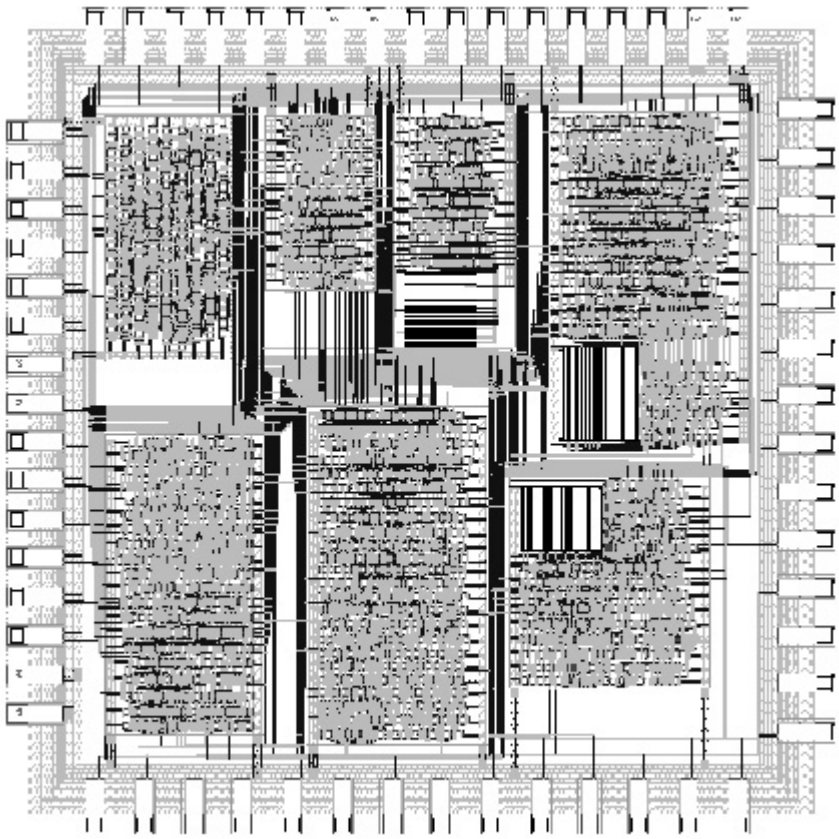
RTU94
Real Time Unit 1994
-Reference Manual-



Contact Person: Johan Furunäs, Mälardalens University, P.O. Box 883, S- 721 23 Västerås, Sweden ,
e-mail: johan.furunas@mdh.se.
Authors: Joakim Adomat, Johan Furunäs, Johan Stärner and Lennart Lindh.
(ver 1.7) 960205

Preface

The RTU94 reference manual describes how to build a realtime system with RTU94.

This manual contains the following chapters and appendices:
- Chapter 1, *Introduction*, provides a general overview of RTU94.

- Chapter 2, *Communication*, describes the registers on RTU94 and how they are used in communication between a cpu and the RTU94.

- Chapter 3, *Service calls*, describes the services RTU94 supports and how they are used and which instruction format they have.

-Chapter 4, *C-interface*, describes C functions avalible for RTU94.

-Chapter 5, *System configuration*, describes the configuration of a realtime system using the RTU94 and [FORCE] VME boards.

-Chapter 6, *Software development*, describes how to develop software on [FORCE] SYS68K CPU boards using Microtec Research 68K software development tools.

- Appendix A, *VME boards*, a short description of some [FORCE] VME boards, and how they should be configured, used in a realtime system based on RTU94.

-Appendix B, *Testprogram*, a testprogram is presented to test all services for one cpu with the RTU94.

-Appendix C, RTU94 pin assignment.

-Appendix D, *Performance analysis*, a performance comparcion between RTU94 and commercial RTKs (software based).

-Appendix E, *Prototype*, a prototype board is described, which can be configured to work as a RTU94 on a VME board.

-Appendix F,a list of publications related to the RTU94 project.

-*References*
.

Contents

1 Introduction

Overview

The RTU94 is a realtime kernel on a chip for use with VME microprocessor boards. It can handle 1 to 3 processors, 64 tasks and 8 priority levels.
VHDL were used as design language, the advantage of using VHDL is that it's easy to change the functionallity of (in this case) the RTU94 e.g change the scheduling algorithm to static instead of priority preemtive (which is used now).
RTU94 has a VME interface, used for communicating with the microprocessors, and a scheduler and a set of services (which are presented in Chapter 3).

Since RTU94 schedule each task, no clock ticks are needed on the microprocessors. But each microprocessor must be able to handle one external interrupt (the taskswitch interrupt from RTU94) and be able to communicate over a VME bus. Three microprocessors can be served and tasks can be initiated to be scheduled to run on a specific cpu or whichever cpu that have time to execute it.
Advantages of using a hardware kernel:
Moveable

- The RAM which RTU94 use is implemented on the chip
- The instruction format doesn't change.
- If the cpu can communicate over a bus similar to e.g VME bus, a RTU can be connected to serve it.
- The only software segment, which must be rewritten on a new cpu type is the assembler code for task-switch.

Performance and determininism

- The performance increases and in some cases the time for RTU services can be neglected, except the taskswitch routine which is implemented in software. E.g scheduling of a task is done within 5 to 80 clock cycles (i.e 5 µ-seconds with a 16 Mhz clock, in the worst case) and furthermore the scheduling doesn´t load the cpu.
- Deterministisc execution of the service-instructions. The RTU takes care of the scheduling, queues, clocktick-administration . Therefore it's easier to calculate the execution time for a system. The cpu load is dependent on how many active tasks and in which state they are in, when a software kernel is used.
- No clock tick interrupts to the cpu are needed.
- The interrupt handler is implemented on the RTU, which means that external interrupts doesn´t interupt high priority running tasks.
- The response time decreases for all service calls, because the RTU is designed of parrallell hardware e.g scheduling, semafore handling etc. are performed in parallell on the RTU.

Software development

- Easier software development , because the code for the realtime kernel doesn't have to be executed by the cpu.
- Easier understandability for the system, because of separateing the realtime kernel from RAM.
- A more safe execution of the realtime kernel functions, because of the functions are designed in physical separated parts. And therefore no interference between the functions are possible.
- No interrupt handler for external interrupts (except the task switch interrupt) has to be implemented in software.
- It is easier to build multiprocessor systems, since synchronisation between cpu:s (using flags), load balancing and redundancy (using watch dogs) are supported by the RTU.
- It is easier to debug a RTU based realtime system whithout affecting it, since the service calls to it can be logged on the bus.

The motivation for using a RTU is almost the same as using a mathprocessor. A mathprossesor helps the cpu to get better perfomance and the RTU does the same. But a difference between a mathprocessor and a RTU, is that the RTU is more complex.

Example of a system based on RTU94:
The Real-Time system configuration is (see figure 1):
• One to three application processors.
These have contact with the VME bus and one interrupt each. The interrupt is used to start the task switch routine in the processor. Each processor has a local memory for local tasks
• The RTU94 is on a separate PCB board. The RTU94 chip contains a whole Real-Time kernel and VME bus interface. The only things needed outside the chip are some buffers and clock generators. There are twelve 16 bit registers and one 8 bit for data communication between application processes and the RTU94 .
• A global memory for global tasks.
The global tasks are dynamically scheduled on all three processors, the local tasks are locked to one processor and the local memory.
• The analyser processor is for graphical display of system status and operation to provide developers with better visibility into their applications. A PC is used for the graphical interface. The analyser processor is optional.

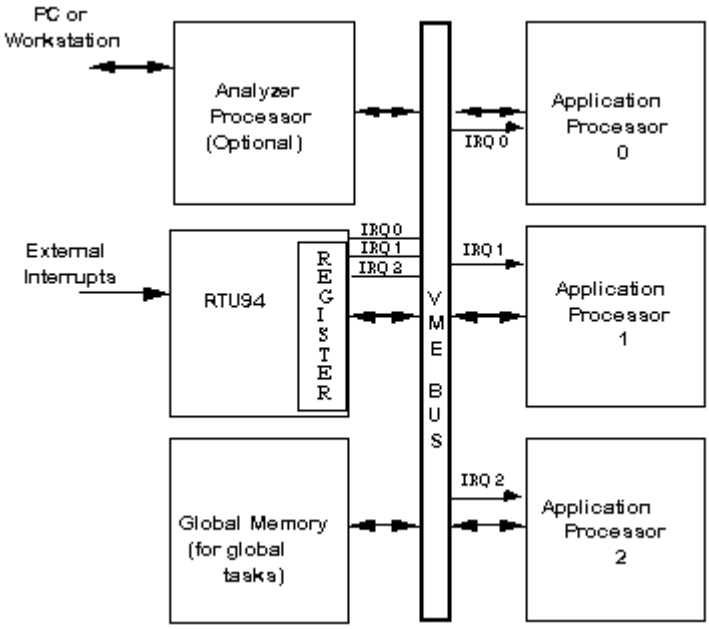


figure 1. System configuration

Internal structure of RTU94

The RTU consists of a number of units, which handles the different functions in the kernel (see figure 2). Functions within RTU94:

• **Scheduler**. The scheduling concept on RTU94 is priority-based, pre-emptive algorithm. The goal of the algorithm is to guarantee that the task which is executing on the processors at any point in time is the one with the highest priority among all tasks in the ready state.

A task can be executing, ready or waiting. There can only be one task executing on a CPU at a time, so if we have three CPU's in the system there can be three executing tasks.

The RTU94 has four different ready queues, one ready queue for each CPU. A task in any of these queues can only be executed on the corresponding CPU. The last ready queue consists of tasks that can be executed on any of the connected CPU's. The scheduler compares in parallel the own and common queue for each CPU, in order for the scheduler to fulfil its goal.

There are two events that can change the executing task. First, the task itself can request a task switch. Second, the scheduler can interrupt the executing task if there is a task with higher priority in the ready queue.

• **Delay**. Holds a task in the delay queue until the delay time has expired, then the task is send to the redy queue.

• **Periodic start**. Holds a task in the periodic queue until the period time has expired, then the task is send to the redy queue.

• **Watch-dog**. Holds a task until the watch dog is not kicked within the watchdog time, then the task is send to the ready queue.

• **Semafore**. Can hold four tasks in the semafore queue. The first task in the queue get the semafore when the semafore get released. When a task get the semafore it is send to the redy queue and the tasks in the queue moves one step ahead.

• **Flag**. Can hold four tasks in the event flag queue. When the flag get set, all tasks in the queue are send to the ready queue.

• **External interrupt**. Holds a task until an interrupt corresponding to the task's interrupt level occurs, then the task is send to the ready queue.

.

• **VME interface**. Is the I/O between the realtime functions and the VME bus. Consists of thirteen registers (see chapter 2).

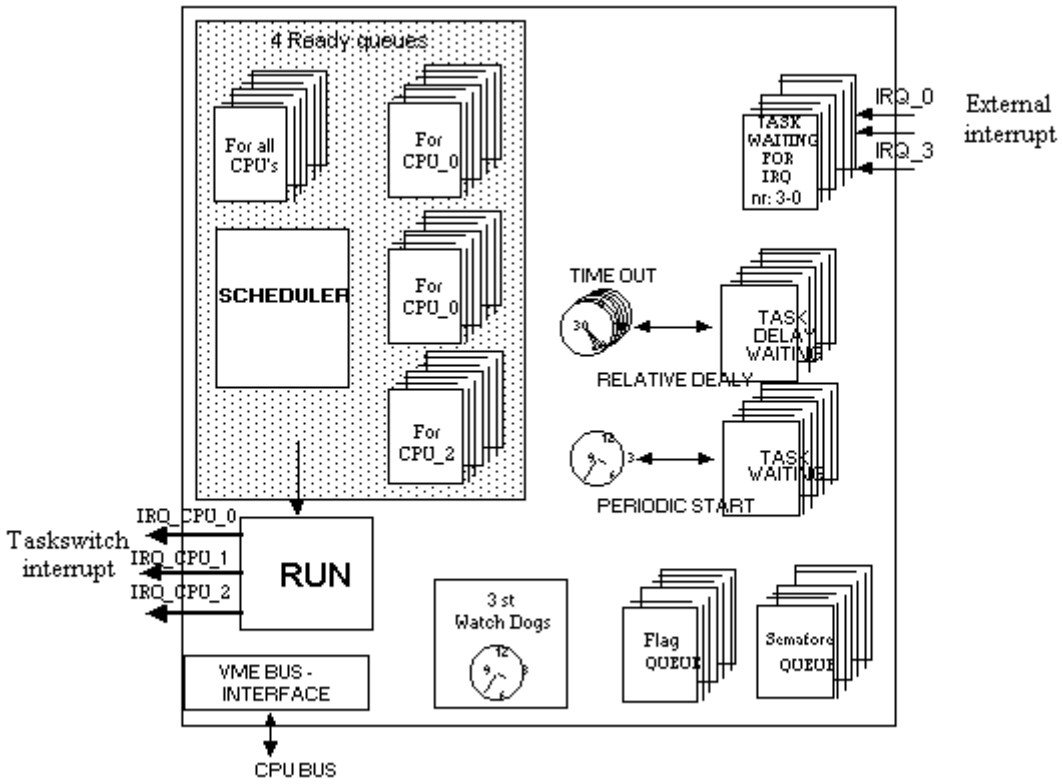


figure 2. Internal structure of RTU94

2 Communication

In this chapter the registers on RTU94 are first described and then the communication protocols to RTU94 are explained.

Register model of RTU94

Table 1 shows the register set in RTU94. Each CPU have three dedicated registers; cpu_status_register, next_task_id and cpu_control_register. There are two registers that holds the overall RTU information; rtu_status_register and rtu_control_register. Finally there are two service-call (SVC) registers; svc_instruction_register and svc_semaphore_register.

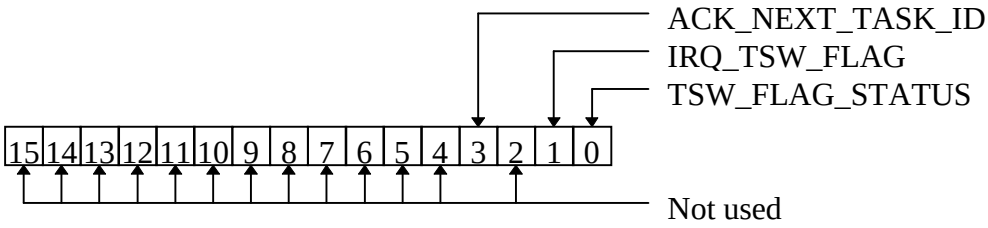
Register name	Address offset (hex)	Read / Write	Size
CPU_STATUS_REGISTER (0 to 2)	0, 2, 4	r	16 bit
RTU_STATUS_REGISTER	6	r	16 bit
RTU_CONTROL_REGISTER	8	r/w	16 bit
NEXT_TASK_ID (0 to 2)	A, C, E	r	16 bit
SVC_INSTRUCTION_REGISTER	10	w	16 bit
SVC_SEMNAPHORE_REGISTER	12	r/w	8 bit
CPU_CONTROL_REGISTER (0 to 2)	16, 18, 1A	r/w	16 bit

Table 1

The register address is calculated by adding a base address to the address offset given in table 1.

CPU_STATUS_REGISTER

There are three cpu_status_registers in RTU94, one for each CPU. It contains the status of respectively cpu.



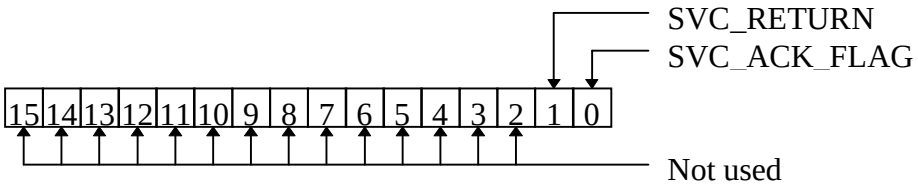
ACK_NEXT_TASK_ID: 0 = No acknowledge
1 = CALL_NEXT_ID acknowledge

IRQ_TSW_FLAG: 0 = No interrupt
1 = Interrupt signal (TSW request)

TSW_FLAG_STATUS: 0 = TSW on
1 = TSW off

RTU_STATUS_REGISTER

It contains the RTU status. Bit SVC_RETURN is affected when an event flag or a semaphore wants to be allocated. If it is set to 0 the allocation succesed else it is set to 1.

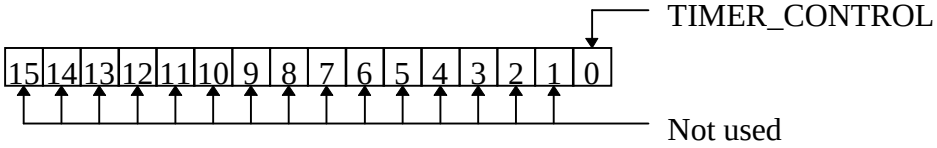


SVC_RETURN: SVC status. See description of service calls for further information.

SVC_ACK_FLAG: 0 = No acknowledge
1 = Service call acknowledge

RTU_CONTROL_REGISTER

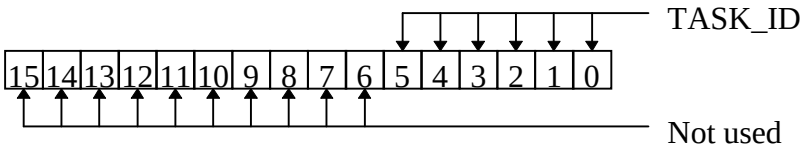
Controls the on chip timer clock, which is used by time dependent services e.g wait for next period.



TIMER_CONTROL: 0 = Timer clock enabled (on)
1 = Timer clock disabled (off)

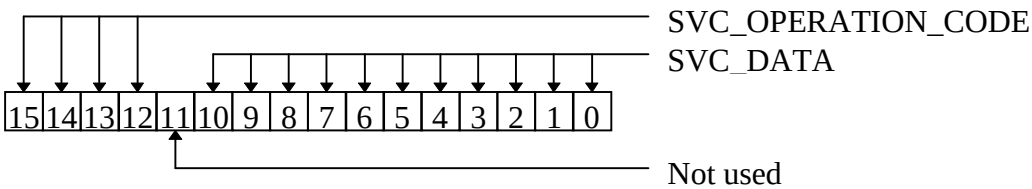
NEXT_TASK_ID

Holds information about the currently executing task-id. During taskswitch it is used to inform the CPU about the next task-id to be executed. There are three next_task_id registers in RTU94, one for each CPU.



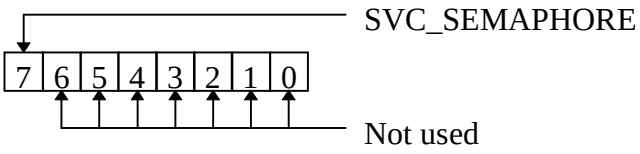
SVC_INSTRUCTION_REGISTER

The RTU perform the service, which are written to this register. Bit 12 to 15 are reserved for the service code and bit 0 to 10 are used for service call data (see chapter 3 for more details about service calls).



SVC_SEMAPHORE_REGISTER

This register is used to protect the svc_instruction_register in a multiprocessor enviroment. Any task that wants to use the svc_instruction_register must take the svc_semaphore. The semaphore is taken by writing 1 to bit 7.

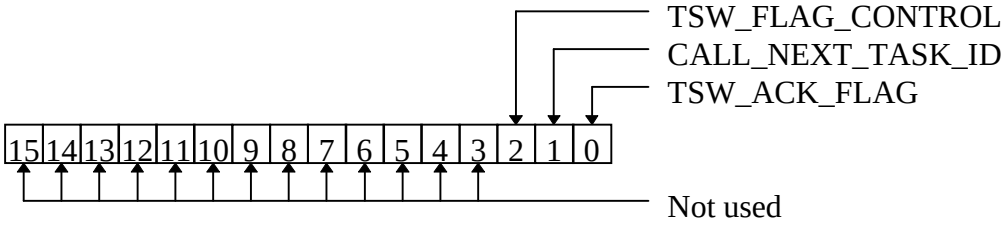


SVC_SEMAPHORE: 0 = SVC semaphore cleared
1 = SVC semaphore set

.

CPU_CONTROL_REGISTER

There are three cpu_control_registers in RTU94, one for each CPU.
Each register controls taskswitch on/off and acknowledge and also the call for a new task, for respectively cpu.



TSW_FLAG_CONTROL: 0 = Set TSW on
1 = Set TSW off
CALL_NEXT_TASK_ID: 0 = No request
1 = Request new task id
TSW_ACK_FLAG: 0 = No acknowledge
1 = TSW acknowledge

Communication protocols

The communication with the RTU94 is done in three different ways:

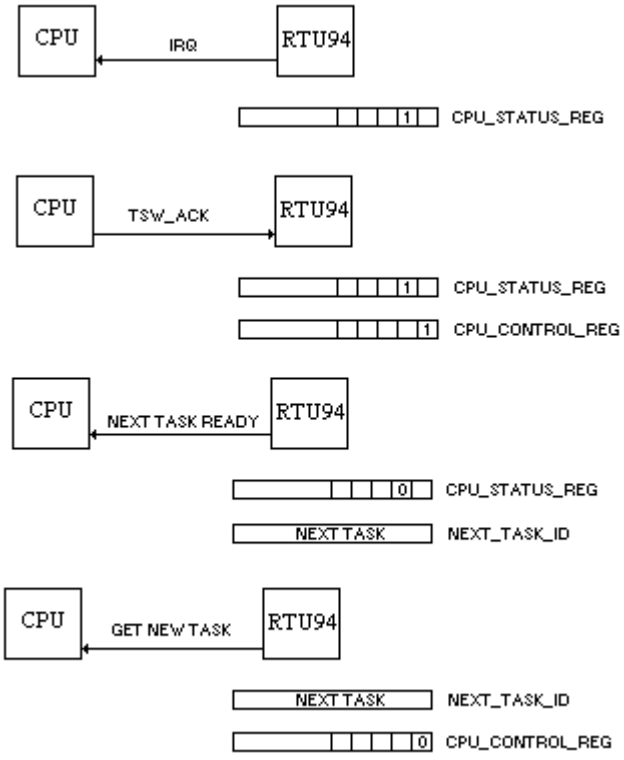
• **Task switch Irq**: The RTU activates the interrupt to the processor card to interrupt the running task and start the task switch routine.

• **Terminate task**: The service call terminates and activates the RTU94 to give next task id to run.

• **Service call**: A simple handshake protocol to get a safe communication between the CPU and the RTU94 when a service call is performed.

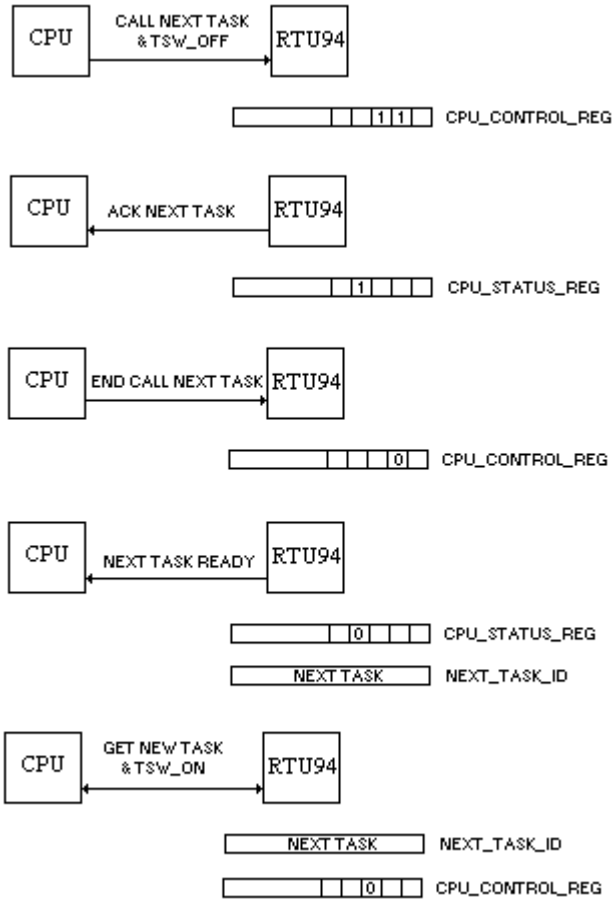
Taskswitch Irq protocol

Operation: 1 - RTU94 sets IRQ_TSW_FLAG on CPU_STATUS_REG and interrupt the cpu.
2 - The cpu must check if IRQ_TSW_FLAG is set (if it is not set ignore interrupt) and acknowledge the interrupt by setting bit TSW_ACK_FLAG on CPU_CONTROL_REG.
3 - RTU94 clear IRQ_TSW_FLAG, clear the interrupt and place next task id on NEXT_TASK_ID.
4 - The cpu must waits until IRQ_TSW_FLAG is cleared and then it gets new task id to switch to by reading NEXT_TASK_ID. After reading next task id the cpu must clear TSW_ACK_FLAG, to finish the taskswitch procedure.



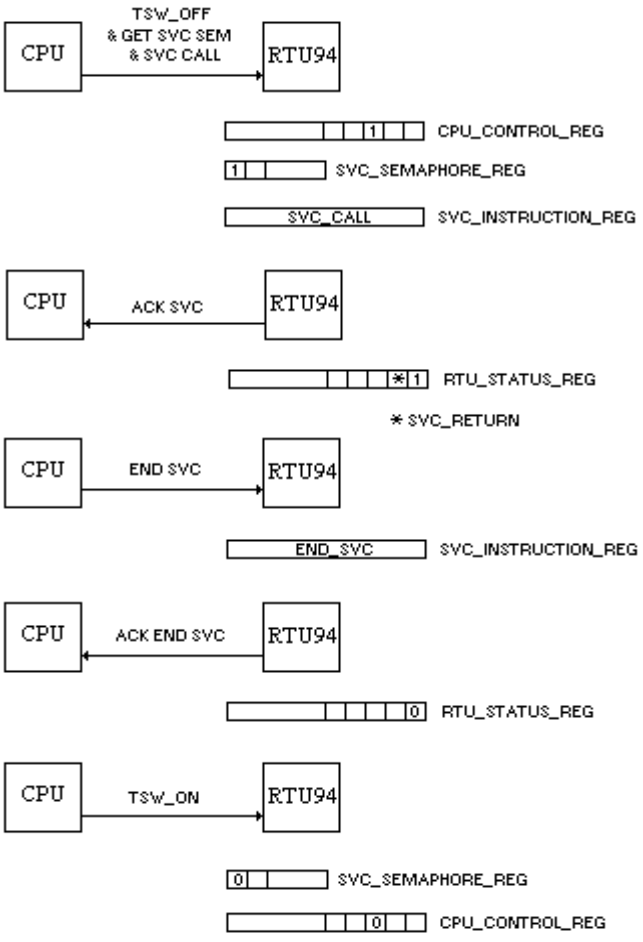
Terminate protocol

Operation: 1 - The cpu must set TSW_FLAG_CONTROL on CPU_CONTROL_REG to make sure that no taskswitch from RTU94 is comming and then set CALL_NEXT_TASK_ID on CPU_CONTROL_REG.
2 - RTU94 sets ACK_NEXT_TASK_ID on CPU_STATUS_REG to acknowledge the call for next task id.
3 - The cpu must wait until ACK_NEXT_TASK_ID is set and then clear CALL_NEXT_TASK_ID.
4 - RTU94 clear ACK_NEXT_TASK_ID and place next task id on NEXT_TASK_ID.
5 - The cpu must wait until ACK_NEXT_TASK_ID is cleared and then it gets new task id to switch to by reading NEXT_TASK_ID. After reading next task id the cpu must (if you want other tasks to be scheduled on the cpu) clear TSW_FLAG_CLEAR.



Service Call protocol

Operation: 1 - The cpu must set TSW_FLAG_CONTROL on CPU_CONTROL_REG to make sure that no taskswitch from RTU94 is comming and then allocate the SVC_SEMAPHORE_REG (if it is a multiprocessor system) to make sure that no other cpu is gonna use the SVC_INSTRUCTION_REG. After that the cpu can write the service call to SVC_INSTRUCTION_REG.
2 - RTU94 sets SVC_ACK_FLAG on RTU_STATUS_REG when it start to perform the service. When **wait_for_semafore** or **wait_for_flag** service is performed the SVC_RETURN on RTU_STATUS_REG is affected.
3 - The cpu must wait until SVC_ACK_FLAG is set and then it write the **end_svc** to SVC_INSTRUCTION_REG, to inform RTU94 that its the end of the service call.
4 - RTU94 clear SVC_ACK_FLAG.
5 - The cpu must wait until SVC_ACK_FLAG is cleared, to be sure that the service is performed. After SVC_ACK_FLAG is cleared the cpu must deallocate the SVC_SEMAPHORE_REG (if it is a multiprocessor system) and (if you want other tasks to be scheduled on the cpu) clear TSW_FLAG_CONTROL.



3 Service calls

In this chapter all the service calls to RTU94 are described and which instruction format they have. As explained in the previous chapter the service calls are placed on the SVC_INSTRUCTION_REG.

Activate

Description: Activates a task with task id. number Task Id.
Argument: Task Id = a task number from 0 to 63.

Instruction Format:

15	14	13	12	11 10 9 8	7 6 5 4 3 2 1 0
0	0	0	1	N.U	Task Id	N.U

*N.U=Not Used.

Init_task_tcb

Description: Initiates a task with Priority and Task Id on Cpu Nr.
Arguments: Priority = a priority number from 0 (highest priority) to 7 (lowest priority). Task Id = a task number from 0 to 63. Cpu Nr = a cpu number from 0 to 2 or a 3 that specifies that the task can be scheduled to run on whichever cpu that is able to execute it.

Instruction Format:

15	14	13	12	11	10 9 8	7 6 5 4 3 2 1 0
0	0	1	0	N.U	Priority	Task Id	Cpu Nr.

*N.U=Not Used.

Init_period_time

Description: Initiates the periodic time for the task.
Arguments: Cpu Nr = a cpu number from 0 to 2. Period Time = a periodic time number that corespond to following bitpattern (when time_sync's period time is set to 2 ms):

00001 = 1 -> 2 ms
00010 = 2 -> 10 ms
00100 = 4 -> 100 ms
01000 = 8 -> 1000 ms
10000 = 16 -> 10 s

Note: The period time can be change by changing the clock frequency on time_sync. Only task 0 - 32 can be initiated to be periodic.
Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	N.U								Period Time			Cpu Nr.

*N.U=Not Used.

Wait_for_next_period

Description: Place the task in the periodic waiting queue until its periodic time has expired, which is initiated by init_period_time.
Argument: Cpu Nr = a cpu number from 0 to 2.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	N.U											Cpu Nr.

*N.U=Not Used.

Off_period_start

Description: Disable periodic start for the task.
Argument: Cpu Nr = a cpu number from 0 to 2.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	N.U											Cpu Nr.

*N.U=Not Used.

Delay

Description: Place the task in the delay waiting queue until the delay time has expired.
Arguments: Cpu Nr = a cpu number from 0 to 2.
Delay Time = a delay time number 0 to 31
(when time_sync's period time is set to 2 ms):
0 -> trig start (synchronization with time_sync),
1 -> 2 ms , 2 -> 4 ms , ... 31 -> 62 ms.
Note: Only task 0 - 32 can be delayed.
Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	N.U							Delay Time				Cpu Nr.

*N.U=Not Used.

Wait_irq_external

Description: Place the task in wait state for interrupt with number Irg. Nr.
Arguments: Cpu Nr = a cpu number from 0 to 2. Irq. Nr = an interrupt number from 0 to 3.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	N.U								Irq. Nr.			Cpu Nr.

*N.U=Not Used.

Init_watch_dog

Description: Starts or stopps a watchdog with number Nr.
Arguments: Cpu Nr = a cpu number from 0 to 2. Nr = a watchdog number from 0 to 2.
On = 1 to start watchdog and 0 to stop watchdog.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	N.U							On	Nr.			Cpu Nr.

*N.U=Not Used.

Kik_dog

Description: Reset the counter in watchdog with number Nr.
Argument: Nr = a watchdog number from 0 to 2.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	N.U									Nr.		N.U

*N.U=Not Used.

Wait_for_semafore

Description: If the semafore is free, the task takes it and continue the execution. If the semafore is already taken, the task is placed in the semafore queue until it gets the semafore.
Argument: Cpu Nr = a cpu number from 0 to 2.
Affected register: Bit SVC_RETURN on RTU_STATUS_REG is set to 0 if the semafore got allocated else 1.
Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	N.U											Cpu Nr.

*N.U=Not Used.

Release_semafore

Description: Release the semafore and the first task in the semafore queue gets it.
Arguments: None.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	N.U											

*N.U=Not Used.

Wait_for_flag

Description: If the event flag is not set, the task will be placed in the event flag queue until the flag gets set. If the event flag is set the task will continue the execution.
Argument: Cpu Nr = a cpu number from 0 to 2.
Affected register: Bit SVC_RETURN on RTU_STATUS_REG is set to 0 if the event flag got allocated else 1.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	N.U											Cpu Nr.

*N.U=Not Used.

Set_or_reset_flag

Description: Set or reset the event flag.
Arguments: Cpu Nr = a cpu number from 0 to 2. Set = 1 all event waiting tasks starts and 0 all tasks that wait for flag are placed in the event flag queue.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	N.U										Set	Cpu Nr.

*N.U=Not Used.

End_svc

Description: Tells RTU94 ,that a service call is finished.
Arguments: None.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	N.U											

*N.U=Not Used.

4 C-interface

In this chapter the C functions avalible for RTU94 are explained.

Initialise a cpu to work with RTU94

Init

```
void    init(int cpu_nr);
```

This function initialise variables, tables and lists wich are used by the cpu. The **cpu_nr** specifies which cpu number the cpu will have i.e which RTU taskswitch interrupt level it must respond to.

cpu_nr = 0 (cpu responds to RTU taskswitch interrupt at level 4)
= 1 (cpu responds to RTU taskswitch interrupt at level 5)
= 2 (cpu responds to RTU taskswitch interrupt at level 6)

Example

```
void main(void)
{               init(0);        /*Initiate cpu to work as cpu 0 (i.e  respond  to RTU taskswitch
                                                interrupt at level  4*/
                :
}
```

Task Management Functions

Enable / disable taskswitch

```
void    on_tsw(void);
```

This function makes RTU taskswitch enabled, on a cpu.

```
void    off_tsw(void);
```

This function makes RTU taskswitch disabled, on a cpu.

Example

```
void task (void)
{               off_tsw();
                :
                section where RTU taskswitch is not allowed
                :
                on_tsw();
}
```

Initialize a task

void init_task_tcb(int prio,int task_id,void *task,char *stack,int stacksize,int cpu_nr);

This function initialise a tasks TCB and on which cpu, specified by **cpu_nr**, it will run on.
cpu_nr = 0 (the task is running on cpu 0).
= 1 (the task is running on cpu 1).
= 2 (the task is running on cpu 2).
= 3 (the task is running on whichever cpu that has time to execute it).
prio = task priority, a number between 0 (high priority) to 7 (low priority).
task_id = task identity number, an id number between 0 to 63 (se note about id number 63).
task = pointer to the start address of the task.
stack = pointer to an allocated stack for the task.
stacksize = size of the stack ,which is allocated for the task.

Example

```
extern void task1(void)
#define size 1024           /* Stack size */
char stack1[size];          /* Allocate stack */
void task(void)
{           init_task_tcb( 7, 1, task1,stack1,size,1);
            :
}
```

Note: When a task is initiated to run on whichever cpu that has time to execute it, the task code must be copied onto every initiated cpu. The id number **task_id** = 63 is always the identity number of the idle task.

Activate a Task

int activate(int task_id);

This function activates a task with specified **task_id**. RTU94 gives a taskswitch interrupt if the activated task has a higher priority level than the running task. Activate returns 0 if **task_id** is already activated else 1.

Example

```
extern void task1(void)

void task(void)
{
   activate(1);
}
```

Terminate a task

void terminate(void);

This function terminates the running task. The TCB and stack are reseted and if the task is activated again, it will start from the beginning.

Example

```
void task(void)
{   :
    terminate();
     :
}
```

Switch task

void taskswitch(void);

This function switch the running task. The TCB and stack are saved and if the task is activated again, it will start from the place it where taskswitched.

Example

```
void task(void)
{   :
    taskswitch();
     :
}
```

Time Functions

Timer on / off

void timer_on(void);

This function turns on the time_sync clock on the RTU94. The time_sync period can be between 100µs - ∞.

void timer_off(void);

This function turns off the time_sync on the RTU94.

Example

```
void task(void)
{           timer_off();
             :
            section where the time functions are not allowed
             :
            timer_on();
}
```

Note: The timer_off turns off the time_sync clock and all the functions which use that clock will be out of function. Therefore you must be aware of what you are doing with the system, because one cpu might want to use time functions.

Initialize the periodic time

void init_period_time(int per_time);

This function initialise the periodic time **per_time** for a task.

per_time (when time_sync's period time is set to 0.5 ms)

= 1 -> 0.5 ms period time.
= 2 -> 2.5 ms period time.
= 4 -> 25 ms period time.
= 8 -> 250 ms period time.
= 16 -> 2.5 s period time.

Note: The period time can be change by changing the clock frequency on time_sync. Only task 0 - 32 can be initiated to be periodic.

Example

```
void task(void)
{           init_period_time(4);      /*Initialise the period time to 25 ms, when
                                        time_sync's period time is set to 0.5 ms. */
              :
}
```

Wait for next period

void wait_for_next_period(void);

This function suspends the executing task until the period time has expired. After expired period time the task is scheduled into the system again and if it has the highest priority the RTU94 interrupt the cpu to switch to that task.

Example

```
void task(void)
{           init_period_time(2);        /*Initialise the period time to 2.5 ms, when
                                        time_sync's period time is set to 0.5 ms. */
            wait_for_next_period();     /*Suspend the task until the period time has
expired.*/
                :
}
```

Note: Initializeing of period time must be done for the task, before waiting for next period.

Disable periodic start

void off_period_start(void);

This function disables periodic start for the task.

Example

```
void task(void)
{           :
            off_period_start();
              :
}
```

Delay

void delay(int delay_time);

This function suspends the executing task until the **delay_time** has expired. After expired delay time the task is scheduled into the system again and if it has the highest priority the RTU94 interrupt the cpu to switch to that task.

delay_time (when time_sync's period time is set to 0.5 ms)

= 0 -> trig start (synchronization with the time_sync clock),
= 1 -> 0.5 ms
= 2 -> 1 ms
:
= 31 -> 15.5 ms.

Example

```
void task(void)
{                   :
                delay(2);       /*Suspend the task at least 1 ms, when
                                time_sync's period time is set to 0.5 ms.*/
                    :
}
```

Note: Only task 0 - 32 can be delayed.

Interrupt Handling Function

Wait for interrupt

void wait_irq_external(int irq_nr);

This function suspends the executing task until an interrupt with number **irq_nr** (0, 1, 2 or 3) has occured. After the interrupt has occured the task is scheduled into the system again and if it has the highest priority the RTU94 interrupt the cpu to switch to that task.

Example

```
void task(void)
{               wait_irq_external(1);       /*Suspend task until interrupt 1 has occured*/
                   :
}
```

Watch Dog Functions

Initialize a watchdog

void init_watch_dog(int on_off,int dog_nr);

This function initialise a watchdog with number **dog_nr** (0, 1 or 2) to be on (**on_off**=1) or off (**on_off**=0) and suspends the, when **on_off**=1, task until the watchdog timer has expired 25 ms (when time_sync's period time is set to 0.5 ms). After 25 ms has expired the task is scheduled into the system again and if it has the highest priority the RTU94 interrupt the cpu to switch to that task.

Example

```
void task(void)
{                   :
                  init_watch_dog(1,1);        /*Suspend task until watchdog 1 timer has expired
                                               25 ms*/
                    :
                  Watchdog 1 has expired 25 ms, because no task has kicked the dog.
                    :
                  init_watch_dog(0,1);        /*Turn off  watchdog timer 1*/
                    :
}
```

Reset a watchdog timer

void kik_dog(int dog_nr);

This function reset the watchdog timer on watchdog with number **dog_nr** (0, 1 or 2).

Example

```
void task(void)
{                   :
                  kik_dog(2);   /*Reset watchdog 2 timer*/
                    :
}
```

Flag Functions

Wait for semafore

```
void    wait_for_semafore(void);
```

This function suspends the executing task until it has got the semafore. After getting the semafore the task is scheduled into the system again and if it has the highest priority the RTU94 interrupt the cpu to switch to that task.

Example

```
void task(void)
{               :
            wait_for_semafore();
               :
}
```

Release semafore

```
void    release_semafore(void);
```

This function release the semafore and the next task in the semafore queue gets the semafore.

Example

```
void task(void)
{               :
            release_semafore();
               :
}
```

Wait for flag

```
void    wait_for_flag(void);
```

This function suspends the executing task until the event flag is set. After event flag got set the task is scheduled into the system again and if it has the highest priority the RTU94 interrupt the cpu to switch to that task.

Example

```
void task(void)
{               :
            wait_for_flag();
               :
}
```

Set / reset flag

```
void    set_or_reset_flag(int set_reset);
```

This function set or reset the event flag.
set_reset = 1 -> set event flag.
= 0 -> reset event flag.

Example

```
void task(void)
{                   :
                set_or_reset_flag(1);
                  :
                set_or_reset_flag(0);
                  :
}
```

Help function

Get status on a local task

```
int task_status(int task_id);
```

This function returns the status of a task that runs local.
task_id = task identity number, an id number between 0 to 63.
Status that a task can be in:
0=Activated,2=Running,3=Wait_Event_Flag
4=Wait_For_Semafore,5=Wait_For_Next_Period,6=Delay
7=Wait_Irq_External

Example

```
void task(void)
{               int a;
                a=task_status(1);           /*a gets the status of task 1*/
                  :
}
```

5 System configuration

In this chapter gives an example of how to configure a realtime system using the RTU94 and [FORCE] VME boards. The following configuration is only an example. It is possible to use more or less VME boards or other VME boards than [FORCE].
The system that we use consist of one VME box with three SYS68K CPU-3VA boards and one SYS68K DRAM-1.
The IACKIN must be connected to IACKOUT (with a jumper) on the VMEbus, on slots with no SYS68K cards inserted.
See appendix A for short information and jumper settings for the [FORCE] cpu and memory boards.
The board inserted in slot 1 on the VMEbus is configured as the system master i.e it drives SYSCLK and handles the arbitration on the bus. The other boards are configured as system slaves i.e they do not drive SYSCLK and do not handles the arbitration.

We uses two types of arbiter, to control the bus mastership on the VME bus.One is a priority arbiter placed on a CPU-3VA board and one is a round robin arbiter implemented on a XILINX 4010. Both are 4 levels arbiters.
The arbiter uses
- the three bus request lines (BR0-BR3).
- the three bus grant out lines (BG0OUT-BG3OUT).
- the bus clear line (BCLR),is not used on the XILNX arbiter.
- the bus busy line (BBSY).

to control the bus. Each cpu board has a bus master timeout (we use 1,5µs for timeout) and a bus request level (BR0-BR3). A bus master realease the bus after the timeout time or after getting a BCLR signal from the arbiter. When a cpu board wants the bus it drives its BRx low. The arbiter detects the BRx, if the bus is free (BBSY is high) the arbiter drives the BGxOUT low. The bus grant then passes by cpu boards until it gets to one which is driving bus request on same level as the bus grant level, this is usually called the bus grant daisy chain. After getting a bus grant the cpu board drives BBSY low and realese BRx. Because of the bus grant daisy chain the cpu card which is located closest to the arbiter, which is located in the first slot, has the highest priority. The differences between how the two arbiters works are.
- if a cpu board already has the bus and a cpu with higher priority wants it the priority arbiter sends a BCLR to the bus master.
- the round robin arbiter don't acts on priority, instead it searches for a bus request. When a bus request is found the arbiter stops on the current request level. And upon release of the bus the arbiter starts to search for a new bus grant on current request level minus one and if no request there, decrease the level again. When level 0 has been checked level 3 is entered and the search continues.

When a multiprocessor system is going to be built, a round robin arbiter is preferbly used.
Figure 3 shows where each part of the system is placed.

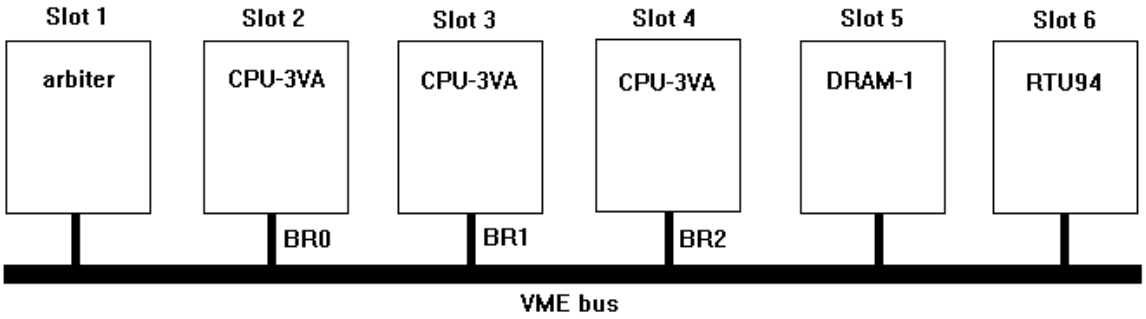

Slot 1
Slot 2
Slot 3
Slot 4
Slot 5
Slot 6
arbiter
CPU-3VA
CPU-3VA
CPU-3VA
DRAM-1
RTU94
BR0
BR1
BR2
VME bus

Figure 3

6 Software development

In this chapter, the software development, for [FORCE] SYS68K boards, with Microtec Research [Microtec] tools on a PC is described.
The tools which are described are Microtec ASM68K assembler, LNK68K linker, XRAY68K debugger, MCC68K ansi-C compiler.
A serial cabel is used to download programs and XRAY68K debugger to the CPU boards from the PC (note that COM68000 * or other downloaders can be used to download programs when the on-board** debugger wants to be used).
* COM68000 is a C-program implemented of people working at Mälardalens University IDt/Department of Real-Time Computer Systems.
** FORCE on-board debugger (it is a ROM resident firmware placed on each cpu board).

See below for a brief description of how to use the assembler,linker etc.

Assembler directives
Assembly a source (.src) file and generate an object (.obj) file and a list (.lst) file.
Example: ASM filename.src
The example above uses following batch file
ASM.BAT
```
asm68k -l >%1.lst %1.src
```

Compiling directives
Compiling a C (.c) file and generate a source (.src) file.
Example: CC filename.c
The example above uses following batch file for relative addressing
CC.BAT
```
mcc68k -Gf -g -l -Mcp -Md5 -nOc -nOl -S -o%1.src %1.c
```

The example above uses following batch file for absolute addressing
CC.BAT
```
mcc68k -Gf -g -l -Mca -Mda -nOc -nOl -S -o%1.src %1.c
```

-Gf = Generates fully qualified path name for input files.
-g = Generates debug information.
-l = Generates a source listing with errors.
-Mcp = Directs the compiler to use PC-relative addressing for all code references.
-Md5 = Directs the compiler to use A5-relative addressing for all static data references.
-Mca =Directs the compiler to use absolute addressing for all code references.
-Mda = Directs the compiler to use absolute addressing for all data references.
-nOc = Generates code to pop stack after every function call.
-nOl = Code hoisting and cross jump optimizations are disabled.
-o%l.src %l.c = Generate output file with (.src) extension from the origin file with extension (.c)

Linking directives

Links the object (.obj) files specified in the command (.cmd) file and generate an absolute (.abs) file and a map (.map) file.
Example: LINK filename.cmd
The example above uses following batch file
LINK.BAT
lnk68k -c %1.cmd -o %1.abs -m >%1.map

Linker command file

```
CHIP    68010                                   /*Specifies target microprocessor*/
LISTABS INTERNALS,PUBLICS       /*Is used when downloading with IEEE-695 format
*/
LISTMAP INTERNALS,PUBLICS
FORMAT   S                                      /*Is used when downloading with
motorola S-record format*/
FORMAT  IEEE                            /*Is used when downloading with IEEE-695 format*/
PUBLIC  ????STACKTOP=$1000          /*Defines were the stack starts*/
SECT    local=$1000                 /*Address where local memory starts*/
SECT    global=$100000              /*Address where global memory starts*/
ORDER  local,KERNCODE,KERNDATA,code,var vars,literals,strings,const, zerovars
            /*local is changed to global if  global memory is used for the program*/
INDEX   ?A5,vars                    /*Used when relative addressing*/
LOAD    RF_KER.OBJ                  /*Routines for communicating with RTU94 and
performing                            taskswitch */
LOAD    CUSKCRIT.KER                            /*Critical routines that RF_KER
uses*/
LOAD    any objects file
LOAD    BASINOUT.OBJ                /*Hardware dependent i/o routines*/
LOAD    INPOUT.OBJ                  /*Hardware independent i/o routines*/
LOAD    C:\MCC68K\68000\MCC68KAB.LIB    /*Used when absolute addressing*/
LOAD    C:\MCC68K\68000\MCC68KA5.LIB    /*Used when relative addressing*/
END
```

Downloading program to cpu board via serial cable
If using XRAY68K debugger: (the .abs file must be in IEEE-695 format)
xhm68k filename.abs (to be able to use this you must first download the XRAY debugger by writing xhm68k -e l: filename.abs).

If using COM68000:(the .abs file must be in motorola S-record format)
com68000 filename.abs

Appendix A

SYS68K CPU-3VA

Specification

Microprocessor	68010 CPU 10 MHz.
DMA Controller	68450 DMAC 10 MHz.
Memory Management	68451 MMU 10 MHz with software programmable address translation, paging and address range protection.
Serial I/O	68561 Multi-Protocol Communication Controller with software-selectable baud rate from 110 to 38400 baud and variable I/O signal assignment.
Control	68230 PI/T for local control and timer function.
Real Time Clock	58167 RTC with Calendar and on-board battery backup.
SRAM	32K bytes of static RAM (70ns).
EPROM	256K bytes of EPROM (max) (JEDEC compatible devices)
VMEbus	Full VMEbus compatible interface with a slave bus arbitration Slot 1 Control functions.
Arbiter	4-level prioritized bus arbiter with bus clear generation.
Firmware	32K bytes of monitor called SYS68K-3 MONITOR.
Power Requirements	+5V/3.0A (max) +12V/200mA (max) -12V/200mA (max)
Operating Temperature	0 to 50 degrees C.
Storage Temperature	-50 to 85 degrees C.
Relative Humidity	0-95 % (non-condensing)
Board Dimensions	Double Eurocard 234x160mm (9.2 x 6.3”)

Memory Map

Address	
$000000-$000007	Start Vectors (SYSTEM Area).
$000008-$007FFF	32K Byte SRAM.
$F00000-$F3FFFF	EPROM Area SYSTEM and USER.
$F40000-$F4FFFF	LOCAL I/O Devices.
$FF0000-$FFFFFF	Short I/O Address Range (VMEbus).

Address Assignment of the I/O Devices

Address	I/O Device
$F40000	MPCC 68561.
$F40801	PIT 68230.
$F41801	RTC 58167A.
$F45000	DMAC 68450.
$F46000	MMU 68451.

Jumper Settings

Time settings

Jumper	Connection between	Function
B7	7-8	Bus error timout is set to min 122000 µs.
	1-10	Must be connected.
	1-9	Bus Mastership timeout is set to max 2 µs.

Interrupt level settings

Jumper	Connection between	Function
B10	1-16	IRQ1 is enabled.
	2-15	IRQ2 is enabled.
	3-14	IRQ3 is enabled.
	4-13	IRQ4 is enabled (Cpu 0 must have IRQ4 enabled).
	5-12	IRQ5 is enabled (Cpu 1 must have IRQ5 enabled).
	6-11	IRQ6 is enabled (Cpu 2 must have IRQ6 enabled).
	7-10	IRQ7 is enabled.
	8-9	Must be connected.

Note: A not inserted jumper is equalent to a disabled VMEbus IRQ signal. If a multi processor RTU system is used, only one of IRQ4 to IRQ6 is allowed to be enabled on a cpu (Because the RTU94 uses IRQ4 to IRQ6 to tell a cpu to perform a taskswitch).

Arbiter settings

	With on-board arbiter and BR3	With external arbiter and BR3
Jumper	Connection between	Connection between
B15	15-18	15-18
	9-24	9-8
	11-22	11-6
	12-21	12-5
	13-14	13-14
	19-20	4-3

Note: If the on-board arbiter is used (only one arbiter is allowed on the VMEbus) the board must be placed in slot 1.

Various settings

Jumper	Connection between	Function
B16	1-2	BCLR is enabled.
	3-4	SYSCLK is driven to the VMEbus.

Note: SYSCLK must be disabled if an external arbiter (wich usually drives the SYSCLK) is used, else it must be enabled.

Jumper	Connection between	Function
B17	Open	SYSRESET is not driven to the VMEbus.
	Inserted	SYSRESET is driven to the VMEbus.
B19	Open	SYSRESET is not received from the VMEbus.
	Inserted	SYSRESET is received from the VMEbus.

For more information about jumper settings and jumpers, see [FORCE] SYS68K CPU-3 (V) hardware user's manual.

SYS68K DRAM-1

Specification

Storage Capacity	512K Bytes.
Word Length	8 Bit with 1 Parity Bit. 16 Bit with 2 Parity Bits.
Page Boundaries	Free Jumper Selectable Base Address (256K Byte Pages) Address Modifier Decoding.
Control	Control and Status Register for Multi Mode Control.
Access Times	Write Access 190 ns (typ). Write Access 210 ns (max). Read Access 300 ns (typ). Read Access 320 ns (max). Write cycle 355 ns (max). Read cycle 400 ns (max).
Operating Modes	Write (Word or Byte). Read (Word or Byte). Read Modify Write (Word or Byte). Read/Write protection, jumper selectable or software programmable. Standby Power Mode for Data. Saving at Main Power Down.
Power Requirements	+5V/2.0A (max) Operating Mode. +5V/1.8A (typ).
Standby Power Req.	+5V STDBY/1.0A (max) Standby Mode.
Operating Temperature	0 to 50 degrees C.
Storage Temperature	-50 to 85 degrees C.
Relative Humidity	0-95 % (non-condensing)
Board Dimensions	Double Eurocard 234x160mm (9.2 x 6.3")

Jumper Settings

Base address settings

Base Address Selection of the Control/Status Register ($FFF000).

Jumper	Connection between
BR1	-*-
BR2	5-12
	6-11
	7-10
	8-9

-*- No jumpers connected.

Memory Area1 settings (start address $100000, end address $13FFFF)
Jumper connections between

FA 1 - FB 1
FA 2 - FB 2
FA 3 - FB 3
FA 5 - FB 5
FA 6 - FB 6

Memory Area2 settings (start address $140000, end address $17FFFF)
Jumper connections between

KA 1 - KB 1
KA 2 - KB 2
KA 3 - KB 3
KA 5 - KB 5

For more information about jumper settings and jumpers, see [FORCE] SYS68K DRAM-1/2 user's manual.

Appendix B

Testprogram

In this appendix a testprogram is presented. The testprogram tests all avalible services in a sequence, on one cpu. If the test is passed, it is quite sure that the system is ok for one cpu. To test different cpu's change the defined CPUNR .

```
#include <stdio.h>
#include "sercom.h"
#include "rf_ker.h"
#define CPUNR 2         /*CPU 2 is tested. Change CPUNR to the cpu that is wanted to be
tested (0 or 1 or 2)*/
/*************************************************************************
*/
/*****************************Watch-dog tasks*********************************/
int kick;
#define size0 128
char stack0[size0];
void t0_main(void)
                {
                int i=0;
                outstr(1,"\n\rChecking Watch-dog 0 to 2");
                while(1)
                                {
                                activate(1);
                                activate(2);
                                kick=0;
                                init_watch_dog(1,0);
                                init_watch_dog(0,0);
                                if (kick!=20000)
                                                {
                                                outstr(1,"\n\rWatch-dog 0 is not OK");
                                                i=1;
                                                }
                                activate(3);
                                activate(4);
                                kick=0;
                                init_watch_dog(1,1);
                                init_watch_dog(0,1);
                                if (kick!=20000)
                                                {
                                                outstr(1,"\n\rWatch-dog 1 is not OK");
                                                i=1;
                                                }
                                activate(5);
                                activate(6);
                                activate(7);
                                kick=0;
```

```
                        init_watch_dog(1,2);
                        init_watch_dog(0,2);
                        if (kick!=30000)
                                {
                                outstr(1,"\n\rWatch-dog 2 is not OK");
                                i=1;
                                }
                        if (i==0)
                                {
                                outstr(1,"\n\rWatch-dog 0 to 2 is OK");
                                }
                        activate(8);
                        terminate();
                        }
                }
#define size1 128
char stack1[size1];
void t1_main(void)
                {
                while(kick<10000)
                        {
                        kik_dog(0);
                        kick++;
                        }
                terminate();
                }
#define size2 128
char stack2[size2];
void t2_main(void)
                {
                while(kick<20000)
                        {
                        kik_dog(0);
                        kick++;
                        }
                terminate();
                }
#define size3 128
char stack3[size3];
void t3_main(void)
                {
                while(kick<10000)
                        {
                        kik_dog(1);
                        kick++;
                        }
                terminate();
                }
#define size4 128
```

```
char stack4[size4];
void t4_main(void)
		{
		while(kick<20000)
				{
				kik_dog(1);
				kick++;
				}
				terminate();
		}
#define size5 128
char stack5[size5];
void t5_main(void)
		{
		while(kick<10000)
				{
				kik_dog(2);
				kick++;
				}
		terminate();
		}
#define size6 128
char stack6[size6];
void t6_main(void)
		{
		while(kick<20000)
				{
				kik_dog(2);
				kick++;
				}
		terminate();
		}
#define size7 128
char stack7[size7];
void t7_main(void)
		{
		while(kick<30000)
				{
				kik_dog(2);
				kick++;
				}
		terminate();
		}
/*****************************End Watch-dog tasks*****************************/
/*****************************************************************************
*/
/*****************************Wait for next per tasks************************/
int time;
int time1;
```

```
int stop;
int finish;
#define size8 128
char stack8[size8];
void t8_main(void)
        {
        outstr(1,"\n\rChecking Wait for next period");
        finish=0;
        stop=0;
        activate(15);
        while(1)
                {
                time=0;
                init_period_time(1);
                wait_for_next_period();
                off_period_start();
                stop=1;
                time1=time;
                activate(9);
                terminate();
                }
        }
int time2;
#define size9 128
char stack9[size9];
void t9_main(void)
        {
        while(1)
                {
                stop=0;
                time=0;
                init_period_time(2);
                wait_for_next_period();
                off_period_start();
                stop=1;
                time2=time;
                activate(10);
                terminate();
                }
        }
int time4;
#define size10 128
char stack10[size10];
void t10_main(void)
        {
        while(1)
                {
                stop=0;
                time=0;
```

```
                        init_period_time(4);
                        wait_for_next_period();
                        off_period_start();
                        stop=1;
                        time4=time;
                        activate(11);
                        terminate();
                        }
                }
int time8;
#define size11 128
char stack11[size11];
void t11_main(void)
                {
                while(1)
                        {
                        stop=0;
                        time=0;
                        init_period_time(8);
                        wait_for_next_period();
                        off_period_start();
                        stop=1;
                        time8=time;
                        activate(12);
                        terminate();
                        }
                }
int time16;
#define size12 128
char stack12[size12];
void t12_main(void)
                {
                while(1)
                        {
                        stop=0;
                        time=0;
                        init_period_time(16);
                        wait_for_next_period();
                        off_period_start();
                        stop=1;
                        time16=time;
                        activate(13);
                        terminate();
                        }
                }
#define size13 128
char stack13[size13];
void t13_main(void)
                {
```

```
		while(1)
				{
				if
((time1<=time2)&&(time2<=time4)&&(time4<=time8)&&(time8<=time16))
					{
					outstr(1,"\n\rWait for next period is OK");
					}
				else
					{
					outstr(1,"\n\rWait for next period is not OK");
					}
				activate(14);
				terminate();
				}
		}
#define size14 128
char stack14[size14];
void t14_main(void)
		{
		while(1)
				{
				finish=1;
				terminate();
				}
		}
#define size15 128
char stack15[size15];
void t15_main(void)
		{
		while(finish != 1)
				{
				while(stop==0)
					{
					time++;
					}
				}
		activate(16);
		terminate();
		}
/*****************************End Wait for next per tasks*******************/
/*************************************************************************
*/
/*********************************Delay tasks*********************************/
int deltime;
int deltime0;
int deltime1;
int deltime2;
int deltime3;
int deltime4;
```

```
#define size16 128
char stack16[size16];
void t16_main(void)
		{
		outstr(1,"\n\rChecking Delay");
		finish=0;
		stop=0;
		activate(23);
		while(1)
				{
				deltime=0;
				delay(0);
				deltime0=deltime;
				deltime=0;
				delay(1);
				deltime1=deltime;
				deltime=0;
				delay(2);
				deltime2=deltime;
				deltime=0;
				delay(3);
				deltime3=deltime;
				deltime=0;
				delay(4);
				deltime4=deltime;
				stop=1;
				activate(17);
				terminate();
				}
		}
int deltime5;
int deltime6;
int deltime7;
int deltime8;
int deltime9;
#define size17 128
char stack17[size17];
void t17_main(void)
		{
		while(1)
				{
				stop=0;
				deltime=0;
				delay(5);
				deltime5=deltime;
				deltime=0;
				delay(6);
				deltime6=deltime;
				deltime=0;
```

```
                        delay(7);
                        deltime7=deltime;
                        deltime=0;
                        delay(8);
                        deltime8=deltime;
                        deltime=0;
                        delay(9);
                        deltime9=deltime;
                        stop=1;
                        activate(18);
                        terminate();
                        }
                }
int deltime10;
int deltime11;
int deltime12;
int deltime13;
int deltime14;
#define size18 128
char stack18[size18];
void t18_main(void)
                {
                while(1)
                        {
                        stop=0;
                        deltime=0;
                        delay(10);
                        deltime10=deltime;
                        deltime=0;
                        delay(11);
                        deltime11=deltime;
                        deltime=0;
                        delay(12);
                        deltime12=deltime;
                        deltime=0;
                        delay(13);
                        deltime13=deltime;
                        deltime=0;
                        delay(14);
                        deltime14=deltime;
                        stop=1;
                        activate(19);
                        terminate();
                        }
                }
int deltime15;
int deltime16;
int deltime17;
int deltime18;
```

```
int deltime19;
#define size19 128
char stack19[size19];
void t19_main(void)
        {
        while(1)
                {
                stop=0;
                deltime=0;
                delay(15);
                deltime15=deltime;
                deltime=0;
                delay(16);
                deltime16=deltime;
                deltime=0;
                delay(17);
                deltime17=deltime;
                deltime=0;
                delay(18);
                deltime18=deltime;
                deltime=0;
                delay(19);
                deltime19=deltime;
                stop=1;
                activate(20);
                terminate();
                }
        }
int deltime20;
int deltime21;
int deltime22;
int deltime23;
int deltime24;
#define size20 128
char stack20[size20];
void t20_main(void)
        {
        while(1)
                {
                stop=0;
                deltime=0;
                delay(20);
                deltime20=deltime;
                deltime=0;
                delay(21);
                deltime21=deltime;
                deltime=0;
                delay(22);
                deltime22=deltime;
```

```
                        deltime=0;
                        delay(23);
                        deltime23=deltime;
                        deltime=0;
                        delay(24);
                        deltime24=deltime;
                        stop=1;
                        activate(21);
                        terminate();
                        }
                }
int deltime25;
int deltime26;
int deltime27;
int deltime28;
int deltime29;
#define size21 128
char stack21[size21];
void t21_main(void)
                {
                while(1)
                        {
                        stop=0;
                        deltime=0;
                        delay(25);
                        deltime25=deltime;
                        deltime=0;
                        delay(26);
                        deltime26=deltime;
                        deltime=0;
                        delay(27);
                        deltime27=deltime;
                        deltime=0;
                        delay(28);
                        deltime28=deltime;
                        deltime=0;
                        delay(29);
                        deltime29=deltime;
                        stop=1;
                        activate(22);
                        terminate();
                        }
                }
int deltime30;
int deltime31;
#define size22 128
char stack22[size22];
void t22_main(void)
                {
```

```
        while(1)
                {
                stop=0;
                deltime=0;
                delay(30);
                deltime30=deltime;
                deltime=0;
                delay(31);
                deltime31=deltime;
                deltime=0;
                stop=1;
                if
((deltime0<=deltime1)&&(deltime1<=deltime2)&&(deltime2<=deltime3)&&(deltime3<=delti
me4)

        &&(deltime4<=deltime5)&&(deltime5<=deltime6)&&(deltime6<=deltime7)&&(
deltime7<=deltime8)

        &&(deltime8<=deltime9)&&(deltime9<=deltime10)&&(deltime10<=deltime11)
&&(deltime11<=deltime12)

        &&(deltime12<=deltime13)&&(deltime13<=deltime14)&&(deltime14<=deltime
15)&&(deltime15<=deltime16)

        &&(deltime16<=deltime17)&&(deltime17<=deltime18)&&(deltime18<=deltime
19)&&(deltime19<=deltime20)

        &&(deltime20<=deltime21)&&(deltime21<=deltime22)&&(deltime22<=deltime
23)&&(deltime23<=deltime24)

        &&(deltime24<=deltime25)&&(deltime25<=deltime26)&&(deltime26<=deltime
27)&&(deltime27<=deltime28)

        &&(deltime28<=deltime29)&&(deltime29<=deltime30)&&(deltime30<=deltime
31))
                                {
                                outstr(1,"\n\rDelay is OK");
                                }
                else
                                {
                                outstr(1,"\n\rDelay is not OK");
                                }
                finish=1;
                terminate();
                }
        }
#define size23 128
char stack23[size23];
void t23_main(void)
```

```
		{
		while(finish != 1)
			{
			while(stop==0)
				{
				deltime++;
				}
			}
		activate(24);
		terminate();
		}
/*****************************End Delay tasks*********************************/
/*****************************************************************************
*/
/*****************************Wait for next per tasks*************************/
#define size24 128
char stack24[size24];
void t24_main(void)
		{
		outstr(1,"\n\rChecking Wait for next period again");
		finish=0;
		stop=0;
		activate(27);
		while(1)
			{
			time=0;
			init_period_time(1);
			wait_for_next_period();
			off_period_start();
			stop=1;
			time1=time;
			activate(25);
			terminate();
			}
		}
#define size25 128
char stack25[size25];
void t25_main(void)
		{
		while(1)
			{
			stop=0;
			time=0;
			init_period_time(8);
			wait_for_next_period();
			off_period_start();
			stop=1;
			time8=time;
			activate(26);
```

```
                        terminate();
                        }
                }
#define size26 128
char stack26[size26];
void t26_main(void)
                {
                while(1)
                        {
                        if (time1<=time8)
                                {
                                outstr(1,"\n\rWait for next period again is OK");
                                }
                        else
                                {
                                outstr(1,"\n\rWait for next period again is not OK");
                                }
                        finish=1;
                        terminate();
                        }
                }
#define size27 128
char stack27[size27];
void t27_main(void)
                {
                while(finish != 1)
                        {
                        while(stop==0)
                                {
                                time++;
                                }
                        }
                activate(28);
                terminate();
                }
/*****************************End Wait for next per tasks*******************/
/*************************************************************************
*/
/*********************************Delay tasks*********************************/
#define size28 128
char stack28[size28];
void t28_main(void)
                {
                outstr(1,"\n\rChecking Delay again");
                finish=0;
                stop=0;
                activate(30);
                while(1)
                        {
```

```
                        deltime=0;
                        delay(0);
                        deltime0=deltime;
                        deltime=0;
                        delay(1);
                        deltime1=deltime;
                        deltime=0;
                        delay(2);
                        deltime2=deltime;
                        deltime=0;
                        delay(3);
                        deltime3=deltime;
                        deltime=0;
                        delay(4);
                        deltime4=deltime;
                        stop=1;
                        activate(29);
                        terminate();
                        }
                }
#define size29 128
char stack29[size29];
void t29_main(void)
                {
                while(1)
                        {
                        stop=0;
                        deltime=0;
                        delay(30);
                        deltime30=deltime;
                        deltime=0;
                        delay(31);
                        deltime31=deltime;
                        deltime=0;
                        stop=1;
                        if
((deltime0<=deltime1)&&(deltime1<=deltime2)&&(deltime2<=deltime3)&&(deltime3<=delti
me4)

                &&(deltime4<=deltime30)&&(deltime30<=deltime31))
                                        {
                                        outstr(1,"\n\rDelay again is OK");
                                        }
                        else
                                        {
                                        outstr(1,"\n\rDelay again is not OK");
                                        }
                        finish=1;
                        terminate();
```

```
                        }
                }
#define size30 128
char stack30[size30];
void t30_main(void)
                {
                while(finish != 1)
                        {
                        while(stop==0)
                                {
                                deltime++;
                                }
                        }
                activate(38);
                terminate();
                }
/*****************************End Delay tasks*********************************/
/*****************************************************************************
*/
/*****************************Event flag tasks********************************/
int tf31;
#define size31 128
char stack31[size31];
void t31_main(void)
                {
                while(1)
                        {
                        wait_for_flag();
                        tf31=1;
                        terminate();
                        }
                }
int tf32;
#define size32 128
char stack32[size32];
void t32_main(void)
                {
                while(1)
                        {
                        wait_for_flag();
                        tf32=1;
                        terminate();
                        }
                }
int tf33;
#define size33 128
char stack33[size33];
void t33_main(void)
                {
```

```
		while(1)
				{
				wait_for_flag();
				tf33=1;
				terminate();
				}
		}
int tf34;
#define size34 128
char stack34[size34];
void t34_main(void)
		{
		while(1)
				{
				wait_for_flag();
				tf34=1;
				terminate();
				}
		}
int tf35;
#define size35 128
char stack35[size35];
void t35_main(void)
		{
		while(1)
				{
				wait_for_flag();
				tf35=1;
				terminate();
				}
		}
int tf36;
#define size36 128
char stack36[size36];
void t36_main(void)
		{
		while(1)
				{
				wait_for_flag();
				tf36=1;
				terminate();
				}
		}
int tf37;
#define size37 128
char stack37[size37];
void t37_main(void)
		{
		while(1)
```

```
                        {
                        wait_for_flag();
                        tf37=1;
                        terminate();
                        }
                }

#define size38 128
char stack38[size38];
void t38_main(void)
                {
                int i,l=0;
                outstr(1,"\n\rChecking Event flag");
                while(1)
                        {
                        set_or_reset_flag(0);
                        tf31=0;tf32=0;tf33=0;tf34=0;
                        for (i=31;i<=34;i++)
                                {
                                activate(i);
                                }
                        while ((task_status(31)!=3)||(task_status(32)!=3)

                ||(task_status(33)!=3)||(task_status(34)!=3));
                        set_or_reset_flag(1);
                        while ((task_status(31)!=0)||(task_status(32)!=0)

                ||(task_status(33)!=0)||(task_status(34)!=0));
                        if ((tf31==0)||(tf32==0)||(tf33==0)||(tf34==0))
                                {
                                outstr(1,"\n\rEvent flag is not OK");
                                l=1;
                                }
                        set_or_reset_flag(0);
                        tf34=0;tf35=0;tf36=0;tf37=0;
                        for (i=34;i<=37;i++)
                                {
                                activate(i);
                                }
                        while ((task_status(34)!=3)||(task_status(35)!=3)

                ||(task_status(36)!=3)||(task_status(37)!=3));
                        set_or_reset_flag(1);
                        while ((task_status(34)!=0)||(task_status(35)!=0)

                ||(task_status(36)!=0)||(task_status(37)!=0));
                        if (((tf34==0)||(tf35==0)||(tf36==0)||(tf37==0))&&(l==0))
                                {
                                outstr(1,"\n\rEvent flag is not OK");
```

```
                                        l=1;
                                        }
                                else
                                        {
                                        if (l!=1)
                                                {
                                                outstr(1,"\n\rEvent flag is OK");
                                                }
                                        }
                                activate(46);
                                terminate();
                                }
                }
/*****************************End Event flag tasks***************************/
/*************************************************************************
*/
/*****************************Semafore tasks*********************************/
int ts39;
#define size39 128
char stack39[size39];
void t39_main(void)
                {
                while(1)
                                {
                                wait_for_semafore();
                                ts39=1;
                                release_semafore();
                                terminate();
                                }
                }
int ts40;
#define size40 128
char stack40[size40];
void t40_main(void)
                {
                while(1)
                                {
                                wait_for_semafore();
                                ts40=1;
                                release_semafore();
                                terminate();
                                }
                }
int ts41;
#define size41 128
char stack41[size41];
void t41_main(void)
                {
                while(1)
```

```
                        {
                        wait_for_semafore();
                        ts41=1;
                        release_semafore();
                        terminate();
                        }
                }
int ts42;
#define size42 128
char stack42[size42];
void t42_main(void)
                {
                while(1)
                        {
                        wait_for_semafore();
                        ts42=1;
                        release_semafore();
                        terminate();
                        }
                }
int ts43;
#define size43 128
char stack43[size43];
void t43_main(void)
                {
                while(1)
                        {
                        wait_for_semafore();
                        ts43=1;
                        release_semafore();
                        terminate();
                        }
                }
int ts44;
#define size44 128
char stack44[size44];
void t44_main(void)
                {
                while(1)
                        {
                        wait_for_semafore();
                        ts44=1;
                        release_semafore();
                        terminate();
                        }
                }
int ts45;
#define size45 128
char stack45[size45];
```

```
void t45_main(void)
		{
		while(1)
				{
				wait_for_semafore();
				ts45=1;
				release_semafore();
				terminate();
				}
		}
#define size46 128
char stack46[size46];
void t46_main(void)
		{
		int i,l=0;
		outstr(1,"\n\rChecking Semafore");
		while(1)
				{
				ts39=0;ts40=0;ts41=0;ts42=0;
				wait_for_semafore();
				for (i=39;i<=42;i++)
						{
						activate(i);
						}
				while ((task_status(39)!=4)||(task_status(40)!=4)

		||(task_status(41)!=4)||(task_status(42)!=4));
				release_semafore();
				while ((task_status(39)!=0)||(task_status(40)!=0)

		||(task_status(41)!=0)||(task_status(42)!=0));
				if ((ts39==0)||(ts40==0)||(ts41==0)||(ts42==0))
						{
						outstr(1,"\n\rSemafore is not OK");
						l=1;
						}
				ts42=0;ts43=0;ts44=0;ts45=0;
				wait_for_semafore();
				for (i=42;i<=45;i++)
						{
						activate(i);
						}
				while ((task_status(42)!=4)||(task_status(43)!=4)

		||(task_status(44)!=4)||(task_status(45)!=4));
				release_semafore();
				while ((task_status(42)!=0)||(task_status(43)!=0)

		||(task_status(44)!=0)||(task_status(45)!=0));
```

```
                        if (((ts42==0)||(ts43==0)||(ts44==0)||(ts45==0))&&(l==0))
                                {
                                outstr(1,"\n\rSemafore is not OK");
                                l=1;
                                }
                        else
                                {
                                if (l!=1)
                                        {
                                        outstr(1,"\n\rSemafore is OK");
                                        }
                                }
                        activate(54);
                        terminate();
                        }
        }
/*****************************End Semafore tasks*****************************/
/*****************************************************************************
*/
/*****************************Event flag tasks*******************************/
int tf47;
#define size47 128
char stack47[size47];
void t47_main(void)
        {
        while(1)
                {
                wait_for_flag();
                tf47=1;
                terminate();
                }
        }
int tf48;
#define size48 128
char stack48[size48];
void t48_main(void)
        {
        while(1)
                {
                wait_for_flag();
                tf48=1;
                terminate();
                }
        }
int tf49;
#define size49 128
char stack49[size49];
void t49_main(void)
        {
```

```
		while(1)
				{
				wait_for_flag();
				tf49=1;
				terminate();
				}
		}
int tf50;
#define size50 128
char stack50[size50];
void t50_main(void)
		{
		while(1)
				{
				wait_for_flag();
				tf50=1;
				terminate();
				}
		}
int tf51;
#define size51 128
char stack51[size51];
void t51_main(void)
		{
		while(1)
				{
				wait_for_flag();
				tf51=1;
				terminate();
				}
		}
int tf52;
#define size52 128
char stack52[size52];
void t52_main(void)
		{
		while(1)
				{
				wait_for_flag();
				tf52=1;
				terminate();
				}
		}
int tf53;
#define size53 128
char stack53[size53];
void t53_main(void)
		{
		while(1)
```

```
                {
                wait_for_flag();
                tf53=1;
                terminate();
                }
        }

#define size54 128
char stack54[size54];
void t54_main(void)
        {
        int i,l=0;
        outstr(1,"\n\rChecking Event flag again");
        while(1)
                {
                set_or_reset_flag(0);
                tf47=0;tf48=0;tf49=0;tf50=0;
                for (i=47;i<=50;i++)
                        {
                        activate(i);
                        }
                while ((task_status(47)!=3)||(task_status(48)!=3)

        ||(task_status(49)!=3)||(task_status(50)!=3));
                set_or_reset_flag(1);
                while ((task_status(47)!=0)||(task_status(48)!=0)

        ||(task_status(49)!=0)||(task_status(50)!=0));
                if ((tf47==0)||(tf48==0)||(tf49==0)||(tf50==0))
                        {
                        outstr(1,"\n\rEvent flag again is not OK");
                        l=1;
                        }
                set_or_reset_flag(0);
                tf50=0;tf51=0;tf52=0;tf53=0;
                for (i=50;i<=53;i++)
                        {
                        activate(i);
                        }
                while ((task_status(50)!=3)||(task_status(51)!=3)

        ||(task_status(52)!=3)||(task_status(53)!=3));
                set_or_reset_flag(1);
                while ((task_status(50)!=0)||(task_status(51)!=0)

        ||(task_status(52)!=0)||(task_status(53)!=0));
                if (((tf50==0)||(tf51==0)||(tf52==0)||(tf53==0))&&(l==0))
                        {
                        outstr(1,"\n\rEvent flag again is not OK");
```

```
                                        l=1;
                                        }
                                else
                                        {
                                        if (l!=1)
                                                {
                                                outstr(1,"\n\rEvent flag again is OK");
                                                }
                                        }
                                activate(62);
                                terminate();
                                }
                }
/*****************************End Event flag tasks***************************/
/*************************************************************************
*/
/*****************************Semafore tasks*********************************/
int ts55;
#define size55 128
char stack55[size55];
void t55_main(void)
                {
                while(1)
                                {
                                wait_for_semafore();
                                ts55=1;
                                release_semafore();
                                terminate();
                                }
                }
int ts56;
#define size56 128
char stack56[size56];
void t56_main(void)
                {
                while(1)
                                {
                                wait_for_semafore();
                                ts56=1;
                                release_semafore();
                                terminate();
                                }
                }
int ts57;
#define size57 128
char stack57[size57];
void t57_main(void)
                {
                while(1)
```

```
                        {
                        wait_for_semafore();
                        ts57=1;
                        release_semafore();
                        terminate();
                        }
        }
int ts58;
#define size58 128
char stack58[size58];
void t58_main(void)
        {
        while(1)
                        {
                        wait_for_semafore();
                        ts58=1;
                        release_semafore();
                        terminate();
                        }
        }
int ts59;
#define size59 128
char stack59[size59];
void t59_main(void)
        {
        while(1)
                        {
                        wait_for_semafore();
                        ts59=1;
                        release_semafore();
                        terminate();
                        }
        }
int ts60;
#define size60 128
char stack60[size60];
void t60_main(void)
        {
        while(1)
                        {
                        wait_for_semafore();
                        ts60=1;
                        release_semafore();
                        terminate();
                        }
        }
int ts61;
#define size61 128
char stack61[size61];
```

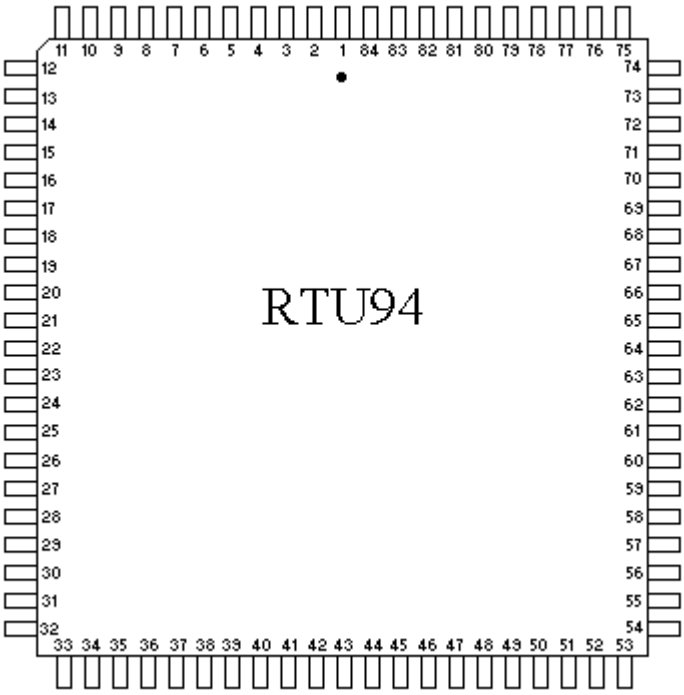
```
void t61_main(void)
		{
		while(1)
				{
				wait_for_semafore();
				ts61=1;
				release_semafore();
				terminate();
				}
		}
#define size62 128
char stack62[size62];
void t62_main(void)
		{
		int i,l=0;
		outstr(1,"\n\rChecking Semafore again");
		while(1)
				{
				ts55=0;ts56=0;ts57=0;ts58=0;
				wait_for_semafore();
				for (i=55;i<=58;i++)
						{
						activate(i);
						}
				while ((task_status(55)!=4)||(task_status(56)!=4)

		||(task_status(57)!=4)||(task_status(58)!=4));
				release_semafore();
				while ((task_status(55)!=0)||(task_status(56)!=0)

		||(task_status(57)!=0)||(task_status(58)!=0));
				if ((ts55==0)||(ts56==0)||(ts57==0)||(ts58==0))
						{
						outstr(1,"\n\rSemafore again is not OK");
						l=1;
						}
				ts58=0;ts59=0;ts60=0;ts61=0;
				wait_for_semafore();
				for (i=58;i<=61;i++)
						{
						activate(i);
						}
				while ((task_status(58)!=4)||(task_status(59)!=4)

		||(task_status(60)!=4)||(task_status(61)!=4));
				release_semafore();
				while ((task_status(58)!=0)||(task_status(59)!=0)

		||(task_status(60)!=0)||(task_status(61)!=0));
```

```
                        if (((ts58==0)||(ts59==0)||(ts60==0)||(ts61==0))&&(l==0))
                                {
                                outstr(1,"\n\rSemafore again is not OK");
                                l=1;
                                }
                        else
                                {
                                if (l!=1)
                                        {
                                        outstr(1,"\n\rSemafore again is OK");
                                        }
                                }
                        outstr(1,"\n\rChecking irq external");
                        outstr(1,"\n\rWaiting for irq0");
                        wait_irq_external(0);
/*      outstr(1,"\n\rWaiting for irq1");
                        wait_irq_external(1);
                        outstr(1,"\n\rWaiting for irq2");
                        wait_irq_external(2);
                        outstr(1,"\n\rWaiting for irq3");
                        wait_irq_external(3);                   */
                        outstr(1,"\n\Irq external is OK");
                        outstr(1,"\n\rTest is finished");
                        terminate();
                        }
                }
/*****************************End Semafore tasks*****************************/
/*****************************************************************************
*/
/*****************************Idle task***************************************/
#define sizeidle 10
char stackidle[sizeidle];
void idle_main(void)
                {
                while(1)
                        {
                        {
                        }
                }
/*****************************End Idle task***********************************/
/*****************************************************************************
*/
void main(void)
                {
                /*serinit(); */
                init(CPUNR);
                init_task_tcb(0,0,t0_main,stack0,size0,CPUNR);
                init_task_tcb(1,1,t1_main,stack1,size1,CPUNR);
                init_task_tcb(2,2,t2_main,stack2,size2,CPUNR);
                init_task_tcb(3,3,t3_main,stack3,size3,CPUNR);
```

```
init_task_tcb(4,4,t4_main,stack4,size4,CPUNR);
init_task_tcb(5,5,t5_main,stack5,size5,CPUNR);
init_task_tcb(6,6,t6_main,stack6,size6,CPUNR);
init_task_tcb(7,7,t7_main,stack7,size7,CPUNR);
init_task_tcb(0,8,t8_main,stack8,size8,CPUNR);
init_task_tcb(1,9,t9_main,stack9,size9,CPUNR);
init_task_tcb(2,10,t10_main,stack10,size10,CPUNR);
init_task_tcb(3,11,t11_main,stack11,size11,CPUNR);
init_task_tcb(4,12,t12_main,stack12,size12,CPUNR);
init_task_tcb(5,13,t13_main,stack13,size13,CPUNR);
init_task_tcb(6,14,t14_main,stack14,size14,CPUNR);
init_task_tcb(7,15,t15_main,stack15,size15,CPUNR);
init_task_tcb(0,16,t16_main,stack16,size16,CPUNR);
init_task_tcb(1,17,t17_main,stack17,size17,CPUNR);
init_task_tcb(2,18,t18_main,stack18,size18,CPUNR);
init_task_tcb(3,19,t19_main,stack19,size19,CPUNR);
init_task_tcb(4,20,t20_main,stack20,size20,CPUNR);
init_task_tcb(5,21,t21_main,stack21,size21,CPUNR);
init_task_tcb(6,22,t22_main,stack22,size22,CPUNR);
init_task_tcb(7,23,t23_main,stack23,size23,CPUNR);
init_task_tcb(0,24,t24_main,stack24,size24,CPUNR);
init_task_tcb(1,25,t25_main,stack25,size25,CPUNR);
init_task_tcb(2,26,t26_main,stack26,size26,CPUNR);
init_task_tcb(3,27,t27_main,stack27,size27,CPUNR);
init_task_tcb(0,28,t28_main,stack28,size28,CPUNR);
init_task_tcb(1,29,t29_main,stack29,size29,CPUNR);
init_task_tcb(2,30,t30_main,stack30,size30,CPUNR);
init_task_tcb(0,31,t31_main,stack31,size31,CPUNR);
init_task_tcb(1,32,t32_main,stack32,size32,CPUNR);
init_task_tcb(2,33,t33_main,stack33,size33,CPUNR);
init_task_tcb(3,34,t34_main,stack34,size34,CPUNR);
init_task_tcb(4,35,t35_main,stack35,size35,CPUNR);
init_task_tcb(5,36,t36_main,stack36,size36,CPUNR);
init_task_tcb(6,37,t37_main,stack37,size37,CPUNR);
init_task_tcb(7,38,t38_main,stack38,size38,CPUNR);
init_task_tcb(0,39,t39_main,stack39,size39,CPUNR);
init_task_tcb(1,40,t40_main,stack40,size40,CPUNR);
init_task_tcb(2,41,t41_main,stack41,size41,CPUNR);
init_task_tcb(3,42,t42_main,stack42,size42,CPUNR);
init_task_tcb(4,43,t43_main,stack43,size43,CPUNR);
init_task_tcb(5,44,t44_main,stack44,size44,CPUNR);
init_task_tcb(6,45,t45_main,stack45,size45,CPUNR);
init_task_tcb(7,46,t46_main,stack46,size46,CPUNR);
init_task_tcb(0,47,t47_main,stack47,size47,CPUNR);
init_task_tcb(1,48,t48_main,stack48,size48,CPUNR);
init_task_tcb(2,49,t49_main,stack49,size49,CPUNR);
init_task_tcb(3,50,t50_main,stack50,size50,CPUNR);
init_task_tcb(4,51,t51_main,stack51,size51,CPUNR);
init_task_tcb(5,52,t52_main,stack52,size52,CPUNR);
```

```
        init_task_tcb(6,53,t53_main,stack53,size53,CPUNR);
        init_task_tcb(7,54,t54_main,stack54,size54,CPUNR);
        init_task_tcb(0,55,t55_main,stack55,size55,CPUNR);
        init_task_tcb(1,56,t56_main,stack56,size56,CPUNR);
        init_task_tcb(2,57,t57_main,stack57,size57,CPUNR);
        init_task_tcb(3,58,t58_main,stack58,size58,CPUNR);
        init_task_tcb(4,59,t59_main,stack59,size59,CPUNR);
        init_task_tcb(5,60,t60_main,stack60,size60,CPUNR);
        init_task_tcb(6,61,t61_main,stack61,size61,CPUNR);
        init_task_tcb(7,62,t62_main,stack62,size62,CPUNR);
        init_task_tcb(7,63,idle_main,stackidle,sizeidle,CPUNR);

        activate(0);

        while (1)

        {}

}
```

Appendix C

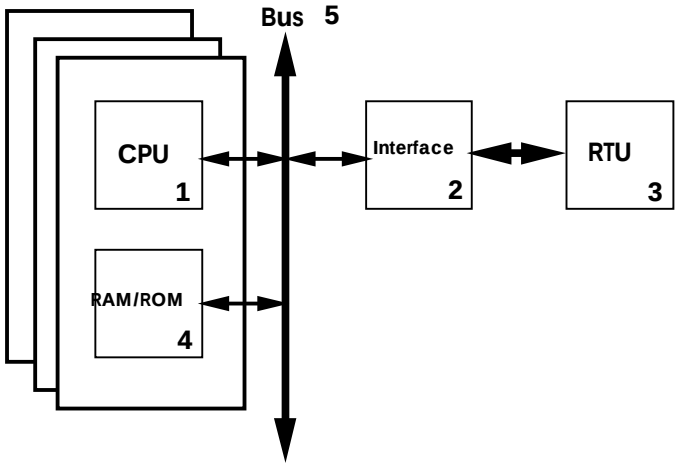


Note: NC = No Connection. D0-D15 = data bus. A0-A3 = address bus.

Number		Number		Number		Number	
1	NC (Com. to external chip)	22	VCC	43	IRQ_IN0 (External interrupt)	64	NC (Test pin)
2	GND	23	D10	44	IRQ_IN1 (External interrupt)	65	CS_N (Chip select)
3	VCC	24	D2	45	IRQ_IN2 (External interrupt)	66	A0
4	DTACK	25	D9	46	IRQ_IN3 (External interrupt)	67	A1
5	CS_N to buffers	26	D1	47	RESET	68	A2
6	D15	27	D8	48	TIME_SYNC	69	A3
7	D7	28	D0	49	CLK	70	IACK
8	D14	29	GND	50	NC	71	NC
9	D6	30	VCC	51	NC	72	NC
10	NC	31	NC	52	NC	73	NC
11	NC	32	NC	53	NC	74	NC
12	NC	33	NC	54	NC	75	NC
13	NC	34	NC	55	NC	76	NC
14	NC	35	NC	56	NC	77	GND
15	D13	36	GND (Com. to external chip)	57	NC (Test pin)	78	VCC
16	D5	37	GND (Com. to external chip)	58	NC (Test pin)	79	AS
17	D12	38	GND (Com. to external chip)	59	NC (Test pin)	80	RW_N
18	D4	39	GND (Com. to external chip)	60	NC (Test pin)	81	DS1
19	D11	40	GND (Com. to external chip)	61	NC (Test pin)	82	IRQ2 (Taskswitch irq6)
20	D3	41	GND (Com. to external chip)	62	NC (Test pin)	83	IRQ1 (Taskswitch irq5)
21	GND	42	GND (Com. to external chip)	63	NC (Test pin)	84	IRQ0 (Taskswitch irq4)

Appendix D

This appendix describes a performance analysis on RTU94 compared with a commercial realtime kernel (RTK) in software. The comparison is analytic, based on data sheets and practical measurings.
It is difficult to compare different realtime kernels, because of the big variation of methods of mesuring performance. Different cpu structures gives problems e.g cpu using cache and/or pipeline. The time behaviour on a program can vary a lot, if a cpu e.g get cache misses. Similar problems is it with pipeline cpus. Cache increase the performance on applications, which often uses the same code part. In realtime systems the executing code is often changing a lot, therefore the performance is sometimes decreasing when using cache in realtime systems.



Figur 4: RTU94 consists of an Interface (2) and a Real-Time Unit (3).

The performance on RTU94 is dependent on how fast the bus (5) can transfer data from cpus (1) to the interface (2). The data is transfered within 1 clock cycle, from the interface to the RTU (3).

Clocktick interrupts

Clocktick interrupts are a necessity for updating timequeues (e.g delay queues) and controllfunctions (e.g watchdogs), when using a realtime kernel in software. The time-resolution of the system is dependent on the clocktick frequency of the system. In nowadays fast system (with few tasks), the time between clockticks are 1 to 2 ms.

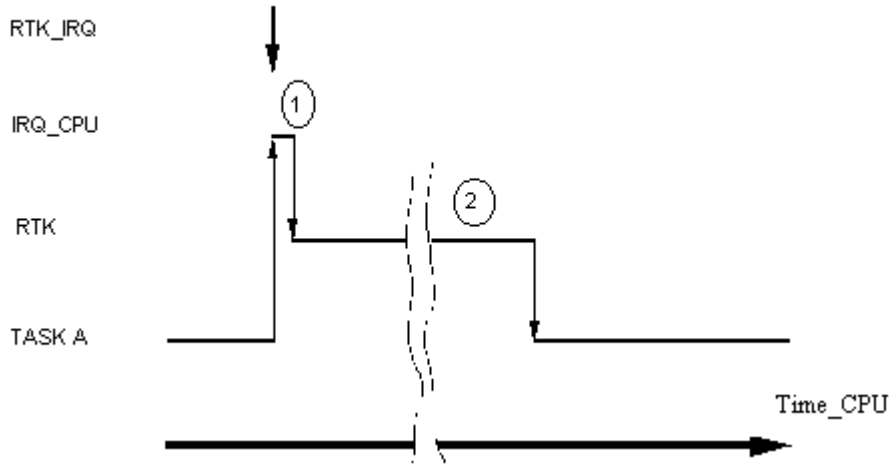
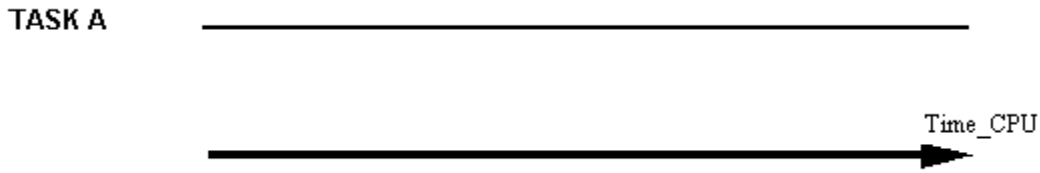


Figure 5: Clocktickadministration in Real-Time Kernels implemented in software

A clock tick interrupt is an external interrupt from a timer (see fig. 3). When an interrupt reach the cpu, it saves registers on the stack i.e an overhead time approx. 10 µs (1). The realtime kernel starts to administrate (2) the cpu i.e control the timequeues, schedule tasks and in some cases check that time demands are fulfilled. The administration time is hard to estimate, since it's not stated in any of the commercial realtime kerenls (that we have analysed) data sheets. This can be due to that a system needs different time to administrate few tasks and a lot of tasks. Some measurings on a commercial realtime kernel, shows that an unloaded RTK needs approx 200 µs administration time (2).
When the RTU is used, the executing task is not interrupted until a task with higher priority is found (by the RTU) in the ready queue.



Figur 6: Clocktickadministration is done by the RTU, not by the CPU

Clocktick administration is for instance used for updating timequeues, therefore the clockticks affects the resolution of the system time. The resolution on RTU94 is 10 µs compared to a comercial RTK, which have approx 1 ms.

Summary

- A lot of time can be saved and easier time analysis can be performed, when the RTU is used as supervisor and clocktick administrator.
- Clocktickadministration is not performed by the cpu or cpus (in a multiprocessor system) when the RTU is used.

Service calls

In this section a time table for the most common service calls are presented. The table is organized in columns with different RTK.

RTU (S): RTU94 as a singel processor RTK. The time consuming part is the parameter transefering between the tasks and the RTU. The system consist of a M68000, 16 MHz on a VME bus.

RTU (M): RTU94 as a multiprocessor RTK with three cpus with a common VME bus. The maximum time (within parenthesis) is dependent on the shared VME bus. The system consists of three M68000, 16 MHz on a VME-bus with a round robin arbitrer.

EX1 RTK: a commercial RTK running on a MC68020, 20 Mhz. In this case it is not stated if the cache is enabled or not. The times presented is the average time for at least 100 service calls.

EX2 RTK: a commercial RTK running on MC68000, 16,7 MHz. This system is similar to RTU(S).

Note: RTU (S) and RTU (M) are presented with clock periods i.e initialize a task is performed within 4 clock periods. Ex1 RTK and Ex2 RTK are presented with µs i.e change priority on a task is performed within 65 µs (Ex1) and 127,1 µs (Ex2).

Service Calls	RTU94 (S)	RTU94 (M)	Ex1 RTK	Ex2 RTK
Task Management Functions				
Enable / disable taskswitch	1	1 (3)		
Initialize a task	4	4 (12)	65	
Change_priority	4	4 (12)	60	127,1
Activate a Task	4	4 (12)	50	109
Terminate a task	4	4 (12)	80	130,2
Switch task	4	4 (12)	81	142,8
Deadline control	0	0	-	-
Time Functions				
Timer on / off	4	4 (12)		
Initialize the periodic time	4	4 (12)		
Wait for next period	4	4 (12)		
Disable periodic start	4	4 (12)		
Delay	4	4 (12)		
Watch Dog Functions				
Initialize a watchdog	4	4 (12)		
Reset a watchdog timer	4	4 (12)		
Flag Functions				
Wait for semafore				
free	5	5 (15)	33	59,9
no wait	5	5 (15)	33	-
nott free, wait	5	5 (15)	85	161,6
Release semafore				
no waiting tasks	4	4 (12)	35	53,7

calling task not preempts	4	4 (12)	56	89,4
calling task preempts	4	4 (12)	75	147,9
Wait for flag				
free	5	5 (15)	-	47,2
no wait	5	5 (15)	-	-
nott free, wait	5	5 (15)	-	143,5
Set / reset flag				
no waiting tasks	4	4 (12)	-	56,6
calling task not preempts	4	4 (12)	-	98,6
calling task preempts	4	4 (12)	-	157,4
Context switch (no register save)	4	4 (12)	≈7	58,6 (Reg save)
Time period	10µs - ∞	10µs - ∞	1ms - ∞	1ms - ∞

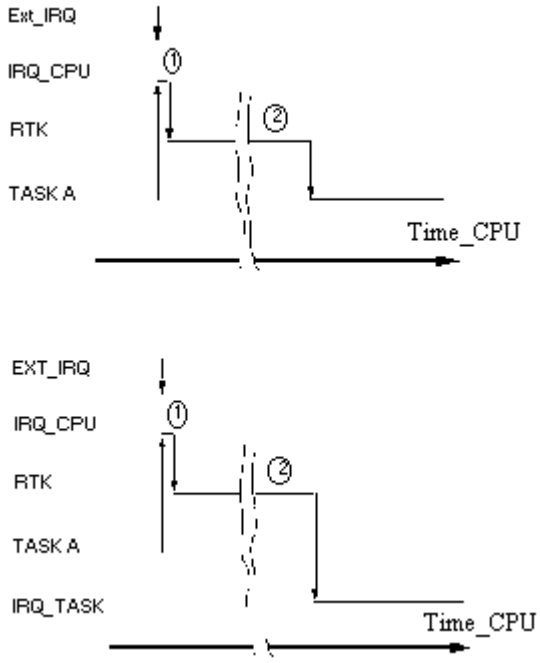
Summary:

- The processing time in a commercial RTK is approx 10 to 20 times longer than for the RTU. In multiprocessor systems it is probably greater differences.
- Parameter transfering is a overhead time for a CPU. Since the RTU uses a bus to transfer parameters, it can be a lot of overheadtime for the CPU in the RTU (M) case. This can be solved by using separately processor buses, which makes bus conflicts impossible.
- Supervisor functions e.g checking task deadlines can easily be done in the RTU, without loading the CPU.

Interrupt handling

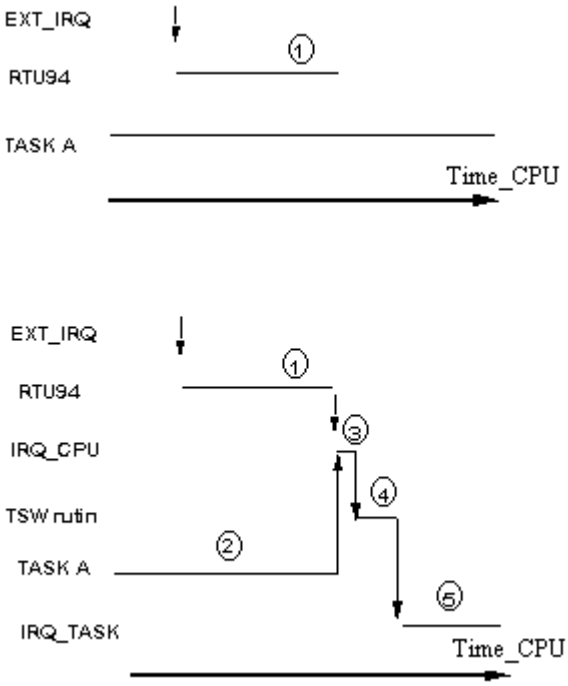
In this section a comparsion between comercial RTK interrupt handlers and the RTU interrupt handler are done.

Figure 7 shows the overhead time for the CPU (when a commercial RTK is used), when an interrupt occurs. Time (1) is the interrupt handler time (approx 10µs) i.e saving registers on stack etc. (2) shows the time, which the RTK uses for scheduling and saving the interrupt in an event list. The top fig. shows the case when the interrupt task has lower priority than the running task. The bottom fig. shows the same event but with an interrupt task with higher priority.



Figur 7: Top figure shows whats happening in the CPU when the interrupt has lower priority than the running task. Bottom figure shows when the interrupt task has higher priority.

Since the RTU has a interrupt handler, all interrupt tasks can be scheduled as all other tasks. The priority of the interrupt task classifies when the task is going to be executed. Figure 8 shows how the RTU does not interrupt the CPU, if an interrupt with lower priority than the executing task has occured. The only time the CPU gets interrupted, is when the interrupt task has higher priority than the executing task. The time from the external interrupt occurs until the interrupt task starts is the sum of (1),(3) and (4). The scheduling of the interrupt task (1) is approx 0,6 - 6,25µs (RTU running at 16 Mhz). Time (3) is the interrupt handler time (approx 10µs) i.e saving registers on stack etc. Restoring a task's registers is time (4), which is approx 10-20µs on a fast CPU. By adding (1),(3) and (4) you get the contextswitch time for interrupt tasks (when the interrupt task has higher priority than the running task). In the multiprocessor case, the same times are valid.



Figur 8: Top figure shows whats happening in the CPU when the interrupt has lower priority than the running task. Bottom figure shows when the interrupt task has higher priority.

The table below shows the performance on interrupt handling on different RTK.

Service Anrop (µs)	RTU94 (S)	RTU94 (M)	EX1 RTK	Ex2 RTK
Interrupt Handeling Function				
Wait for IRQ				
Return to IRQ task	36 (30)	36 (30)	-	79,3
Return to preemted task_a	0	0	-	29,5
IACKL	0	0	12,5	13

Note: The CPU load time is within paranthesis

IACKL is the maximum time an interrupt is disabled.

Summary

- Interrupt handling does not load the CPU, when a RTU is used.
- When the interrupt task has lower priority than the running task, the CPU does not get interrupted if a RTU is used.
- The interrupt handler on the RTU works in a similar way in a multiprocessor environment as in a single processor environment.

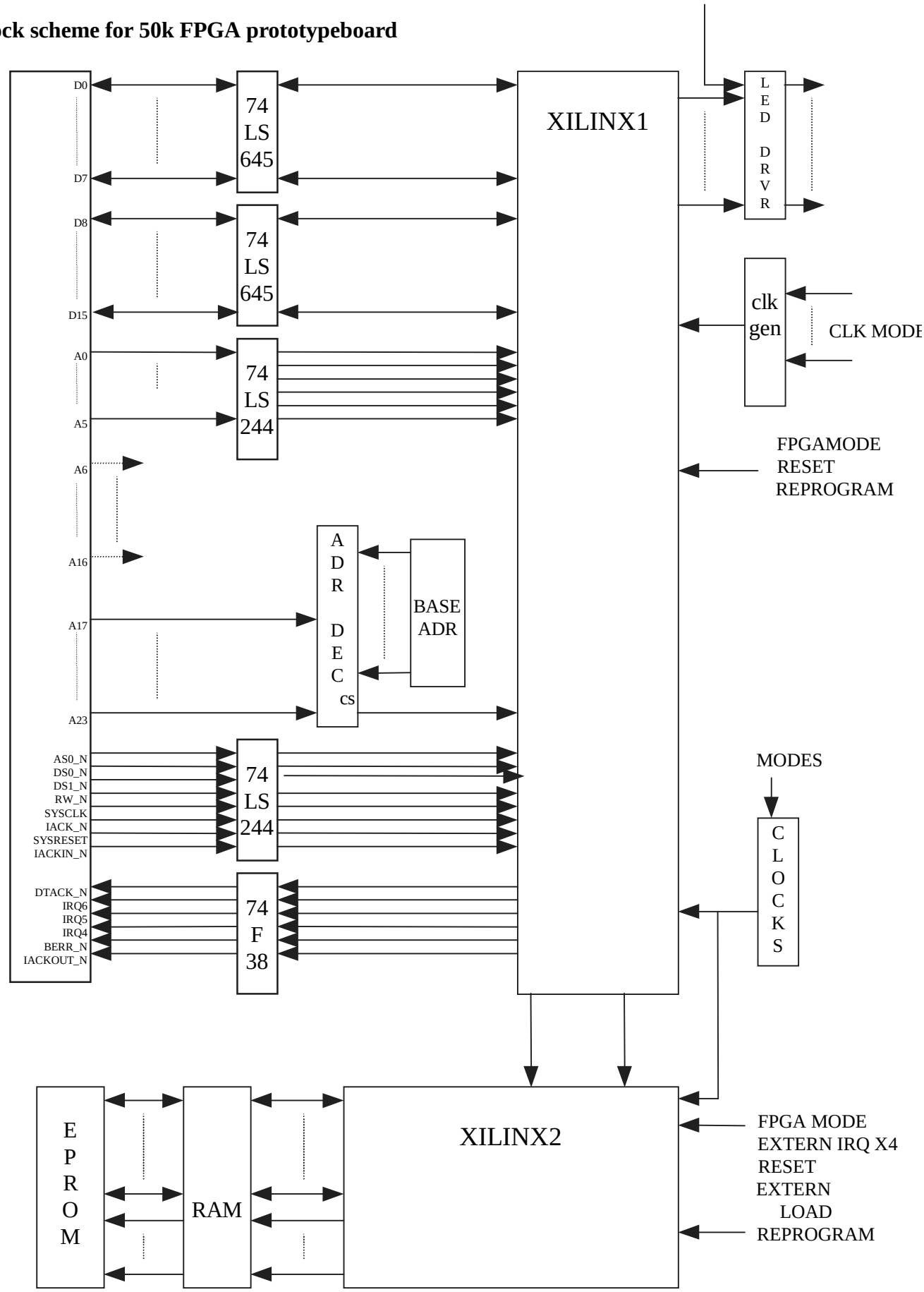
Appendix E

This appendix describes a FPGA prototype board, which can be configured to work as a RTU94 on a VME board. This board can also be used when new RTU prototypes has to be implemented.

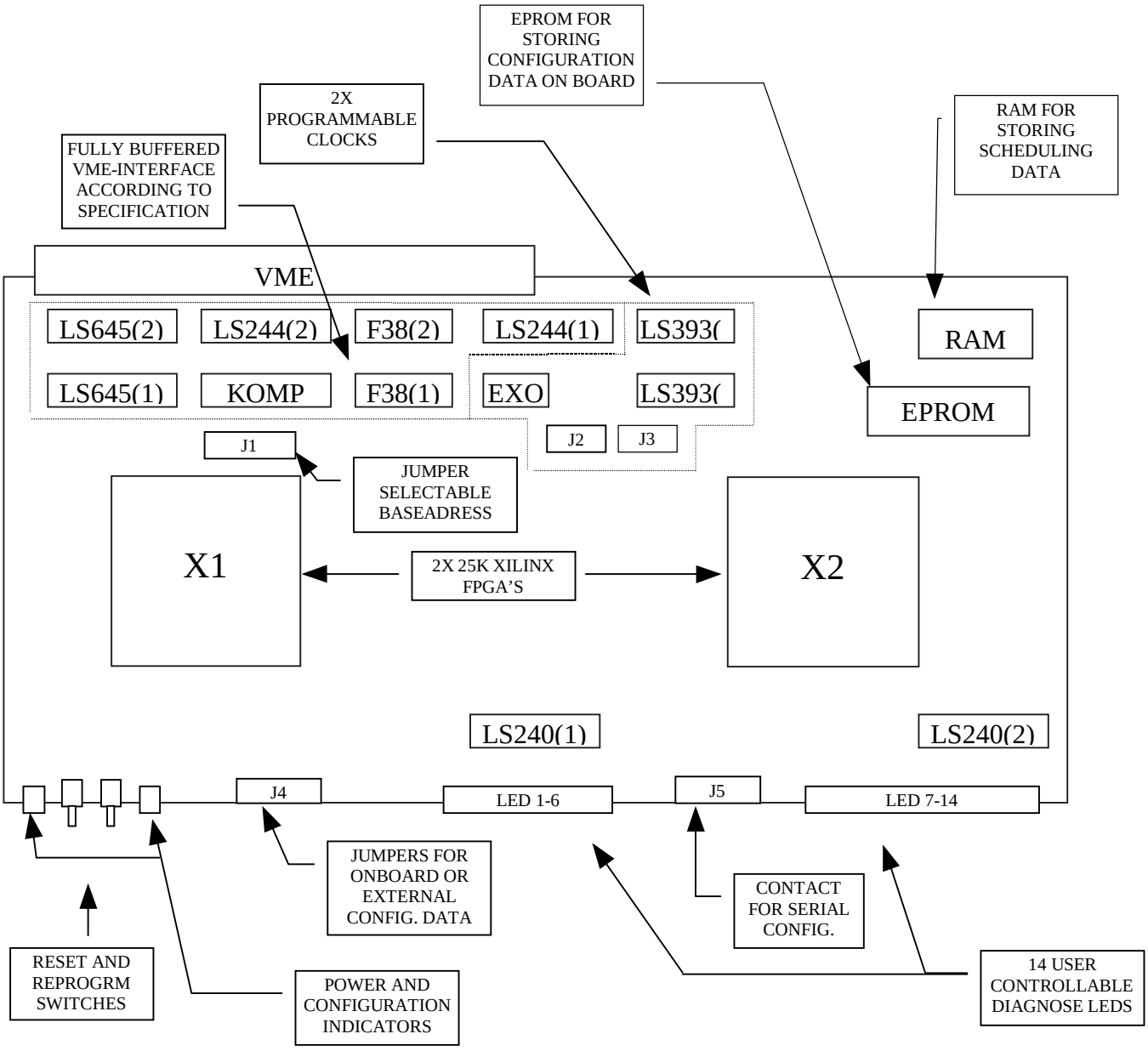
Hardwarespecification for 50k FPGA prototypeboard

* two FPGA's (XILINX 4013/4025)
* two different onboardgenerated clockfrekvencies
* variable fastclock is generated by an EXO-3 jumper programmable clockoscillator
* slow clock(time sync) is generated by dividing 16mhz in 4 4bit binarycounters
* jumper selectable configuration either from a serial link or parallell from an onboard mounted EPROM
* the 16 datasignals are buffered by two bidirectional 74LS645
* A1-A6 connects to X1 via 74LS244 buffers
* A23-A17 are indirectly buffered by the comparator
* controlsignals out from the board are buffered by 74F39
* adressdecode is implemented with a 74HCT688 comparator, A23-A17 are compared with the 7bit jumper programmable baseadress together with AS* for stabile CS*out.
* X2's mode is jumper programmable to either "serial slave mode" for serial link configuration or to "parallell master mode" for loading configuration data from an onboard mounted EPROM
* X1 is configured serially by X2 and the mode is always set to "serial slave mode"
* X2 shall have a 6116 RAM for storing scheduling data
* RESET and REPROGRAM(from EPROM) buttons in the frontpanel
*Jumper to set EPROM in constant tri-state

Block scheme for 50k FPGA prototypeboard



Component placement on 50k FPGA prototypeboard



Appendix F

In this appendix a list of RTU94 related publications are adressed.

• Thesis, Lindh, L: Utilization of Hardware Parallelism in Realizing Real Time Kernels, TRITA-TDE 1994:1, ISSN 0280-4506, ISRN KTH/TDE/FR--94/1--SE, Royal Institute of Technology, Department of Electronics

• A book about today's Real Time Systems, Studentlitteratur,
Christer Eriksson and Lennart Lindh,
"Realtidssystem - grunderna för styrsystem-", Studentlitteratur, ISBN 91-44-28821-2, 1989.

• Measuring and Analysing Real Time Kernel Performance,
H. Berggren, M. Gustafsson and L. Lindh,
Measuring and Analysing Real-Time Kernel Performance, Euromicro 92, Paris (14 - 17 September, 1992).

• Compendium for design methodology,
L. Lindh,
Utvecklingsmetodik för hårdvara, Sweden, Västerås, Department of Real-Time Systems, Internal, 1989.

• The Idea of FASTCHART,
L. Lindh, & F. Stanischewski,
FASTCHART - A Fast Time Deterministic CPU and Hardware Based Real-Time-Kernel, IEEE press, Real-Time Workshop, Paris, 12-14 June, 1991.

• Implementation of FASTCHART**,**
L. Lindh, & F. Stanischewski,
FASTCHART - Idea and Implementation, IEEE press, International Conference on Computer Design (ICCD) Cambridge MIT, USA, 14-16 Oct. 1991.

• Idea of FASTHARD,
L. Lindh,
FASTHARD - A Fast Time Deterministic Hardware Based Real-Time Kernel, IEEE press, Real-Time Workshop, Athens, 3-5 June, 1992.

• Implementation of FASTHARD,
L. Lindh,
FASTHARD Prototype - A Real-Time-Kernel Implemented In One Chip. IEEE press, Real-Time Workshop, Oulu, 3-5 June, 1993.

• Survey of FASTHARD - Real Time Kernel Implemented In special Hardware,
L. Lindh,
Survey of FASTHARD - Real-Time Kernel Implemented In Special Hardware -, 93 års Konferens om Realtidssystem, Stockholm, 25-26 Aug 1993.

• Rapid Prototyping with VHDL, Design Experience,
L. Lindh,
Rapid Prototyping with VHDL, Design Experience, EURO-VHDL 91, Stockholm (8-11 September, 1991), Sweden.

• A Design of a Real Time Unit in Hardware
L. Lindh, & F. Stanischewski,
A Design of a Real-Time Unit in Hardware, Svenska Nationella Arbetsgruppen i Realtid - SNART, Uppsala, 19-20 august, 1991.

• Rapid Prototyping with VHDL and FPGAs,
L. Lindh, Prof K.D. Müller-Glaser and H. Rauch,
Rapid Prototyping With VHDL and FPGAs, International Workshop on Field-Programmable Logic and Application, Vienna, Austria, 31 august - 2 September, 1992.

• Rapid Prototyping with VHDL and FPGAs,
L. Lindh, Prof K.D. Müller-Glaser and H. Rauch,
Rapid Prototyping With VHDL and FPGAs, Lecture notes in Computer Science 705,
Springer-Verlag, ISBN 0-387-57091-8 or ISBN 3-540-57091-8, 1993.

• Experiences with VHDL and FPGAs
L Lindh, J Stärner and J Adomat,
VHDL-Forum EUROPE, Nantes, France, April 24-27, 1995.

• Experiences with VHDL and FPGAs (extended),
L Lindh, J Stärner and J Adomat,
Journal of System Architecture, North Holland 1995.

•"VHDL för konstruktion" (360 pages),
LLindh and S Sjöholm,
Studentlitteratur, ISBN 91-44-47781-3, 1994

• From Single to Multiprocessor Real-Time Kernels in Hardware,
LLindh, J Starner and J Furunäs,
IEEE Real-Time Technology and Applications Symposium, Chicago, May 15 - 17, 1995

• From Single to Multiprocessor Real-Time Kernels in Hardware,
LLindh, J Starner and J Furunäs,
SNART 95 (Svenska Nationella Realtidsföreningen), Göteborg, 22-23 augusti, 1995, 1995

• "VHDL för konstruktion" (ca 600 pages),
LLindh and S Sjöholm,
Studentlitteratur, Dec 1995

• "VHDL for Design" (about 500 pages),
LLindh and S Sjöholm,
Prentice Hall, Våren 1996

References

[Microtec] Microtec manuals which are relevant to this document.
Microtec Research Software Development Tools:
”XRAY68K Debugger”, ”ASM68K Assembler”, ”MCC68K C Compiler”

[FORCE] FORCE manuals which are relevant to this document.
”SYS68K CPU-3VA Hardware user’s manual”,
”SYS68K DRAM-1/2 user’s manual”