies the source as resident. This marking is mandatory for the deck.

The discovery of the token is the actual purpose of scanning. Also tokens can Select stack with the preceding prefixes \"A:\".

token ← (executable | ident | number | floating | string | control |) (EOF)

Executable ← \"NOP\" | \"!\" | \"@\" | \"C!\" | \"C@\" | \"H!\" | \"H@\" | \"DEPTH\" | \"DROP\" | \"2DROP\" |
\"NIP\" | \"PICK\" | \"PUT\" | \"VAL\" | \"DUP\" | \"OVER\" | \"R@\" | \"R1 @\" | \"SAVE\" |
\"STOP\" | \"_SP!\" | \"SP @\" | \"+\" | \"-\" | \"1+\" | \"1-\" | \"2*\" | \"2\" | \"AND\" | \"CELL +\"
|
\"HALF +\" | \"INVERT\" | \"LALT\" | \"NEGATE\" | \"OR\" | \"RSHIFT\" | \"XOR\" |
\"+B\" | \"-B\" | \"LSHIFTC\" | \"RSHIFTC\" | \"0<!\" | \"0<\" | \"0=!\" | \"0=\" | \"0<>!\" |
\"0<>\" |
\"0>!\" | \"0>\" |\"H*\" | \"EXIT\" | \"CALL\" | \"TRAP\" | \"BRANCH\" |
\"0BRANCH!\" | \"0BRANCH\" | \"B@\" | \"B!\" | \"BREAK\"

number ← [+-] ([0-9] + |) [0-9a-fA-F] + \"H\")

← floating [+-] [0-9] +.[0-9]*([eE][+-][0-9]+)?

string ← \"U\" \"charsequence\" \"\" | -String of characters of the Unicode code
\"S\" \"charsequence\" \"\" |
\"C\" \"charsequence\" \"\" |
\"\" \"charsequence\" \"\"

← \"IF\" control | \"ELSE\" | \"THEN\" | \"ENDIF\" |
\"?\"DO\" | \"DO\" | \"+ LOOP\" | \"LOOP\" |
\"BEGIN\" | 'WHILE' | \"REPEAT\" | \"UNTIL\" | \"AGAIN\" |
\"LEAVE\" |
\"CASE\" | \"OF\" | \"ENDOF\" | \"ENDCASE\" |

2.1.1 Of the scanner assembler

\"TO\" |

\"LOCAL\" | \"2LOCAL\" | \"LOCALS\" | \"2LOCALS\" | \"|\" | \"PURGE\" |

\"THROW\" | \"CATCH\" |

\"$ ORG\" | -absolute start address of code

MODULE NAME -Name, under which the generated module in the Dictionary

-entered is

EOF ← read all source files, the stack of the source files is empty

The data of a token.

```
public class token
{
  String sourceFile;                \/ \/ filename
  public boolean alternate;          \/ \/ marks alternative stack
  public int child;                  \/ \/ type of token
  public int line;                   \/ \/ sourceline number
  public int col;                    \/ \/ column in sourceline
  public long val;                  \/ \/ number
  public string ident;              \/ \/ identifier
  public string string;             \/ \/ read string
  double d;                         \/ \/ floating
}
```

Extensions of the standard relating to the scanner are recorded in table 60.

Language elements	***Declaration***
\"U\" \"charsequence\" \"\"	Defines a string in Unicode. Is Java required.
DEFINE UserDefined expansion	Defines a macro, a pure text replacement.
\"UNDEFINE\" userdefined	Deletes the definition of a macro

Table 60: Not defined in FORTH token

2.1.2 Of the parser of the assembler

The rules of grammar of the parser.

parse ← [ident \"Module NAME\"] {statement | colonDefinition}.

statements ← {{\"A:\"} (\"NOP\" | \"!\" |)} \"@\" | \"C!\" | \"C@\" | \"H!\" | \"H@\" | \"DEPTH\" | \"DROP\" |

\"2DROP\" | \"NIP\" | \"PICK\" | \"PUT\" | \"DUP\" | \"OVER\" |

\"R@\" | \"R1 @\" | \"SAVE\" | \"STOP\" | \"_SP!\" | \"SP @\" | \"+\" | \"-\" | \"1+\" |
\"1-\" | \"2*\" | \"2\/\" | \"AND\" | \"CELL +\" | \"HALF +\" | \"INVERT\" |
\"LALT\" | \"NEGATE\" | \"OR\" | \"RSHIFT\" | \"XOR\" | \"H*\" |
\"+B\" | \"-B\" | \"LSHIFTC\" | \"RSHIFTC\" | \"0<!\" | \"0<\" | \"0=!\" | \"0=\" |
\"\" U0 < \"|\" U0 > \"|\"0<>!\" | \"0<>\" | \"0>!\" | \"0>\" | \"CMP\" | \"EXIT\" |
(ident | number | build) (\"VAL\" |) \"CALL\" | \"TRAP\" | \"BRANCH\" |
\"0BRANCH!\" | \"0BRANCH\" | \"LABEL\" |
(\"$ ORG\") |
\"B@\" | \"B!\" | (\"BREAK\") |
\"CATCH\" | \"THROW\" |
(\"LOCAL\" |) Ident (\"2LOCAL\") |
(\"LOCALS\" |) (\"2LOCALS\") ident {ident} \"|\" |
\"PURGE\" number |
{ \"A:\" } \"TO\" (number | ident) |
{ \"A:\" } (string | floating | ident | number | build |) \"LEAVE\" |
ifStatement | doStatement | caseStatement | {(beginStatement)}.

colonDefinition ← (\":\" | \": LOCAL\") statements {\"A:\"}-ident \";\".
ifStatement ← \"IF\" statements [\"ELSE\" statements] (\"ENDIF\" |) (\"THEN\").
doStatement ← (\"DO\" |) \"?(\"DO\") statements (\"LOOP\" |) (\"+ LOOP\").
caseStatement ← \"CASE\" {statements \"OF\" block \"ENDOF\"} \"ELSE\" statements \"ENDCASE\".
beginStatement ← \"BEGIN\" statement
('WHILE' statements \"REPEAT\" |) \"UNTIL\" | (\"AGAIN\").
← build \"BUILD >\" expr \"DOES >\".
expr ← (build | number | floating) {\"1 +\" |} \"1-\" | \"2*\" | \"2\/\" | \"CELL +\" | \"HALF +\" | \"INVERT\" |
\"ABS\" | \"FABS\" | \"D > F\" | \"F > D\" | \"FLOOR\" | FROUND
{expr (\"+\" |)} \"-\" | \"AND\" | \"OR\" | \"XOR\" | \"LALT\" | \"RSHIFT\" | \"F+\" | \"F-\" | \"F*\" |
\"F\/\" | \"F**\" | \"*\" | \"\/\" | \"MOD\" | \"FMAX\" | \"FMIN\" | \"MAX\" | (\"MIN\")
{ \"1+\" | \"1-\" | \"2*\" | \"2\/\" | \"CELL +\" | \"HALF +\" | \"INVERT\" | \"ABS\" | \"FABS\" |
\"D > F\" | \"F > D\" | \"FLOOR\" | {{\"FROUND\"}}.

2.1.2 Of the parser of the assembler

Some language elements are not available , sie Sh(d) in table 61 arrested.

Language Declaration	
\"[\" statements \"]\"	Purpose: Commands in this block are not compiled, but executed immediately. Reason: A compiler may not have this capability. Replacement: such blocks define as a stand-alone procedure or the Move block in the startup code in a module
IMMEDIATE	Purpose: tagged immediately executable words Rationale: all words in the dictionary, as well as words in the created Module, are immediately executable. Replacement: not necessary, as implicitly given.

Language elements	***Declaration***
\"COMPILE\" | [COMPILE] | POSTPONE	Purpose: the production of an immediate force Word Rationale: superfluous, since all immediate words are
RECURSE	Purpose: enables recursion Rationale: all procedures are recursively callable
: NONAME	Purpose: defines a nameless procedure Reason: Words in the dictionary you must inen Namen have Replacement: named local procedure
\"ENVIRONMENT?\"	Purpose: Queries the system environment Rationale: There are no system environment
EVALUATE	Purpose: interprets the contents of a string Reason: A compiler may not have this capability. Replacement: define a macro or procedure

Table 61: language elements of FORTH, which could not be realized

Were following, not defined in the standard language elements set out in table 62 included:

Language elements	***Declaration***
\"Module NAME\" ident	The loadable module is designed with a specific name in the Dictionary are entered. It must always be the first Statement in the source file be.
(ident | number | build) $ORG	The absolute starting address of the generated code is prescribed;. has only an effect if the target file \"romable\" should be.
(ident | number | build) LABEL	A brand that can be used as jump target.
': LOCAL\"ident	The definition allows a local procedure, the is not entered in the dictionary.
\"PURGE\" number	Thus the last declared number variables are immediately deleted. Is required by Java.

Language elements	***Declaration***
number [\"VAL\"]	*number* kann aeven a Double word – 64 Bit – Repräsentieren. It will automatically to load code the double Word creates - a modification of Standards

Table 62: In addition defined language elements

All rules that are derived after statements, monitor the health of the stack the return addresses - the stack height must be the derivation to statements do not change, and emit an error message in violation of this condition. For the Data stack is These Monitoring not possible,. man would have to the Compile-time know how many elements does a procedure call to the data stack and it returns. If the procedure in the dictionary, this is to the incorporated Maybe for the client not available (offline), and the number of parameters therefore is not known.

Local variables can be declared, to the stack of return addresses is applied, and can, if once declared, using the name be referenced, the necessary code is automatically generated. The variables can again with PURGE be deleted. An automatic deletion is one in any case at the end of Made the procedure or the initial code, if they are not previously with PURGE have been released.

2.1.2.1 build

Evaluates arithmetic expressions, compile-time and can therefore only constant process. The result is a token of type number or floating.

2.1.2.2 Flow controls

The control statements translate into sequences, the in the source stream of Insert the scanner, equal to a macro.

2.1.2.3 Colon definitions

This defines a procedure. If it is not local, initial code will automatically produces, which they in the dictionary enters. In any case, the procedure in which is Reference list.

2.1.2.4 Statements

Here be the Machine instructions recognized, translated and on the Code tables redirected. The management of the references is an important task of part of.

2.1.2.5 Parse

After translating the source files is tries to resolve the references. After all references of jump commands and CALL by 16-bit must this step Offset values be replaced, otherwise a local reference could not be resolved. The remaining references must be resolved by the binder and bound. Referenced values that are declared in the program must be bound only,. their references have the type local (BIOS). The rest must be in the dictionary to find be. If their use by VAL or TRAP them, is their type is *positive, otherwise defines the type, and thus the loading code, the referenced token* and negative is of type.

Finally, the deck is written if the type of the destination file linkable. Alternatively, there are the type of romable, of the code as a vector of 32-bit words in VHDL Syntax on the destination file writes. This code can directly in eine VHDL description be adopted.

2.1.3 Code generation

Manages two vectors of code, one for the initial code of the program, and one for the resident part - procedures, and strings. The selection of the desired Code vector will make the parser. There are quite a few functions on a vector applied be can. attach override a date at one certain Position, Optional read,. etc. One important Action item is the Logging of the height of the stack of return addresses.

2.1.4 Interface

The method in table 63 provides the class assembler of package forthassembler.

Method
public static boolean assemble(String_option,_String_args,_String_editor,_String_report,) (boolean applet) assemble forth source to produce targets ending with \".obj\" or \". Rome\" and listing ending \".lst\" option string, \"linkable\" or \"romable\" args string sourcename Editor string name of preferred editor report string name of the report file applet boolean true, if started from applet return boolean true, if successful

Table 63: Method Assembler.assemble to invoke of the assembler

2.2 Java

The compiler translates directly in MaSCHinencode, but generated from the Source files Assembler programs, the with the Assembler to translate are. It is beneficial that the Compilat is readable and editable, which is a review of the Correctness of translation allows. The compiler works incrementally. In each Step is called the parser which returns a list of referenced files what creates a reference file with extension \".ld\". Each list entry has a Note If the source file is already translated, or must be translated separately. The Iteration ends when all required references are translated. The order of the Steps of inside a can.

Source file of scanner → → → translation of → destination files parser

The parser is part of flying, he reads the files completely, created this He writes operator trees and reference lists or scopes, the latter in a separate file with extension \". h\", now called header to renewed translation of parsing to save, and resolves the reference lists. Usually failing in one Step, but additional sources from the package and the import declarations need to be translated, or whose scopes are loaded, if they already successful have been translated. This is done recursively, using the final translation in imported files is eliminated. Only when all references of the source file or the files are dissolved from the package, is done recursively descending, the translation - the training ranking the header and its operator trees for the production of the assembler programs. Of each source file, two files are generated. The resident module with extension \". fs\", and the volatile, initializing part - the code of the static blocks and

Initializers - with suffix \". start.fs\".

2.2.1 Of the scanner

He reads the source file and extracts the next token from the request
Source current. A source line obeys the following rule.

SourceLine ← {whitespace | token} [XmlWriter::endcomment-create] \"\\n\".

whitspace ← [\\0-]

← comment token | XmlWriter::endcomment create | charConst | number | lnumber | stringLiteral | dnumber | ident |

keyword.

Comment ← V \\ born. \\ V **

endcomment ← W[^\\n]*

[charConst ← \\b|\\t|\\n|\\f|\\r|\[0-9a-fA-f][0-9a-fA-f]?[0-9a-fA-f]?[0-9a-fA-f]? |[0-3][0-7]?[0-7]? |

[0-7][0-7]? |

number ← 0 [0-7] born |0x [0-9a-fA-F] + |[0-9] +.

lnumber ← (0 [0-7] born |0x [0-9a-fA-F] + | [])0-9] +) [lL]

stringLiteral ← \"(\\\\.|)\"[^\"])*\"

dnumber ← [0] born.[0-9]*([eE][+-][0-9]+)?[fF]

← ident [a-zA-Z$ _] [a-zA-Z$ _0-9].

← keyword \"+\" | \"-\" | \"*\" | \"\/\" | \"%\" | \"==\" | \"!=\" | \"<=\" | \">=\" | \"<\" | \">\" | \"

\"!<\" | \"!>\" | \"++\" | \"--\" | \";\" | \":\" | \"?\" | \"(\" | \")\" | \"{\" | \"}\" | \"[\" | \"]\" | \".\" | \"!\" | \"~\" | \",\" |

\"break\" | \"else\" | \"if\" | \"new\" | \"return\" | \"#ass\" | 'while' | \"for\" | \"do\" | \"continue\"

|

\"try\" | 'catch' | \"finally\" | \"switch\" | \"throw\" | \"case\" | \"default\" | \"instanceof\" |

\"this\" | \"super\" | \"true\" | \"false\" | \"zero\" | \"+=\" | \"-=\" | \"*=\" | \"\/=\" | \"%=\" |

\"

\"import\" | \"extends\" | \"implements\" | \"assert\" | \"throws\" | \"byte\" | \"short\" | \"char\"

|

\"int\" | \"long\" | \"float\" | \"double\" | \"boolean\" | \"void\" | \"public\" | \"private\" |

\"protected\" |

\"static\" | \"final\" | \"synchronized\" | \"volatile\" | \"transient\" | \"native\" | \"abstract\" |

\"strictfp\" | \"interface\" | \"class\" | EOF.

Come tare weRoden not ignored, the parser it into the operator tree inserts, and
the translator inserts it as FORTH comments in the generated assembler program.

The keywords have been!< as an alternative to > = and! > for < = in addition defined.
Experience logic errors are by confusingly, of < = < or

> = > arise, often unable to immediately identify and prove to be very stubborn. In FORTH is < = by definition \"> INVERT\" or > = \"< INVERT\" expressed, making it according to my own experience, also to no confusion of < with < = or *> with > = comes. I think that the defined alternatives are easier to read than the* Standard and probably - they would be standard - not generally to less logical Would result in errors in programs. < = or > = completely to delete like in FORTH,. I consider unfeasible.

2.2.2 Of the parser

The accompanying the parser underlying grammar is one a corrected version Grammar of the Institute of computer science of SSW [11] Johannes Kepler University and is as the template LALR(1). Unlike the template, the new grammar recognizes the JDK Source files the Package *J2SDK-sec-1_4_2-src-scsl* by Sun anstandslos. The Modifications are:

- Label are usually possible statement, assert is detected
- the rule of expression2Rest has been completely revised and now consider them Hierarchy of the operators.
- *! > was as an alternate identifier for < = recorded*
- *! < was as an alternate identifier for > = recorded*

The new grammar is as follows:

parse ← [package \"qualident\";] {importDeclaration} {typeDeclaration}.

qualident ← ident {\".\" ident}.

importDeclaration ← import \"ident qualifiedImport\";.

qualifiedImport ← {\".\" ident} [\". born\"].

typeDeclaration ← \";\" | classOrInterfaceDeclaration.

classOrInterfaceDeclaration ← [modifiers] (classDeclaration | interfaceDeclaration).

type ← (qualident | basicType) bracketsOpt.

basicType ← \"byte\" | \"short\" | \"char\" | \"int\" | \"long\" | \"float\" | \"double\" | \"boolean\".

BracketsOpt ← {\"[\" \"]\"}.

typeList ← type {\",\" type}.

formalParameter ← [\"final\"] type variableDeclarartorId.

qualidentList ← qualident {\",\" qualident}.

variableDeclarator ← ident variableDeclaratorRest.

variableDeclaratorId ← ident bracketsOpt.

variableDeclaratorRest ← bracketsOpt [\"=\" variableIntitializer].

variableIntitializer ← arrayInitializer | expression.

classDeclaration ← \"class\" ident [\"extend\" type] [\"implement\" typeList] classBody.

classBody ← \"{\" classBodyDeclaration \"}\".

← classBodyDeclaration \";\" | [\"static\"] (block |) [modifiers] memberDecl).

memberDecl ← constructorDeclaratorRest | \"void\" voidMethodDeclaratorRest |
classDeclaration | interfaceDeclaration | methodOrFieldDeclaration.

methodOrFieldDeclaration ← methodOrFieldRest ident.

methodOrFieldRest ← methodDeclaratorRest | variableDeclaratorsRest.

variableDeclaratorsRest ← variableDeclaratorRest {\",\" variableDeclarator}.

arrayInitializer ← \"{\" [variableIntitializer {\",\" variableIntitializer}] [\",\"] \"}\".

methodDeclaratorRest ← formalParameters bracketsOpt [\"throws\" qualidentList] (\";\" |)
(block).

voidMethodDeclaratorRest ← formalParameters [\"throws\" qualidentList] (\";\" | block).

constructorDeclaratorRest ← formalParameters [\"throws\" qualidentList] block.

formalParameters ← \"(\" [formalParameter {\",\" formalParameter}]\")\".

interfaceDeclaration ← \"interface\" ident [\"extends\" typeList] interfaceBody.

← interfaceBody \"{\" interfaceBodyDeclaration \"}\".

interfaceBodyDeclaration ← \";\" | [modifiers] interfaceMemberDecl.

voidInterfaceMethodDeclaratorRest interfaceMemberDecl ← \"void\" | classDeclaration |
interfaceDeclaration | interfaceMethodOrFieldDeclaration.

interfaceMethodOrFieldDeclaration ← interfaceMethodOrFieldRest ident.

interfaceMethodOrFieldRest ← constantDeclaratorsRest |
interfaceMethodDeclaratorRest.

constantDeclaratorsRest ← constantDeclaratorRest {\",\" constantDeclarator}.

constantDeclaratorRest ← bracketsOpt \"=\" variableIntitializer.

constantDeclarator ← ident constantDeclaratorRest.

interfaceMethodDeclaratorRest ← formalParameters bracketsOpt [\"throws\" qualidentList] \";\" .

voidInterfaceMethodDeclaratorRest ← [qualidentList \"throws\"] \";\".

statement ← {ident \":\"} (block)
| \"assert\" expression [\":\" expression] \";\" -is only detected!
| parExpression \"if\" statement [\"else\" statement]
| \"for\"\"(\"[forInit] \";\" [expression] \";\" [forUpdate] \")\" statement
| \"while\" parExpression statement
| do \"statement\"while\"parExpression\";
| 'try' block (catches [\"finally\" block] |) ('finally' block)
| parExpression \"switch\" \"{\" switchBlockStatementGroups \"}\"
| \"synchronized\" parExpression block
| \"return\" [expressio] n] \";\"
| throw \"expression\";
| break [ident];
| continue [ident];
| \";\"
| statementExpression \";\"
| \"#ass\" stringLiteral {\"+\" stringLiteral} \";\") . - inline code

← block \"{\" blockStatement \"}\".

blockStatement ← localVariableDeclaration \";\" | classOrInterfaceDeclaration | statement.

localVariableDeclaration type variableDeclarators ← [\"final\"].

variableDeclarators ← variableDeclarator {\",\" variableDeclarator}.

forInit ← localVariableDeclaration | statementExpression moreStatementExpression.

forUpdate ← statementExpression moreStatementExpression.

statementExpression ← expression.

moreStatementExpression ← {\",\" statementExpression}.

catches catchClause ← {catchClause}.

catchClause ← \"catch\" (\"formalParameter\") block.

2.2.2 Of the parser

switchBlockStatementGroups ← {switchBlockStatementGroup}.

switchBlockStatementGroup ← switchLabel {switchLabel} blockStatement.

switchLabel ← (\"default\" |) (type \"case\") \":\".

expression ← expression1 [assign expression].

expression1 ← expression2 [conditionalExpr].

conditionalExpr ← \"?\"expression\":\" expression1.

expression2 ← expression3 [expression2Rest].

expression2Rest ← logicalOr.

logicalOr ← [logicalAnd] {\"|\" expression3 [logicalAnd]}.

logicalAnd ← [relop] {\"}

relop ← \"instanceof\" type [(\"==\" |)] (\"!=\") expression3 [instanceOf]] |
[or] [(\"==\" | (\"! =\") expression3 [instanceOf] |
(\"<=\" | \"<\" | \">=\" | \">\" | \"!< \"|\"! > \") [or] expression3].\"

← instanceOf (or |) (\"instanceof\" type).

or ← [and] {(\"|\" |)} (\"^\") expression3 [and]}.

and ← [shift] {\"}

← SHIFT [lowArith] {(\">\" |)} (\"<<\"_|_\">>>\") expression3 [lowArith]}.

ArgumentException: The incoming token has expired. Get a new access token from the Authorization Serve

ArgumentException: The incoming token has expired. Get a new access token from the Authorization Serv

ArgumentException: The incoming token has expired. Get a new access token from the Authorization Server

ArgumentException: The incoming token has expired. Get a new access token from the Authorization Se
ArgumentException: The incoming token has expired. Get a new access token from the Aut
| superSuffix \"super\"
| literal
| \"new\" creator
| ident {\".\" ident} [identifierSuffix]
| basicType bracketsOpt \".\" \"class\"
| \"void\" \".\" \"class\".

argumentsOpt ← [argument].

← argument ([expression {\",\" expression}]).

superSuffix ← argument | \".\" ident argumentsOpt.

literal ← charConst | number | lnumber | stringLiteral | dnumber | \"false\" | \"true\" | \"null\".

Creator ← basicType arrayCreatorRest | qualident (arrayCreatorRest | (classCreatorRest).

arrayCreatorRest ← \"[\" (\"\") \"]\" bracketsOpt arrayInitializer | expression \"]\" {\"[\" expression \"]\"} (bracketsOpt).

classCreatorRest ← argument [classBody].

identifierSuffix ← \"[\" \"]\"bracketsOpt\".\" \"class\" | argumentsOpt | \".\" (\"class\" |) (\"this\").

← selector \".\" (ident argumentsOpt |) \"super\" argument | (\"new\" innerCreator) | \"[\" expression \"]\".

innerCreator ← ident classCreatorRest.

castExpression ← expression.

parExpression ← (\"castExpression\").

assign ← \"=\" | \"+=\" | \"-=\" | \"*=\" | \"\/=\" | \"%=\" .

← prefix \"++\" | \"--\" | \"!\" | \"~\" | \"+\" | \"-\" .

Postfix ← \"++\" | \"--\" .

← modifiers \"public\" | \"protected\" | \"private\" |
\"abstract\" | \"final\" | \"strictfp\" | \"static\" | \"transient\" | \"volatile\" |
\"synchronized\" | \"native\".

The parser creates a header and fills it recursively descending the source stream, analyzing, on.

```
class header
 {
  String name;                  \/ \/ name of java sourcefile
  String [] imports;            \/ \/ import declarations
  String myPackage;             \/ \/ package
  Vector scopes;                \/ \/ starting scopes of all declared classes
  Scope base;                   \/ \/ root tree scope of scope
  final int depth;              \/ \/ priority: 0 sourcefile...
 }
```

The header is an integral class, the methods for creating, reading and

2.2.2 Of the parser

Write a header provides. The actual information is in the scopes and the registered therein declarations of Base types ClassType, VariableType and MethodType, all derived from Basic.

```
class scope
 {
  private static vector table = new vector();  \/ \/ holds all scopes for indexing
  static final int automatic = 1, \/ \/ storage types
                   heap = 2,.                    \/ \/ static
                   classed = 3, \/ \/ object class or interface
                   BLOCK = 1, \/ \/ scope types
                   MAIN = 2,.
                   LOOP = 3;
                   TRY = 4.
                   FINALLY = 5,.
                   CATCH = 6,.
                   SWITCH = 7,.
                   SEQUENCE = 8,.
                   DUMMY = 9;
  final String prefix;                  \/ \/ name of class, if starting scope
  int storageClass;                     \/ \/ automatic or heap for static scope
  final string trailer;                 \/ \/ trailing path of scope
  int offset = 0;                       \/ \/ offset for class variables
  private TreeMap map;                  \/ \/ entries of scope
  private vector follower;              \/ \/ scopes of inner classes
  Scope prev;                           \/ \/ enclosing scope
  Vector label;                         \/ \/ list of labels (for jumps)
  int block;                            \/ \/ type of scope
  String exception = null;              \/ \/ for exception handling
  protected static scope cur = null;    \/ \/ for loading of header
}
```

This class provides many methods for the management and search of entries in the Tree of the scopes. The class basic implements the CRC-16, as a base to the Generate a hash code is used.

```
public class basic
 {
   int modify;                  \/ \/ modifier
   Token name;                  \/ \/ name
   int version;                 \/ \/ holds the hash code
 }
```

The derived classes overload all hash code from the signature of the object.
expects will.

```
class ClassType extends basic
 {
  ClassType extend;             \/ \/ superclass
  ClassType [] implement;        \/ \/ abstract classes and interfaces
  Scope scope;                  \/ \/ scope of class
  Vector statics;                \/ \/ list of operator trees of static blocks
  HashSet unresolved;            \/ \/ list of names of unresolved references
  String comment;               \/ \/ trailing comment
 }
class VariableType extends basic
 {
   int offset;             \/ \/ relative position in object, for class variables
   Boolean referenced = false;   \/ \/ needed in code generation
   Type type;                   \/ \/ type of variable
 }
MethodType class extends basic
 {
   Type type;                    \/ \/ type of result
   Parameter [] parameter;       \/ \/ list of parameters
   String [] throwing;           \/ \/ list of exceptions
   Scope scope;                  \/ \/ scope of method
   Vector operation;             \/ \/ list of operator trees
   String comment;               \/ \/ trailing comment
 }
```

Each statement in the source code is translated into one or more operator trees, the
Are trees of operations in Postfix notation. The roots of the trees are in a
consecutive list entered operation or statics.

```
class operation
 {
   Operation left;               \/ \/ left son
   Operation right;              \/ \/ right son
```

```
    Keyword operator;              \/ \/ operator
    static int labelno = 0;
    Token name;                     \/ \/ name, if identifier
    Type type;                     \/ \/ resulting type
    Scope scope;                   \/ \/ scope
    String code = \"\";             \/ \/ resulting FORTH code
    Boolean loaded = false;        \/ \/ resulting value on stack
}
```

The leaves of the tree (references) are characterized by the operator of LEAFSY. A sheet containing the information rule expression3, these are from left to handled right.

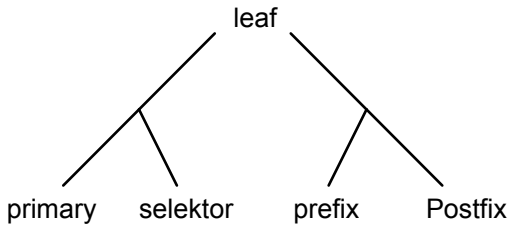


Figure 19: Building a leaf of a tree of the operator

2.2.3 The translation

It translates each class separately. First is the code for the static variables When you choose a module name declared, and code for the static blocks generates. Finally, methods and constructors are translated.
Core is controlled by the method the evaluation of lists the operator trees *code. Code of following structure results from each tree.*

← [locals] statement [exception] [free purge] code [exceptionhandling].
locals ← \"LOCALS\" name {name} \"|\".
name ← [0-9] + §
statement... the code of the operator tree
← exception \"FALSE DUP IF\" exceptionlabel \"LABEL TRUE ENDIF\".
exceptionlabel ← std [0-9] +.
free... the references of the objects in the local variables will be humiliated
purge ← PURGE [0-9] +.
ExceptionHandling... Code for the introduction of an exception handling

The statement required any hidden local variables to intermediate results, dynamic memory management here include references that take into account

must, it must be created first. Auxiliary variables generated by the compiler
marked with a leading or trailing characters \"article\" here hot,
they \"0 §\", \"1 §\", etc. Whether they are needed is the evaluation of the operator tree
traverse through the method, as well as the need for an exception handling.
Should this be possible, is at the regular end of FALSE on the stack put, otherwise
TRUE. The size of this test indicates whether finally initiated exception handling
must be.
Method causes traverse optimizations like constant expressions, partially
by. Assignments to variable account of memory management,
may as well be existing, has become superfluous data from the data stack,
eliminated. Special operators, table 64, not indicated by Java.

Operator	***Action***
ALLOCATESY	outerclass superclass- Allocates memory for an instance of a class. Fix awarded are following relative Adresspositionen in a class object. 0 Reference to the superclass 4 Reference to the object of the derived classes 8 Reference to the outer class 12 Reference to the table of tuples (hash code, start address) of the not static methods, including constructors 16 Reference to a FORTH string, the class name contains. From 20, the attributes are starting with offset 0 At the References Super class and external Class is the Reference counts incremented, the derived class not - so independent town
PUSHSY	A new scope is opened and applied its local variables
POPSY	The current Scope is closed and to the Surrounding returned. Where the local Variables of the Stack the Return addresses are eliminated, after possibly affected References of memory management have taken into account.
LOCKSY	Locks an object for other processes.
UNLOCKSY	Releases an object for other processes.

Table 64: Additional operators that are not dictated by Java

2.2.3.1 Deferred reference counting

The reference counter is only incremented, if the object on the stack from a (indexed)

2.2.3 The translation

Variable is loaded, and is the result of the right side of an assignment, or the
Return value of a method, or a throw statement is, or an actual parameter
a method or constructor call, but not the hidden object reference \"section this\"
– other than \"this article\" is the return value one *return, where* very probably too
increment-, is.

The reference count is decreased, if the hidden destructor \"~ destructor\" a
Object is invoked. He achieved a value of 0, the object is deleted, after
the destructors for the contained objects were called. Is a local variable
deleted, the destructor, except for \"this article\", is invoked.

In this way, the costs can be kept minimal for memory management.

2.2.3.2 Classes

Each class has an automatically generated constructor and destructor, but
übersch programmer Reben weauthorities can. For all classes is an integer
\"_dynamicBlocking\" before declared,.It shouldat one later Implementation the
Operators LOCK and UNLOCK are used. For each class the static is
Integer variable \"_staticBlocking\" - for synchronized static methods - before declared,
just falls für provided the locking protocol.

2.2.3.3 Variable

Static variables are created on the heap, all the others are either part
a class thus in dynamic memory, or locally declared and therefore
temporary to the stack of return addresses in the processor.

2.2.3.4 Methods

If they are not static, their first (hidden) parameter is \"this article\", the reference
the object of the method. The method can also applies to constructors, a
Exception return,. is the last (hidden) Parameter one alternative
Return address - exceptionlabel specified above. Upon entry into the method
are all actual parameters as local variable onto the stack of return addresses
moved and formally associated with the scope of the method or constructor. Since the
Is to accommodate polymorphism, is a method of indirectly via the BIOS procedure
EXECUTE METHOD, started.

2.2.3.5 Constructors

Member and anonymous classes have as first (hidden) parameters \"section outer\", the
Reference to their appearance object. The first action in the body is always, if there is no call
\"this(...)\" is specified, the production of the Super object, then create the \"
the actual object and assign the reference to \"this section\". Immediately afterwards, all are
non-static initializers and blocks of the class declaration inserted, it
follow the instructions of the constructor. Implicit return value is \"this article\". In fact
the constructor is started by calling the BIOS procedure EXECUTE-NEW.

2.2.3.6 Strings

Use the Unicode character set.

2.2.3.7 JavaArray

Specially for manipulating fields this class in entitled to the java.lang package make is ready.

```
public class JavaArray
 {
   public final int length;      \/ \/ length of array
   public final int shift;       \/ \/ ld(wordlength)
   public final int code;        \/ \/ (crc16, dimension)
   public int array;             \/ \/ reference to physical array
 }
```

The methods of the class are documented in table 65.

Method
JavaArray (int length, int code) constructor length number of elements code lower 16 bits of are the dimension, high part is the type return a new instance of class
public int getElem (int pos) fetch an element of the array POS index return a reference to the element

2.2.3 The translation

Method
public JavaArray clone (a JavaArray) clone an array a array to clone return a clone
public void ~destructor() Decrement (and remove) object and array
public static string createString (int bytestring, int length) create standard string length StringLength ByteString points to initial content return an initialized string object
public static string createUnicode (int forthstring, int length) create an unicode string length StringLength forthstring points to initial content return an initialized string object

Method
public static void kill (object obj, int polymorphically) Decrement (and remove) an object obj the object \"sleep\"-the complete object, if true, only this part otherwise
public static void handler (Exception e) default exception handler e exception
public static void print (string s) print string s s string

Table 65: Interface of the JavaArray class

2.2.4 Interface

The interface is the class compiler of package MyJava. It provides a single
Table 66, ready method.

2.2.4 Interface

Method
public static boolean compile (string arg, string jdk, string editor, boolean applet, boolean) (force) compile source named in loader arg and produce list ARG sourcename get path of jdk sources Editor name of editor applet true, if called from an applet force true, force recompilation of all necessary sources return true, if compiled successfully

Table 66: Method MyJava.compile to call of the compiler

2.3. The user interface

So presents the client on the screen.

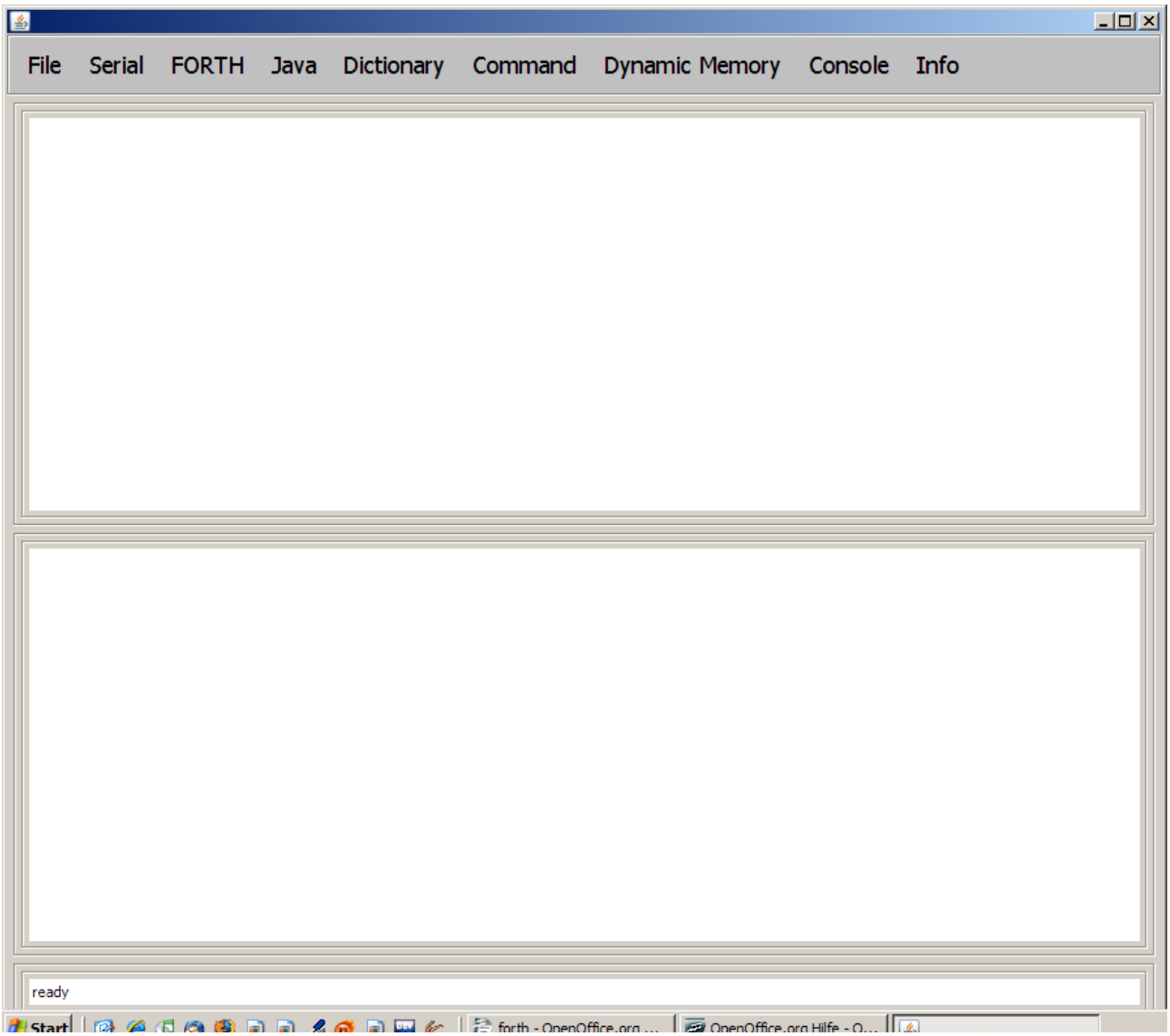


Figure 20: A screenshot of the client

The Application offers one Menu bar, followed by one Output, one Command window and a status bar. A brief explanation only you need Menus, tables 67 to 74.

Menu file	
Menu	***Action***
New project	allows a new project in the root directory of the current project to create by typing its name. The new project will be automatically the current project.
New file	It can be a new file in the project directory created and with the editor be created.
Open project	Opens an existing project which is to the current project.
Open file	Opens a file with the editor.
Close project	Closes the current project. The root directory is the new current Project.
Set editor	The editor can be set.
Exit	Closes the applet and writes the properties in the file myProperties .

Table 67: The file menu

Menu serial	
Menu	***Action***
Port	The used serial port can be selected.
Stop bits	Option 1 (default), 1.5 or 2
Parity	Either even, odd or none (default)
Baud rate	One of the preset values can be selected (9600 default).
Control	Are using RTS\/CTS flow control, or no control
Force changes	The taken forward Settings be taken over and the Interface between server and client is reinitialized.
Default values	Die Standardwerte are elected and can force ü changes ber Nommen are.

Table 68: The menu serial port

Menu FORTH	
Menu	***Action***
Option	The type of the deck can be \"romable\" (VHDL code) and \"linkable\" (Object file, intended for the server) be selected.
Template	With this template, and the assembled code is a VHDL Description of Rome generated be inserted directly in the project can.
Blocksize	the size of an of the used FPGA - Spartan3a 2048 Blockram Byte.
File	The assemblierende file can be selected. It is then assembles and the list of source files, as well as any error messages displayed in the output window.
File list	The files of a loading list are assembled. This list was previously produced by translation of a Java file. The action supports mainly debugging in Java-language sources.

Table 69: The menu FORTH assembler

Menu Java	
Menu	***Action***
Option	Between full renewed Translation all Files, and exclusive translation of modified files can be selected.
GC	The implemented in the BIOS, garbage collector is in code generation be taken into account. It is between \"tricolor marking\", \"reference counting\" and choose \"modified reference counting\".
To forth	You can choose the file to be translated. It is then translated. The list of source files, as well as any error messages are displayed in the output window.
Sun sources	You can choose the directory with the source code of the JDK.
Home	The directory of the source files can be selected.

Table 70: The menu Java compiler

Menu Dictionary	
Menu	***Action***
list	The dict ionary wIRD requested from the server and displayed.
Load forth modules	A loadable deck can be selected. It is on the server the transfer, it binds and initializes.
Load Java modules	All yet not charged Decks one translated,. free selectable Java application are transmitted to the server, He she binds and initializes.
Initialize Java modules	All initiating start decks of a translated, freely selectable Java application be on the Server transfer and running.
Unload	A loaded module can be selected. Then the Server prompted the Dictionary to shorten. The selected module and everything was incorporated in later deleted.

Table 71: The menu Dictionary

Menu command	
Action menu	
Reset	The server is reset and reinitialized.
Abort	The application launched by the server will be canceled and the dynamic Reset memory.

Table 72: The direct commands

ArgumentException: The incoming token has expired. Get a new access token from the Authori	
ArgumentException: The incoming token has expired. Get a new access token from the Authori	***ArgumentException: The incoming token has expired. Get a new access token from the Authori***
ArgumentException: The incoming token has expired. Get a new access token from the Authorization S	The output window will be deleted.
Save output as	The contents of the output window is under a defined name as File saved.
Clear input	The command window will be deleted.
Save input as	The contents of the command window is under a defined name as File saved.
Load file	A freely selectable file is loaded into the command window.
Execute input	The contents of the command window is assembled, loaded from the server and initialized.

Table 73: The console menu

Menu dynamic memory	
Menu	***Action***
available	The size of the free heap and the size of any dynamic memory is displayed.
allocate	A dynamic memory of selectable size is applied to the heap and Initializes.
free	The dynamic memory is freed.
reset	The dynamic memory is reinitialized.
Default size	of dynamic standard size can be defined here.

Table 74: The menu to the dynamic memory

Results

1 Hardware

Implemented and tested on a \"Spartan-3A FPGA Starter Kit\" from Xilinx.
At the heart of these boards is a Spartan-3A-700 k. Furthermore, it contains a 32MBx16 DDR2-RAM, as well as various interfaces, including a serial interface via which Notebook can be connected. For the test of computing power, he was Whetstone benchmark [19] in FORTH implements.

Four variants of the processor, table 75, were realized and tested.

Variations	
FORTHSP	Only contains an Adder in the calculator and realized nur einen Core
FORTHSPM	In addition to the Adder, the calculator contains a 64 x 64-bit Multipliers and realized only a core
FORTHSPC	Contains an Adder in the calculator just realized but 4 cores
FORTHSPMC	In addition to the Adder, the calculator contains a 64 x 64-bit 4 Cores, multiplier and realized

Table 75: The variants of CPU

Following characteristics, table 76, have been identified for them.

Characteristics				
	FORTHSP	FORTHSPM	FORTHSPC	FORTHSPMC
Area (%)	63	72	88	93
Maximum clock (MHz)	73.6	67.5	62.5	57,5
Critical path	Stack management (Indexing)	Stack buffer-Shift register -Stack buffer	Stapelverwalt-Ung (Indexing)	Stapelverwalt-Ung (Indexing)
Whetstone (KWIPS)	8,32	25,15	7,32	22.5

Table 76: The characteristics of variants

1.1 Resources and critical paths

The following collections are the abstracts of the Sysnthesis reports of individual projects.

1.1.1 FORTHSP

Selected device: 3s700afg484-4

Number of slices:	3418 out of 5888	58%
Number of Slice Flip Flops:	2706 out of 11776	22%
Number of 4 input LUTs:	6385 out of 11776	54%
Number used as logic:	6236	
Number used as shift register:	85	
Number used as RAMs:	64	
Number of IOs:	73	
Number of bonded IOBs:	73 out of 372	19%
IOB flip flops:	1	
Number of BRAMs:	12 out of 20	60%
Number of GCLKs:	4 out of 24	16%
Number of DCMs:	2 out of 8	25%

Data path: myCore\/SelectedStack_0 to myCore\/myStacks \/ stack <1>.Cached_8

Cell: in-> out	Fanout	Gate Delay	NET delay	logical name (net name)
FDCE: C-> Q	6	0.591	0.701	myCore\/SelectedStack_0
LUT3_D: I2-> LO	1	0.648	0.132	myCore\/myStacks\/Selected_0_mux00001_1
LUT3: I2-> O	4	0.648	0.667	myCore\/myStacks \/ newCached_mux0000 <0>1
LUT2: I1-> O	1	0.643	0.000	myCore\/myStacks \/ Msub_newCached_sub0000_lut <0>
MUXCY: S-> O	1	0.632	0.000	myCore\/myStacks \/ Msub_newCached_sub0000_cy <0>
MUXCY: CI-> O	1	0.065	0.000	myCore\/myStacks \/ Msub_newCached_sub0000_cy <1>
MUXCY: CI-> O	1	0.065	0.000	myCore\/myStacks \/ Msub_newCached_sub0000_cy <2>
MUXCY: CI-> O	1	0.065	0.000	

myCore\myStacks\Msub_newCached_sub0000_cy <3>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Msub_newCached_sub0000_cy <4>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Msub_newCached_sub0000_cy <5>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Msub_newCached_sub0000_cy <6>

MUXCY: CI-> O 0 0.065 0.000
myCore\myStacks\Msub_newCached_sub0000_cy <7>

XORCY: CI-> O 16 0.844
1.066myCore\myStacks\Msub_newCached_sub0000_xor <8>

LUT3_D: I2-> O 3 0.648 0.611 myCore\myStacks\newCached_mux0001 <0>1

LUT4: I1-> O 13 0.643 0.986 myCore\myStacks\incrCached_1_mux00031

LUT4: I3-> O 1 0.648 0.500 myCore\myStacks\incrCached_0_mux0003

LUT2: I1-> O 1 0.643
0.000myCore\myStacks\Madd_newCached_add0000_lut <0>

MUXCY: S-> O 1 0.632 0.000
myCore\myStacks\Madd_newCached_add0000_cy <0>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <1>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <2>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <3>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <4>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <5>

MUXCY: CI-> O 1 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <6>

MUXCY: CI-> O 0 0.065 0.000
myCore\myStacks\Madd_newCached_add0000_cy <7>

XORCY: CI-> O 2 0.844
0.000myCore\myStacks\Madd_newCached_add0000_xor <8>

FDCE: D 0.252 myCore\myStacks\stack <1>.Cached_8

Total 13 889ns (9 226ns logic, 4 663ns route)

(66.4% logic, 33.6% route)

1.1.2 FORTHSPM

SElected device: 3s700afg484-4

Number of slices: 3975 out of 5888 67%

Number of slice flip flops: 2871 out of 11776 24%

Number of 4 input LUTs: 7318 out of 11776 62%
Number used as logic: 7169
Number used as shift register: 85
Number used as RAMs: 64
Number of IOs: 73
Number of bonded IOBs: 73 out of 372 19%
IOB flip FLoPS: 1
Number of BRAMs: 11 out of 20 55%
Number of MULT18X18SIOs: 16 out of 20 80%
Number of GCLKs: 5 out of 24 20%
Number of DCMs: 2 out of 8 25%

Data path: myCore\myStacks\fast_1 to myCore\myStacks\BufferedInput_19

Cell:	in->	out	fanout	Gate delay	NET delay	logical name (net name)
FDCE:	C	Q	13	0.591->	1.015	myCore\myStacks\fast_1 (myCore\myStacks\fast_1)
LUT3_D:	I2	O->	26	0.648	1.292	myCore\myStacks \ Sideâ <0>base1.11_1.11.2 (myCore\myStacks \ Sideâ <0>11)
LUT3:	I2	O->	12	0.648	1.041	myCore\myStacks \ Sideâ <18>1 (myCore\myStacks \ Sideâ <18>)
LUT4:	I1	O->	1	0.643	0.000	myCore\myStacks\myALU \ res_and0000_wg_lut 1 (myCore\myStacks\myALU \ res_and0000_wg_lut <1>)
MUXCY:	S	O->	1	-0.632	0.000	myCore\myStacks\myALU \ res_and0000_wg_cy 1 (myCore\myStacks\myALU \ res_and0000_wg_cy <1>)
MUXCY:	CI	O->	1	0.065	0.000	myCore\myStacks\myALU \ res_and0000_wg_cy <2> (myCore\myStacks\myALU \ res_and0000_wg_cy <2>)
MUXCY:	CI	O->	1	0.065	0.000	myCore\myStacks\myALU \ res_and0000_wg_cy <3> (myCore\myStacks\myALU \ res_and0000_wg_cy <3>)
MUXCY:	CI	O->	1	0.065	0.000	myCore\myStacks\myALU \ res_and0000_wg_cy <4> (myCore\myStacks\myALU \ res_and0000_wg_cy <4>)
MUXCY:	CI	O->	1	0.065	0.000	myCore\myStacks\myALU \ res_and0000_wg_cy <5> (myCore\myStacks\myALU \ res_and0000_wg_cy <5>)
MUXCY:	CI	O->	1	0.065	0.000	myCore\myStacks\myALU \ res_and0000_wg_cy <6> (myCore\myStacks\myALU \ res_and0000_wg_cy <6>)
MUXCY:	CI	O->	2	0.269	0.450	myCore\myStacks\myALU \ res_and0000_wg_cy <7> (myCore\myStacks\myALU\res_and0000)
LUT4:	I3	O->	1	0.648	0.423	myCore\myStacks\myALU\Mmux_res_mux0000_5_f5 (myCore\myStacks\myALU\Mmux_res_mux0000_5_f5)
LUT4:	I3	O->	32	0.648	1,265	myCore\myStacks\myALU \ result <1><0>

1.1.2 FORTHSPM

(myCore\/myStacks\/myALU \/ result <1><0>)
LUT4: I3 O-> 1 0.648 0.000 myCore\/myStacks\/myALU\/Mmux_AluResult_910
(myCore\/myStacks\/myALU\/Mmux_AluResult_910)
MUXF5: I0 O-> 1 0.276 0.000 myCore\/myStacks\/myALU\/Mmux_AluResult_7_f5_9
(myCore\/myStacks\/myALU\/Mmux_AluResult_7_f510)
MUXF6: I0 O-> 1 0.291 0.563 myCore\/myStacks\/myALU\/Mmux_AluResult_5_f6_9
(myCore\/myStacks\/myALU\/Mmux_AluResult_5_f610)
LUT4: I0 O-> 1 0.648 0.000 myCore\/myStacks \/ BufferedInput_mux0001 <19>113_G
(N3678)
MUXF5: I1 O-> 1 0.276 0.423 myCore\/myStacks \/ BufferedInput_mux0001 <19>-113
(myCore\/myStacks \/ BufferedInput_mux0001 <19>-113)
LUT4: I3 O-> 1 0.648 0.000 myCore\/myStacks \/ BufferedInput_mux0001 <19>-148
(myCore\/myStacks \/ BufferedInput_mux0001 <19>)
FDCE: D 0.252 myCore\/myStacks\/BufferedInput_19

Total 14 564ns (logic 8 091ns, 6 473ns route)
(55.6% logic, 44.4% route)

1.1.3 FORTHSPC

Selected device: 3s700afg484-4

Number of slices:	4798 out of 5888	81%
Number of slice flip flops:	3657 out of 11776	31%
Number of 4 input LUTs:	9094 out of 11776	77%
Number used as logic:	8945	
Number used as shift register:	85	
Number used as RAMs:	64	
Number of IOs:	73	
Number of bonded IOBs:	73 out of 372	19%
IOB flip flops:	1	
Number of BRAMs:	14 out of 20	70%
Number of GCLKs:	4 out of 24	16%
Number of DCMs:	2 out of 8	25%

Data path: myCore\/Func_2 to myCore\/myStacks\/newTarget_0

Cell: in-> out	Fanout	Gate Delay	NET delay	logical name (net name)

```
    FDCE: C-> Q           3   0.591   0.674 myCore\/Func_2 (myCore\/Func_2)
    LUT3: I0-> O          1   0.648   0.000 myCore\/myStacks \/ newcore_mux0001 <0>
    MUXF5: I0-> O         5   0.276   0.665 myCore\/myStacks \/ newcore_mux0001 <0>
    LUT3_D: I2-> O       15   0.648   1,049 myCore\/myStacks \/ newcore_mux0000 <1>
    LUT3: I2-> O          1   0.648   0.000 myCore\/myStacks \/ newCached_mux0001 <0>
    MUXF5: I0-> O         3   0.276   0.563 myCore\/myStacks \/ newCached_mux0001 <0>
          LUT3: I2-> O                    1        0.648        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_lut <0>
          MUXCY: S-> O                    1        0.632        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <0>
          MUXCY: CI-> O                   1        0.065        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <1>
          MUXCY: CI-> O                   1        0.065        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <2>
          MUXCY: CI-> O                   1        0.065        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <3>
          MUXCY: CI-> O                   1        0.065        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <4>
          MUXCY: CI-> O                   1        0.065        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <5>
          MUXCY: CI-> O                   0        0.065        0.000
myCore\/myStacks \/ Msub_newCached_sub0000_cy <6>
          XORCY: CI-> O                  21        0.844        1,131
myCore\/myStacks \/ Msub_newCached_sub0000_xor <7>
    LUT4_D: I3-> O        3   0.648   0.534 myCore\/myStacks \/ newCached_mux0002 <7>1
    LUT4_L: I3-> LO       1   0.648   0.132 myCore\/myStacks\/incrCached_1_mux00031_SW3
    LUT4: I2-> O         94   0.648   1,285 myCore\/myStacks\/newSave_mux00021
    LUT4: I3-> O         25   0.648   1,260 myCore\/myStacks\/newTarget_and00001
    FDE: CE                   0.312         myCore\/myStacks\/newTarget_0
    ----------------------------------------
    Total                15 798ns (8 505ns logic 7 293ns route)
                             (53.8% logic, 46.2% route)
```

1.1.3 FORTHSPMC

Selected device: 3s700afg484-4

Number of slices: 5261 out of 5888 89%

1.1.3 FORTHSPMC

Number of slice flip flops: 4097 out of 11776 34%
Number of 4 input LUTs: 9415 out of 11776 79%
Number used as logic: 9010
Number used as shift register: 85
Number used as RAMs: 320
Number of IOs: 73
Number of bonded IOBs: 73 out of 372 19%
IOB flip flops: 1
Number of BRAMs: 14 out of 20 70%
Number of MULT18X18SIOs: 16 out of 20 80%
Number of GCLKs: 5 out of 24 20%
Number of DCMs: 2 out of 8 25%

Data path: myCore\/SelectedStack_0 to myCore\/myStacks\/ core <1>.Stack <0>.Top_6

Gate NET
Cell: in-> out fanout delay delay logical name (net name)
---------- ------------
FDCE: C Q 10 0.591-> 1,025 myCore\/SelectedStack_0 (myCore\/SelectedStack_0)
LUT4: I0-> O 1 0.648 0.000 myCo Re\/myStaCKS\/ReloadState_and000121_F (N2251)
MUXF5: I0 O-> 6 0.276 0.701 myCore\/myStacks\/ReloadState_and000121
(myCore\/myStacks\/N169)
LUT4_D: I2 O-> 42 0.648 1,297 myCore\/myStacks\/ReloadState_and0007
(myCore\/myStacks\/ReloadState_and0007)
LUT4: I2 O-> 1 0.648 0.423 myCore\/myStacks\/ newCached_mux0001 1 21
(myCore\/myStacks\/ newCached_mux0001 <1>21)
LUT4: I3 O-> 8 0.648 0.760 myCore\/myStacks\/ newCached_mux0001 <1>36
(myCore\/myStacks\/ newCached_mux0001 <1>)
LUT4: I3 O-> 1 0.648 0.452 myCore\/myStacks\/newCached_cmp_gt0000214
(myCore\/myStacks\/newCached_cmp_gt0000214)
LUT4: I2 O-> 19 0.648 1,088 myCore\/myStacks\/newCached_cmp_gt0000247
(myCore\/myStacks\/newCached_cmp_gt0000)
LUT4: I3 O-> 3 0.648 0.534 myCore\/myStacks\/ newToPop_mux0001 <1>1
(myCore\/myStacks\/N1111)
LUT4: I3 O-> 1 0.648 0.000 myCore\/myStacks\/newTop_and0000_f5_F (N2253)
MUXF5: I0 O-> 5 0.276 0.713 myCore\/myStacks\/newTop_and0000_f5
(myCore\/myStacks\/newTop_and0000)
LUT2: I1 O-> 1 0.643 0.000 myCore\/myStacks\/ Msub_newTop_addsub0000_lut <2>
(myCore\/myStacks\/ Msub_newTop_addsub0000_lut <2>)
MUXCY: S O-> 1-0.632 0.000 myCore\/myStacks\/ Msub_newTop_addsub0000_cy <2>
(myCore\/myStacks\/ Msub_newTop_addsub0000_cy <2>)
MUXCY: CI O-> 1 0.065 0.000 myCore\/myStacks\/Msub_newTop_addsub0000_cy <3
(myCore\/myStacks\/Msub_newTop_addsub0000_cy <3>)
MUXCY: CI O-> 1 0.065 0.000 myCore\/myStacks\/ Msub_newTop_addsub0000_cy <4>
(myCore\/myStacks\/ Msub_newTop_addsub0000_cy <4>)
MUXCY: CI-> O 0 0.065 0.000 myCore\/myStacks\/ Msub_newTop_addsub0000_cy <5>
(myCore\/myStacks\/ Msub_newTop_addsub0000_cy <5>)
XORCY: CI O-> 1 0.844 0.423 myCore\/myStacks\/ Msub_newTop_addsub0000_xor <6>

(myCore\myStacks\ newTop_addsub0000 <6>)
LUT4: I3 O-> 8 0.648 0.000 myCore\myStacks\ newTop_mux0004 <6>51
(myCore\myStacks\ newTop_mux0004 <6>)
FDPE: D 0.252 myCore\myStacks\ core <1>.Stack <0>.Top_6

Total 16 957ns (9 541ns logic 7 416ns route)
(56.3% logic, 43.7% route)

The footprint is, due the expensive stack management, high and varies between 441 k-gate and 623 k-gate. The maximum frequencies are satisfactory, given that all - comply conditions quite same as MicroBlaze. Stands a hard supply voltage in the range available can also overclocked be. In the test – also for several hours - which was not a problem and led to No crash...

The critical paths to go as hoped for, at least in the version with a core and a multiplier, not by the stack management, but rather, as in register machines by the ALU. It is worth noting that in the critical path not the Is the multiplier, but the shift register. An improvement of the critical path with the Targetbne Speed increase can only by Optimization the Stack management (indexing) be reached, bring additional operation steps no further improvement.

The number of floating-point operations per second is, due to its realization as Software, not intoxicating - compared to a PC where achieved several MIPS be - but for the use of the processor to measuring, evaluating, Ste ueRNund Fix quite enough. The use of a multiplier, as well as the realized Floating-point division as multiplication by the reciprocal, increases the number of Whetstones clear.

2 Software

To the user interface in the client no results will be presented, it was tested only on their correct function. For demonstration of the system were following applications, table 77, created.

Application	
Whetstone.FS	whetstone benchmark in FORTH
Bubble.FS	Demo of a bubble sort in FORTH
Hanoi.txt	Demo of the towers of Hanoi in FORTH
Switch.txt	Demo, shift from core to core 1 0 and again back
Sort.Java	Demo of bubble sort and quicksort
Equation.Java	demo for solving linear equation systems

Table 77: Applications to the test and demonstration

2.1 BIOS

All functions and services are tested and work properly. Following tests, Table 78, were carried out.

Programs	
Testall.txt	Tests all arithmetic functions and the file operations of BIOS
Memorytest.FS	tests the dynamic memory and its basic operations

Table 78: test programs for the BIOS

The arithmetic functions in the BIOS provide the same values as the corresponding The class Math Java functions on a PC. Compared were the first 16 actually significant digits of the results, da ceil(53_*_log_(2)) == 16 applies. The Results vary more than in the last bit of the mantissa. Details can not be said, since the failure is also due to the different conversion of Output functions can be caused before the showing. The error is not more than 2 at the last point of the converted string.

2.2 Assembler

ArgumentException: The incoming token has expired. Get a new access token from the Authorization Server. : I
ArgumentException: The incoming token has expired. Get a new access token from the Authorization Server. : I
ArgumentException: The incoming token has expired. Get a new access token from the Authorization Server. : I
Command to add and initialize it with its operation code. If it is to a command with immediate operands is, needs a separate case in which Case distinction usually statements inserted by Parser.java, otherwise

are there any modifications required. In appendCommand of Code.java je a case must in the two case distinctions to be entered. In the first command is a Is command group associated - only for statistic purposes-, in the second he is in accordance with the \"Number of results (either 0 or 1)\" - \"number of consumed stack values\" (either 0, 1 or 2) \"classified.\" This completes the extension.

2.3 Java

The function of the compiler was just sort of applications with the equation checks, but not fully tested. It all control structures (for that) but seem While, do-while, continue, break, switch, and try - catch), the creation and deletion of Scopes, and reference counting to work. The reason for this short Testing is located on the long charging time of an application. It consists not only of the Source files alone, it must rather all files of the referenced classes loaded are, what required a lot of time. The execution of the application, however, it is gratifying fast. The long load time is mainly on the serial interface lead to back, a USB port would bring a substantial improvement and testing of Compiler easier.

A Java application is loaded with the menu load Java modules and initialize *Java modules called and not through a command line in the command window! This must* a final static block inserted into the main class, the calling carries out and provides the parameters. When initialized, the application is then running. This block is for calling different parameters to modify and to translate the application. But no longer need to be charged, it can immediately with the menu initialize Java modules will be restarted.

The base classes (object, String, etc.) an application can aus dem Source package *J2SDK-1_4_2-src-scsl.zip Sun are transferred, but mostly restated, in particular* special need to out be programmed all native methods, reflection (Class.java) is not supported and must be removed from Sun acquired sources. Some classes - they are sort or are already partly available equation - adapted, class Math entirely.

3 Final views

Currently, the system is as a basis for micro-controller to be regarded as prototype, is to see a wide range of applications. It can have the Hirose plug. various devices and hardware are connected, USB, ADC-, and DAC Modules are already available on the Xilinx Kit starter. You need only to the Processor to be connected. The system is used for which tasks, If anything, it is completely open.

3.1 Suggestions

3.1.1 The processor

The prefetch queue program could be revised since, in the versions with 4 Cores of the bottleneck is.

3.1.2 The memory in the BIOS

First, the method of memory allocation can be changed,. under keep the Administration with reference counting. Secondly, reference can *Counting change for the benefit of another method, as mark and sweep,.* To modifications to the processor could be necessary but, approximately an additional bit in the Stack, which marked a reference in a local variable. The step mark on the Hike through the local variable, living memory blocks are detected it and marked as alive. Another variant would be a third stack for local variables only the references contain.

This does not mean that special hardware is necessary, but should consider you already.

3.1.3 Of the client

A USB port would be desirable, since the serial port but right is slow. Unfortunately no library that provides Sun, these would have to be purchased. Also the USB module should be connected to the server.

3.1.4 Java

Is discarded reference counting in favor of another method, the following shall To make changes to the compiler:

Pass.java, method ClassBody is in the production of ~ species to stop, i.e. The if block after the comment line \"\/ \/ default destruktor\" is to delete. In Code.java are the methods freeVariable, incrementReference, and their calls to delete.

If necessary source classes from Sun should - what sure the case will be - for the system be adapted.

Extensions

1 Garbage collector tricolor marking

This method - source file \"onthefly.fs\" - Alternatively, to reference counting
be used.

The method is a modification of the method of mark and sweep. Mark is in the State
a visited block not immediately as essential (color: black) marked, but as
essential, but not yet evaluated (grey color) marked. Only when all its
hidden references were visited and are grey marked with color, its
Color: black. Also applies: an allozierter block has the color white, has a free block
no color. At the end of mark, all essential occupied blocks are no longer white.
In the step of sweep, the white blocks are released and bland, and the non-white
Blocks white. The garbage collection is thus complete. At the beginning of step
Mark are all entries in the stack of the reference variable (VBLOCK) and all entries
grey dyed in the cyclic store of the static reference variable (SBLOCK), unless
the entries are non-zero. Then follows a loop in which all grey blocks abgearbei tet
wground. YouwIRD only leave if no grey block longer exists, then
all essential blocks processed and black marks. Tricolor marking is in the
Opposed to mark and sweep unterbrechbar. One is from the garbage collector
Condition variable VBARRIER to true set when he starts the cleanup and am
End of step sweep to false set. The management routines, table 1, of
Store of the reference variables require this information.

Function	Importance
VALLOCATE	n - drobnicy Is n == 0, an address is adr expiration from the cyclic store given. N is > 0, n be blank entries on the stack of the reference variable is filed and as adr allocated the address of the first Entry is returned. Is n greater than the free capacity of the stack, 0 will be returned
SETVTOP	drobnicy- Refers the new adr will stack the reference variable, in the COR Top of the stack
V!	adr - handle Is adr == 0, will handle in the Refernzvariable with the index 0 in the Stack posted the reference variable. Otherwise in by adr identified location. Is set VBARRIER and handle < > 0,. is the color of handle on grey set (the powerful invariant of) (tricolor marking).
MARK	Every millisecond is checked, whether MACCU greater than a quarter of the Dynamic memory size is. If so, he is garbage collector MARK

Table 79: The garbage collection routines

The storage management provides a battery MACCU, the at the Allocate of a block of (MALLOC) accumulates the requested block sizes. Of the garbage collector is again reset this variable.

1.1 Times for routines

The following abbreviations apply:

- r the size of the cyclic store the apels Reference Varia riable in words
- h the number of applied handle
- n_i the number of hidden references of the block i
- x the number of iterations of step mark, here unknown

Also applies the assumption that a command in a bar is worked off.

The constants represent the counted out number of the commands of a sequence, the The number of iterations of a sequence represent variables.

1.1 Times for routines

The number of commands for the function VALLOC:

VALLOC(m) = 36 + m born 9

The number of commands to the function SETVTOP:

SETVTOP = 20

The number of commands for the procedure of V!:

V! = 32

The number of commands for the garbage collector MARK

MARK i

1.2 Dynamic memory (buddy method)

Alternatively to the memory management with the MS-DOS method - source file \"memorymana-\" \"gement.fs\" - can be used the buddy method - source file \"buddy.fs\" -. regardless of the garbage collector.

The buddy method requires a memory which length is a power of 2. At a block with the length of a request is pow (2, (int) ld(2_*_n_-_1)) returned,. so a blo CKMIt einIt includes the length of the smallest power of 2, the n. Each block with power, m has a equal big buddy. The relative address of the buddies in following relationship Adresse(budy0) = Adresse(buddy1) ^ pow(2,_m). Each Buddy contains in turn 2 buddies with a power of m-1, etc.

Standard functions are: UNUSED, modified standards are ALLOCATE, FREE,. RESIZE [2] Chapter 14.6.1, Page [87]. The Modification is it,. that the Storage management user a block of memory not directly makes available,. but a handle. Establishing handle one:

Word	meaning
0	The reference counter
1	Start address the Data range the preallocated Memory block

This solution allows only a garbage collection! Should an allocation request not immediately satisfied are can, the cleanup is automatically called. At least 2 bucks, which are only half as large as the required are free, which is trying Cleanup by copying to the buddies is one of the free blocks, one twice as to create large free block. Achieve this, the request can be fulfilled.

Java expects a memory management, which automatically detects when an occupied block can be actually released. This is a very old method that supports it, Counting the references [8, Chapter 2.1, page 19-25], a task by the Application must - be done the memory management but must support. In short, a block is only in the list of free blocks transferred when be Reference count reaches zero, otherwise the counter is simply humiliated. Advantage of this method is the distribution of administrative work, it is only

If it is really necessary. Other procedures require a
own periodic garbage collection process which will not be interrupted
may what is not desirable.

The dynamic memory is a closed block of memory via the services,
can be created, shared, and initializes. It is only a dynamic storage
possible. In the first word of the block is the number of the created handle in the second word
is the head of the list of free blocks. As of the third word die
Konsekutive list der handle, which can grow any. The rest of the block disintegrates in
documented, non-contiguous areas that handle that are accessible. The
blank areas are organized in a doubly linked list for each power of
2 your own. A free block has the following structure:

Word	use
0	Address of the next free block
1	Address the previous free Block
2	Power of the length of the block
3 Unused	

An occupied block:

Word	use
0	Power of the length the block
1	Address the Handle
From	2 data area

1.2.1 Allocation

First is a free handle searched or angelegt and a best matching free
Block searched. None exists is searched a garbage collection again,
aborts on failure. Otherwise, the block is from chained, adapted,
initialized, the List the free Blocks modified and the Address the handle
returned.

1.2.2 Free

The counter will be humbled, and only if it is 0, the list blocks free, to
possible merger with neighbouring countries, registered. The handle is not
released, can be reused but.

1.2.3 Resize

The data area of the block is copied to the heap, and go-ahead given for the block. A new block of the desired size is allocated, the data field of the current
Copied blocks from the heap, and the old handle, which now contains the new data range, returned.

Additional commands:

Command	Importance
ALLOCATED SIZE	handle - size Returns the size of the data area, without administrative data,
HANDLEVALID	handle - flag If it is a valid address, - 1, otherwise 0 is returned
INCREFERENCE	handle- increments the reference counter

1.2.4 Times for INCREFERENCE, ALLOCATE, and DECREFERENCE

The following abbreviations apply:

- N the amount of dynamic memory in words
- h the number of applied handle
- n, i 3 < n, i < ld(N) + 1

Also applies the assumption that a command in a bar is worked off.

The constants represent the counted out number of the commands of a sequence, the The number of iterations of a sequence represent variables.

The number of commands for the hidden procedure to the a free block from chains:

UNCHAINFREEBLOCK = 24

The number of commands for the hidden procedure to the hinged chains of a free block:

CHAINFREEBLOCK = 37

The number of commands for the hidden procedure to the documents of a free block:

OCCUPYBLOCK (n) = 41 + n born (24 + CHAINFREEBLOCK)

The number of commands for the hidden procedure to move a buddy:

MOVEBUDDY (n) = 13 + n born (51 + UNCHAINFREEBLOCK + 63 + SEARCHLAST(i) +) OCCUPYBLOCK(i)) \/ 2

The number of commands for the hidden procedure for memory compaction:

COMPACT POOL = 50° + UNCHAINFREEBLOCK + MOVEBUDDY + CHAINFREEBLOCK

The number of commands for the hidden function to the Handlesuche:

$GETFREEHANDLE_{usually}$ = 12 + 13 b. h

$GETFREEHANDLE_{worst}$ = 12 + 13 born h + MOVEBUDDY(ld(h)) + 19

The number of commands for the hidden function to determine of the last best free Blocks:

SEARCHLAST (n) = 24 + (ld(N) - n + 1) born 12

The number of commands for the hidden function to determine of the best free block:

$BESTFREEBLOCK_{usually}$ = 26 + SEARCHLAST

$BESTFREEBLOCK_{worst}$ = 26 + SEARCHLAST + COMPACT POOL + SEARCHLAST

Die Number of commands for the function MALLOC:

MALLOC (n) = 59 + GETFREEHANDLE + BESTFREEBLOCK + (n + 3) \/ 4 born 12

The number of commands to the function DECREFERENCE:

$DECREFERENCE_{Decrement}$ = 48

$DECREFERENCE_{destrOy}$ = 48 + 53 + 48 born n, with n = ld (block size)

The number of commands for the procedure INCREFERENCE:

INCREFERENCE = 40

1.2.5 Final remark

The implemented Memory allocation is sure not optimal redress ich the Memory usage, but quickly. The buddy method can but easily through a other memory allocation will be replaced. The implementation starts with MOVEBLOCK-UP including and ends at UNUSED exclusively and is 1186 bytes of BIOS a. Just under 2 kB in Rome are available for a different implementation, longer she may not be.

Annex

1 Used resources

Hardware:

- Xilinx, Spartan 3A Kit: for the realization of the processor
- Digitus USB 2.0 to RS232 adapter: serial interface for a notebook
- A nichtausgekreutztes cable with 9-pin plug and socket: Connects notebook with the hardware
- Notebook with Windows XP

Software:

- Xilinx ISE Web Pack 9.2: development environment for the processor, free software
- ModelSim Simulator 6.1e: to test your VHDL code
- JBuilder 2005: development of the client
- javacomm20-win32.zip: Java interface for serial, free software from Sun
- Open Office: for documentation, free software
- J2SDK-1_4_2-src-scsl.zip: Source files of the Java base classes from Sun, free software

2 Description of the adaptation of the RAM controller from Xilinx

The adaptation in vhdl_syn_bl4_parameters_0.vhd:

Eifügen of

```
use work.global.all;
```

and Modfikation of

```
constant reset_active_low: std_logic: = '1';
constant rfc_count_value: std_logic_vector(5_downto_0): =.
std_logic_vector (to_unsigned (integer(60.0e-9_*_real(theClock)_+_0.49), 6));
constant max_ref_cnt: std_logic_vector(10_downto_0): =.
std_logic_vector(to_unsigned(integer(real(theClock)*7.6e-6),max_ref_width));
```

The Adaptation by *vhdl_syn_bl4_infrastructure.vhd* by Modification (bold) (highlighted) immediately follow the begin keyword up including the first *process - block:*

```
  sys_clk_ibuf &lt; = SYS_CLK;
  -lvds_clk_input: IBUFG port map ()
  -I => SYS_CLK,
  -O => sys_clk_ibuf
  --  );
  clk_int_val &lt; = clk_int;
  clk90_int_val &lt; = clk90_int;
  sys_rst_val &lt; = sys_rst;
  sys_rst90_val &lt; = sys_rst90;
  sys_rst180_val &lt; = sys_rst180;
  delay_sel_val1_val &lt; = delay_sel_val1;



-To remove delta delays in the clock signal observed during simulation
-Following signal are used


  clk_int_val1 &lt; = clk_int;
  clk90_int_val1 &lt; = clk90_int;
  clk_int_val2 &lt; = clk_int_val1;
  clk90_int_val2 &lt; = clk90_int_val1;
  user_rst &lt; = not reset_in_n when reset_active_low = '1' else reset_in_n;
  user_cal_rst &lt; = reset_in_n when reset_active_low = '1' else not
reset_in_n;



  process(clk_int_val2)
```

```
begin
 If clk_int_val2 'event and clk_int_val2 = ' 1' then
  If user_rst = '1' or dcm_lock = '0' then
     wait_200us_i &lt; = '1';
     Counter200 &lt; = (others => '0');
  else
      If (Counter200 &lt; integer(real(theClock) born 200. 0e-6 + 0.5)) then
         wait_200us_i &lt; = '1';
         Counter200 &lt; = Counter200 + 1;
      else
         Counter200 &lt; = Counter200;
         wait_200us_i &lt; = '0';
     end if;
  end if;
 end if;
end process;
```

The adaptation of vhdl_syn_bl4_controller_0.vhd by the inclusion of

```
use work.global.all;
```

The Adaptation by *vhdl_syn_bl4.vhd* by Remove the Component *vhdl_syn_bl4_main_0, the test engine.*

References

[1] Koopman, p. (1989) stack computer - the new wave, Ellis Horwood

[2] ANSI, 1994 DPANS'94 - programming languages - FORTH, ANSI, 24 March 1994

[3] R. (1985) Zech, the programming language FORTH, Franzis Verlag

[4] FIG, http:\/\/www.forth.org\/, FORTH interest group (present August 2008)

[5] Knuth, Donald E. (2005) the art of computer programming vol. 1 MMIX, Addison-Wesley

[6] RIIC, VLSI design lecture script, Institute for integrated circuits, Johannes Kepler

Univ.

[7] Lindholm, t., Yellin, F. The Java virtual machine specification Second Edition, Addison-

Wesley

[8] Jones, R., Lins, R. (1996) garbage collection, John Wiley

[9] Dunn, D. (2002) Java rules, Addison-Wesley

[10] Gosling, j., joy, B., Steele, D., Bracha, G. (2005): the Java language specification Third

Edition, Addison-Wesley

[11] SSW, http:\/\/www.ssw.uni-linz.ac.at\/Coco\/Java\/JavaGrammar.html, Institute for system

software, Johannes Kepler University (present August 2008)

[12] Kahan, W. (1997) http:\/\/www.cs.berkeley.edu\/~wkahan\/ieee754status\/IEEE754.PDF

Berkley (present August 2008)

[13] Abramowitz, M., I. (1965) and Stegun Handbook of mathematical functions, Dover

Publications

[14] Aho, Hopcroft, Ullman, (1974) the design and analysis of computer algorithms, Addison-

Wesley

[15] Trenz, http:\/\/www.trenz-electronic.de\/home\/indexde.htm, FPGA modules Development

boards (present August 2008)

[16] Xilinx http:\/\/www.xilinx.com ISE Web Pack, development boards (present August) (2008)

[17] Forth Inc., http:\/\/www.forth.com\/resources\/evolution\/index.html, history of FORTH

(present August 2008)

References

[18] O'Connor, Derek, 2006, http:\\www.derekoconnor.net\NA\Notes\RecSqRoot.pdf, RECIPROCAL

[19] Wikipedia, http:\\de.wikipedia.org\wiki\Whetstone, Whetstone benchmark (present August 2008)

[20] JavaDoc, http:\\java.sun.com\j2se\1.4.2\docs\api\, (present August 2008)

Curriculum vitae

Gerhard Hohner
Ding court first race 63
4020 Linz

Personal data:	born on December 20, 1956 in Linz, Upper Austria, Austria
	an Austrian citizen, röm. Kath.
Family:	Herbert Hohner, locksmith (deceased)
	Paula Hohner, housewife
	1 Brother (50 years)
Professional experience:	1963-1967 Elementary school in Linz
	1967-1975 Gymnasium in Linz
	June 1975 graduated with Matura
	1975-1981 Studying computer science in Linz, without a degree
	1981-1986 of Freelancer at the Institute for
	Experimental physics 2
	1987 Performance of military service
	1987-1990 Programmers at the company Plasser
	1990-1995 inactive
	From 1996 pensioner
	1996 Recovery of studies
	2004 Start of the thesis
	2005 Completion of first phase of the study
	2008 likely graduation

Affidavit of

I declare this affidavit, that I present work independently and without assistance written, others not used as the given sources and which sources used which literally or content taken from bodies as such did identify.

Linz, at

_____

Name