SGS-THOMSON MICROELECTRONICS

ST20 TOOLSET
ST20–SWC

ST20 software development and debugging tools

PRODUCT INFORMATION

FEATURES

- ANSI C compiler (X3.159-1989).
- Excellent compile time diagnostics.
- Global and local optimization.
- Assembler inserts and stand alone assembler.
- Support for EPROM programming.
- Support for placing code and data in user specified memory locations.
- Support for dynamically loading programs and functions.
- Small runtime overhead.
- Cross-development from PC and Sun-4 platforms.
- Support for trap and interrupt handlers.

INQUEST Interactive and post-mortem debugging:–

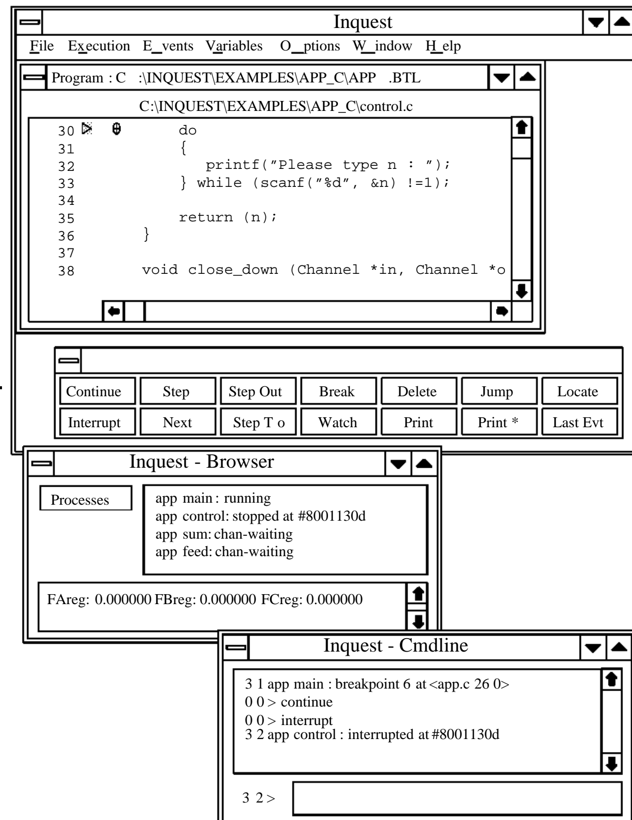
- Windowing interface using X Windows or Windows.
- Programmable command language.
- Source code or assembly code view.
- Stack trace-back facility.
- Variable and Memory display facility.
- C expression interpreter.

INQUEST Interactive debugging:–

- Process and thread break points.
- Single stepping of threads.
- Read/Write/Access watch point capability.
- Facilities to interrupt and find threads.

Performance analysis tools:–

- Analysis of time spent in each function.
- Analysis of block execution frequency.
- Analysis of processor idle time.
- Analysis of processor utilization.



DESCRIPTION

The ST20 ANSI C Toolset provides a complete high quality software development environment for the ST20 microcontroller and microprocessor. The compiler supports the full ANSI C language definition and includes both local and global optimizing features. Embedded application support is provided by both configuration and symbol map utilities.

An *interactive windowing debugger* provides single stepping, breakpoints, watchpoints and many other features for debugging sequential and multi-tasking programs. *Execution profilers* give various post-mortem statistical analyses of the execution of a program.

The information in this datasheet is subject to change

Downloaded from Elcodis.com electronic components distributor

Contents

1 Introduction . . . 3

1.1 Applications . . . 3

2 Code building tools . . . 4

2.1 How programs are built . . . 4
2.2 ANSI C compilation system . . . 5
2.3 Support for embedded applications . . . 6

3 INQUEST windowing debugger . . . 9

3.1 Interactive debugging . . . 12
3.2 Post-mortem debugging . . . 13

4 Execution analysis tools . . . 14

4.1 Execution profiler . . . 14
4.2 Utilization monitor . . . 15
4.3 Test coverage and block profiling tool . . . 15

5 Host interface and AServer . . . 18

5.1 The application loader – `irun` . . . 18
5.2 AServer . . . 18
5.3 AServer features . . . 19

6 ST20 Toolset product components . . . 20

6.1 Documentation . . . 20
6.2 Software Tools . . . 20
6.3 Software libraries . . . 20
6.4 Operating requirements . . . 20
6.5 Distribution media . . . 21

7 Support . . . 21

8 Ordering Information . . . 21

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

1 Introduction

The ST20 ANSI C Toolset provides a complete high quality software development environment for the ST20 microprocessor. The compiler supports the full ANSI C language definition and includes both local and global optimizing features. The run-time library includes both standard C functions, supported by host target connections, and ST20 specific functions to facilitate real-time, multi-tasking and embedded control operations. The real-time and multi-tasking support uses the on-chip hardware micro-kernel and timers, so for many applications no operating system or real-time kernel software is needed.

An interactive windowing debugger provides single stepping, breakpoints, watchpoints and many other features for debugging sequential and multi-tasking programs running on an ST20. Execution analysis tools give post-mortem statistical analyses including execution profiles, processor utilization, test coverage and block profiles.

The host interface is provided by the AServer. This can be used simply as an application loader and host file server, invoked by the **irun** command. The INQUEST tools have their own commands which in turn load **irun** in order to load the application. The AServer may also be used to customize the host interface if required.

1.1 Applications

- Single- and multi-tasking;
- Embedded systems;
- Real-time applications;
- Low cost single chip applications;
- Low level device control applications;
- Porting of existing software and packages.

Downloaded from Elcodis.com electronic components distributor

2 Code building tools

The ST20 ANSI C Toolset provides a complete C cross-development system for the ST20. It can be used to build single task and multi-tasking programs for the ST20. Programs developed with the toolset are both source and binary compatible across all host development machines.

The ST20 ANSI C Toolset is available for the following development platforms:

- IBM 386/486 PC and compatibles under MSDOS 5 and Windows 3.1, or later versions.
- Sun 4 under SunOS 4.1.3 or Solaris 2.4 with X11 Release 4 server or OpenWindows 3, or later versions.

2.1 How programs are built

The toolset build process is shown diagrammatically in Figure 1.

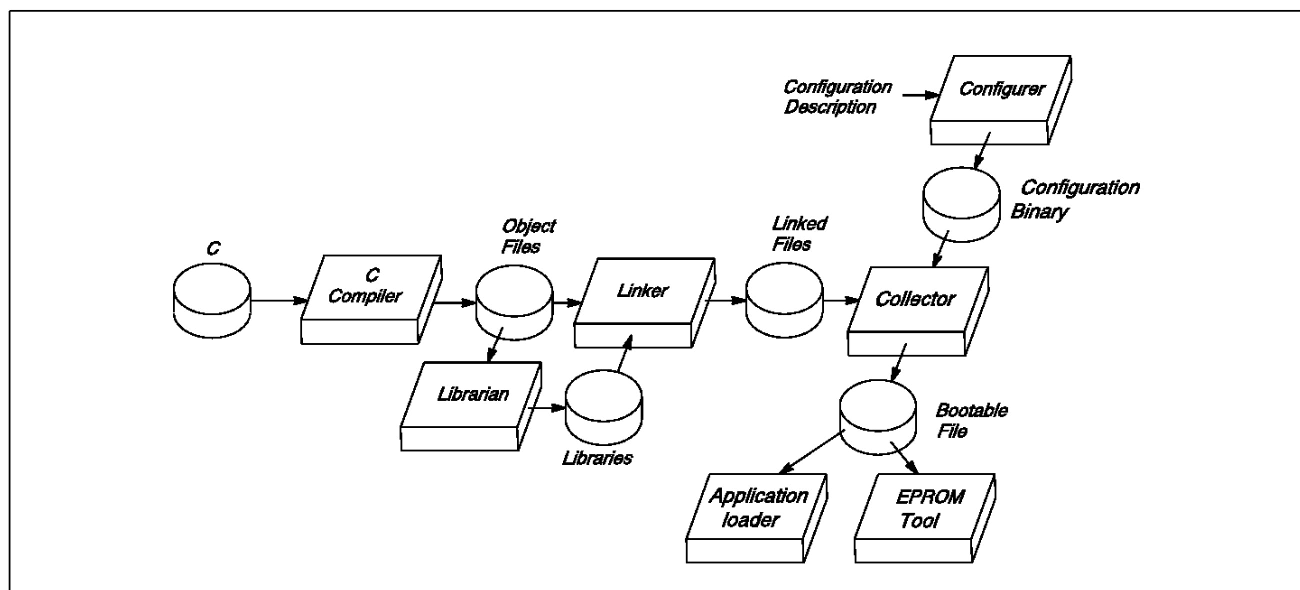


Figure 1 The tool chain

ANSI C source files may be separately compiled into *object files*. The compiler and libraries are described in section 2.2. The *librarian* may be used to collate object files into libraries. The *linker* links object files and libraries into fully resolved linked units.

A *configuration description* is a text file describing the target hardware and how the software maps onto it. The *configurer* converts the configuration description file into a *configuration binary* file. The *collector* removes any debugging information, and uses the configuration binary to collect the linked files with bootstrap code to make an executable file called a *bootable file*.

During development and for hosted systems, the bootable file may be loaded down a hardware serial link onto the target hardware using the *application loader*. For stand-alone systems, the bootable file may be converted, using the EPROM tool, to an industry standard EPROM format for programming EPROMs.

In addition to loading programs down a hardware serial link, the application loader program provides access to host operating system facilities through a remote procedure call mechanism. This method is used to support the full ANSI C run-time library.

A memory configurer tool is supplied for describing a ST20450 memory configuration. This data is used to initialize the memory interface of the ST20450. The memory interface can be initialized either using the hardware serial link or from ROM.

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

2.2 ANSI C compilation system

2.2.1 Compiler operation

The compiler operates from a host command line interface. The preprocessor is integrated into the compiler for fast execution. The compile time diagnostics provided by the compiler are comprehensive and include type checking in expressions and type checking of function arguments.

2.2.2 ANSI conformance

The ST20 ANSI C Toolset supports the full standard language as defined in X3.159-1989. The compiler passes all the tests in the validation suites from Plum Hall and Perennial.

2.2.3 Local optimized code generation

The compiler implements a wide range of local code optimization techniques.

Constant folding. The compiler evaluates all integer and real constant expressions at compile time.

Workspace allocation. Frequently used variables are placed at small offsets in workspace, thus reducing the size of the instructions needed to access them, and hence increasing the speed of execution.

Dead-code elimination. Code that cannot be reached during the execution of the program is removed.

Peephole optimization. Code sequences are selected that are the fastest for the operation.

Constant caching. Some constants have their load time reduced by placing them in a constant table.

Unnecessary jumps are eliminated.

Switch statements can generate a number of different code sequences to cover the dense ranges within the total range.

Special idioms that are better on ST20s are chosen for some code sequences.

2.2.4 Globally optimized code generation

The ANSI C globally optimizing compiler extends the types of optimizations it performs to global techniques. These have typically given a 15–25 percent improvement in speed over the local optimizations as measured by a suite of internal benchmarks.

Common sub-expression elimination removes the evaluation of an expression where it is known that the value has already been computed; the value is stored in temporary local workspace. This improves the speed of a program and reduces code size.

Loop-invariant code motion can move the position where an expression is evaluated from within a loop to outside it. If the expression contains no variables that are changed during the execution of a loop, then the expression can be evaluated just once before the loop is entered. By keeping the result in a temporary, the speed of execution of the whole loop is increased.

Tail-call optimization reduces the number of calls and returns executed by a program. If the last thing a function does is to invoke another function and immediately return the value there from, then the compiler attempts to re-use the same workspace area by just jumping to (rather than calling) the lower level function. The called function then returns directly to where the upper level function was called from. In the case where the call is a recursive call to the same function, then the workspace is exactly the right size, and a saving is also made because the stack adjustment instructions are no longer needed either. This optimization saves speed and total stack space.

Downloaded from Elcodis.com electronic components distributor

Workspace allocation by coloring reduces the amount of workspace required by using the same word for two variables when it can be determined that they are not both required at the same time.

The optimizing compiler also implements a pragma, `IMS_nosideeffects`, whereby the user can indicate that a function does not have any side-effects on external or static variables. The optimizer can then use this information to make assumptions about what state can be changed by a function invocation and hence about the validity of any previously computed expressions.

2.2.5 Libraries

The full set of ANSI libraries is provided.

The standard library operates in double precision. Versions of the mathematical functions are provided that operate on float arguments and return float values. These libraries provide improved performance for applications where performance requirements override accuracy requirements.

A reduced C library is supplied to minimize code size for embedded systems applications. This library is appropriate for code which does not need to access host facilities.

Collections of functions can be compiled separately with the ANSI C compiler and then combined into a library. The linker is used to combine separately compiled functions into a program.

2.2.6 Assembler inserts

The ANSI C Toolset provides a very powerful assembler insert facility. The assembler insert facility supports:

- Access to the full instruction set of the ST20;
- Symbolic access to C variables (automatic and static);
- Pseudo operations to load values into registers;
- Loading results of C expressions;
- Labels and jumps;
- Directives for instruction sizing, stack access, return address access etc.

2.3 Support for embedded applications

The toolset has been designed to support the development of embedded applications. The features include the ability to place code and data at particular places in memory, being able to access the ST20 instruction set efficiently from C, and to reduce the C run-time overhead to suit the application.

2.3.1 Placing code and data

At configuration level, a program consists of its code, stack, static and heap segments. The configurer allocates each of these separate segments a place in the memory of the processor.

By default, the configurer allocates all the code and data segments to a contiguous default block of memory. The location of this default block and the order of the segments may be defined in the configuration description. The configuration description can also specify that any one of the code or data segments of an application are to be allocated to particular places in memory outside the default block.

The compiler, linker and collector each will optionally produce a listing of how the various parts of an application are mapped into the segments and memory. A tool is provided that can read all these map files and produce a summary of the whole application, giving the locations of all the functions and static variables. Information is collated about code and data segments including the start address and size.

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

2.3.2 Access to the instruction set

ANSI C is a good language for writing embedded applications, since it combines the constructs of a high level programming language with low level access to the hardware through assembler inserts.

To make the access to some of the ST20 instructions even more effective, a number of special library functions have been defined which the optimizing compiler can render as in-line code. This removes the overhead of a library call, but it also gives the optimizer more information on what the program is doing.

Normally, when the optimizer sees a function containing some assembler code, it must make very conservative assumptions about the effect the code has on its surroundings, e.g. on static variables and parameters. By using the functions defined to access the instructions, the optimizer knows exactly what the effects will be and can make the correct assumptions for the side-effects of the code.

The ST20 instructions that can be accessed in this way include block moves, channel input and output, bit manipulation, CRC computations, semaphores and some scheduling operations.

2.3.3 Run-time overhead reduction

In order to support the full ANSI C language, a significant run-time library is necessary. The toolset is supplied with another library, known as the *reduced library*, which does not support file system and environment requests, which depend on a host.

If some of the other features in ANSI C are not used, then it may be possible to reduce the overhead further by modifying the run-time initialization, the source of which is provided.

2.3.4 Assembler inserts

Within the SGS-THOMSON implementation of the ANSI C language, assembler code can be written at any point to achieve direct access to ST20 instructions for reasons of speed and code size. Full access is available to the ST20 instruction set and C program variables.

2.3.5 Assembler

If there is no other way to obtain the code required, for example when writing customized bootstrap mechanisms, then the toolset contains an assembler as the final phase of the compiler. This can be invoked to assemble user-written code.

2.3.6 Dynamic loading of processes

The toolset can encapsulate the code and data of a process in a file called a *relocatable separately-compiled unit* or `rsc`. This form is suitable for it to be loaded by another application and called. A set of functions is provided for this to be achieved. The `rsc` can be found either in a file, or already in memory, or is input along a channel. The memory variant allows an `rsc` to be placed into ROM and executed if required. For example, if an application wishes to select a device driver to be placed in on-chip memory, then a number of possible drivers can be placed in ROM and the application can choose one for the occasion, copy it into low memory and execute it.

2.3.7 Bootstraps

The source code of the standard bootstraps are provided. The user can then write bootstraps that are tailored to a specific application by using the standard ones as templates.

2.3.8 Memory configuration

An interactive memory configurer tool is supplied for describing a ST20450 memory configuration. The memory interface of the ST20450 is configurable and must be initialized before the memory can be accessed. This data is used by the Toolset tools to generate the code to initialize the memory

Downloaded from Elcodis.com electronic components distributor

interface of the ST20450. This code is either downloaded from the host using the hardware serial link using the host server or is incorporated into a ROM using the EPROM programming tool.

For an ST20 variant with a memory interface different from the ST20450, example bootstrap code is provided.

2.3.9 Interrupt and trap handling

Library support is supplied for trap handlers and ST20450 interrupt handlers. This allows trap handlers and ST20450 interrupt handlers to be written in C which will use the on-chip trap handling and interrupt handling support. Interrupt handlers for other ST20 variants may be based on the provided libraries.

Downloaded from Elcodis.com electronic components distributor

3 INQUEST windowing debugger

The INQUEST debugger can debug ANSI C ST20 programs either interactively or post-mortem. A user interface displays source code or disassembled code and enables the user to interact with the debugger by means of buttons and menus, mouse and keyboard. The interface is built using the X Window System and OSF/Motif for Sun-4s or Microsoft Windows for PCs.

The program being debugged may consist of any number of tasks (or threads of execution) some of which may be running while others are stopped or being single stepped. The host debugger program is asynchronous and holds a copy of the last stopped state of each thread, so values may be inspected by the host while the user program is running on the ST20. Multiple debugging windows may be opened to view different parts of the program simultaneously.

The INQUEST debugger has three debugging modes:

- interactive debugging, i.e. monitoring the application as it executes on the target processor;
- post-mortem debugging on the target processor when the application has stopped;
- post-mortem debugging on the host from a dump file.

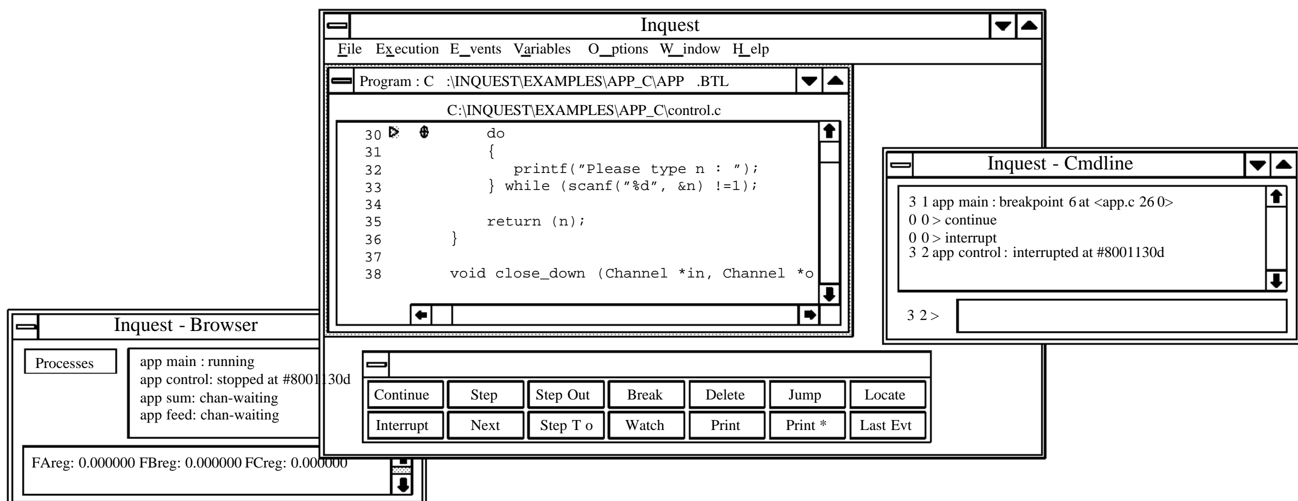


Figure 2 The Microsoft Windows debugger display

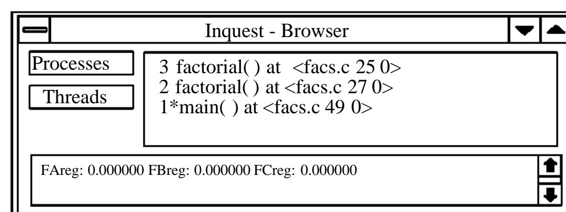


Figure 3 A Microsoft Windows stack trace

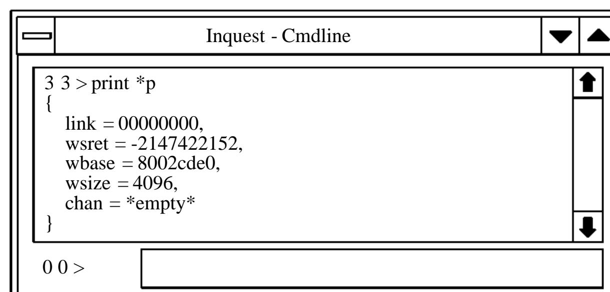


Figure 4 Displaying a structured variable on Microsoft Windows

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

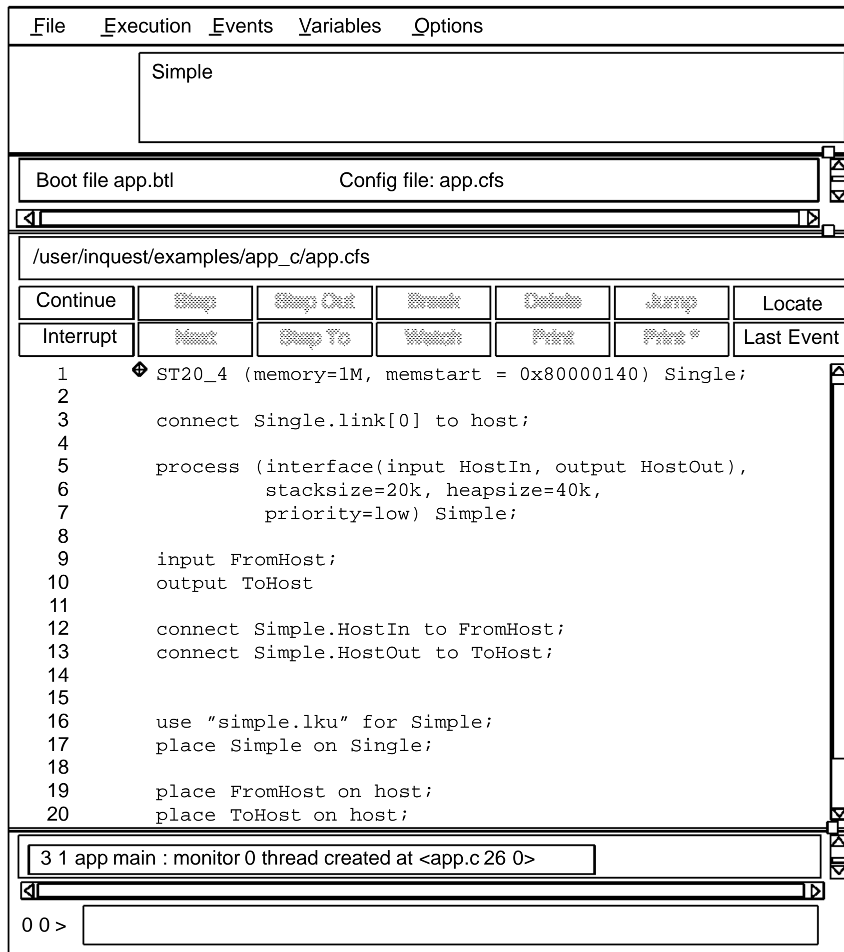


Figure 5 The X-Windows debugger display

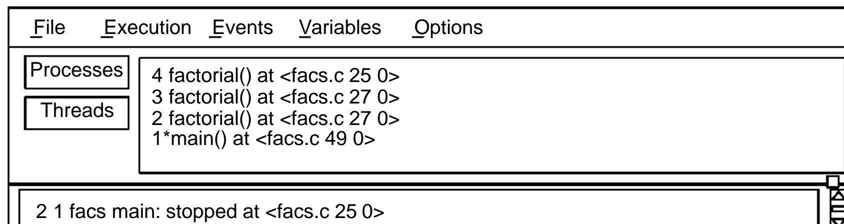


Figure 6 An X-Windows stack trace

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

```
31                ChanOutInt(outfeed, 0);

3 3 > print *p
{
  link = 00000000,
  wsret = -2147422152,
  wbase = 8002cde0,
  wsize = 4096,
  wp = 8002ddc4,
  psize = 4,
  ofunc = 80010650,
  chan = *empty*
}

3 3 >
```

Figure 7 Displaying a structured variable on X-Windows

```
File   View   Options

Processor:  0
Start address:  0x8000a098          End address:  0x8000a31c
Format:  Hexadecimal          Type:  Word

#8000a098:  0x00000000  0x00000001  0x00000001  0x8003b5e4  0x8003b5dc  0x8000fbd1
#8000a0b0:  0x80014fdc  0x00000001  0x8000a0d4  0x00000000  0x00000000  0x00000000
#8000a0c8:  0x00000000  0x00000000  0x80014fdc  0x8000fbd8  0x00000000  0x8003b95c
#8000a0e0:  0x80014fdc  0x8003b8d8  0x8171f824  0xfa7671d1  0x80000000  0x80000000
#8000a0f8:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a110:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a128:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a140:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a158:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a170:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a188:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a1a0:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a1b8:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a1d0:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a1e8:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a200:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a218:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a230:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a248:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a260:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a278:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a290:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a2a8:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a2c0:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a2d8:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a2f0:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000
#8000a308:  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000  0x80000000

Processor:  Single     Type:  ST20          Memory:  2048k
```

Figure 8 Displaying memory contents on X-Windows

Downloaded from Elcodis.com electronic components distributor

3.1 Interactive debugging

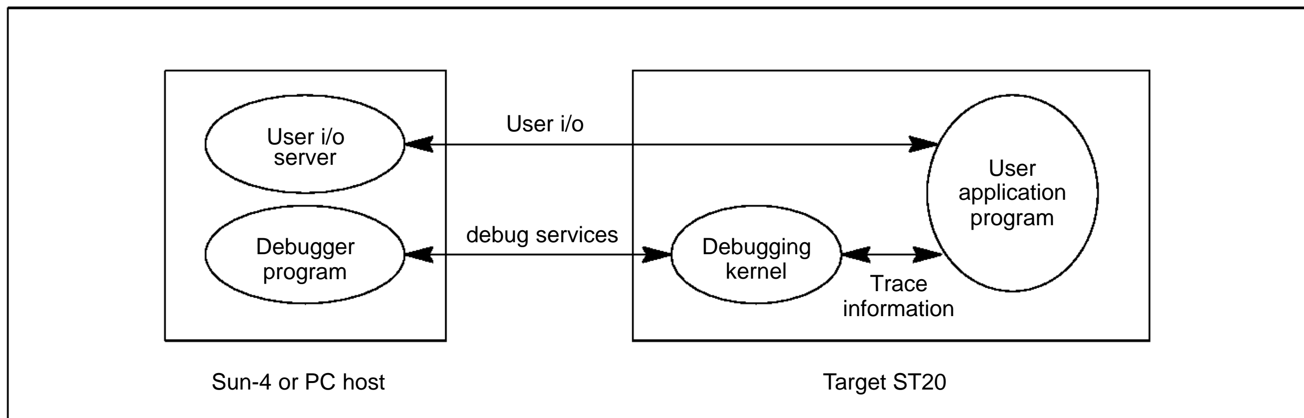


Figure 9 Interactive debugger architecture

The interactive debugger consists of a host-based symbolic debugger and a target-resident debugging kernel that is configured into the application program on an ST20 processor.

The interactive debugger provides the following features:–

- A break point facility that can be used on particular threads of execution.
- A single stepping facility that allows a thread of execution to be single stepped at the source level or at the assembly code level.
- A watch point facility that enables the program to be stopped when variables are to be written to or read from.
- A facility to find the threads of execution of a program and set break points on them.
- A stack trace facility.
- A facility to monitor the creation of threads of execution.
- Commands to print the values of variables and display memory.
- Commands to modify variables and memory at run time.
- A simple interpreter to enable C aggregate types (i.e. structures and arrays) to be displayed.
- A programmable command language that allows complex break points and watch points to be set and enables debugging scripts to be generated.
- A source and object browser to select a process, a thread and source code.
- An interface to the real-time operating system for dynamic code loading and thread creation.

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

3.2 Post-mortem debugging

The post-mortem debugger provides the following features:–

- A source and object browser to select a process, a thread and source code.
- Commands to print the values of variables and display memory.
- A simple interpreter to enable C aggregate types (i.e. structures and arrays) to be displayed.
- Creation of dump files.
- Debugging from a dump file.

Downloaded from Elcodis.com electronic components distributor

4 Execution analysis tools

Three profiling tools are supplied for analyzing the behavior of application programs; the execution profiler, the utilization monitor, and the test coverage and block profiling tool. The monitoring data is stored in the target processor's memory, so the profiling tools have little execution overhead on the application. After the program has completed execution, the monitoring data is extracted from the processor and is analyzed to provide displays on the program execution.

The tools provided are the execution profiler, the utilization monitor, and the test coverage and block profiling tool. The execution profiler estimates the time spent in each function and procedure, the processor idle time and various other statistics. The utilization monitor displays a Gantt chart of the CPU activity of the processor as time progresses. The test coverage and block profiling tool counts how many times each block of code is executed.

4.1 Execution profiler

The execution profiler gives an analysis of the total time spent executing each function on each processor.

It provides the following information on program execution:–

- The percentage time spent executing each low priority function.
- The percentage time spent executing at high priority.
- The percentage idle time of the processor.
- The number of low priority calls of each function and where it was called from.

```
Processor "Root"
Idle time 35.3% (19516)
High time 0.1% (37)
Wptr Misses 0
Iptr Misses 0
Resolution 4

-----------------------------------------------------------------------------
Process "ex" (99.9% processor) (35.666s)
Stack 100.0% (35666)    Heap 0.0% (0)    Static 0.0% (0)
Function Name                                   | Process | Processor |Samples
-----------------------------------------------------------------------------
libc.lib/getc                                   |    11.4 |      11.4 |4081
cc/pp.c/pp_rdch0                                |    10.1 |      10.1 |3605
cc/bind.c/globalize_memo                        |     6.9 |       6.9 |2467
cc/pp.c/pp_process                              |     4.3 |       4.3 |1525
cc/pp.c/pp_rdch3                                |     4.2 |       4.2 |1497
cc/pp.c/pp_rdch2                                |     3.9 |       3.9 |1380
cc/pp.c/pp_rdch1                                |     3.8 |       3.8 |1354
cc/pp.c/pp_rdch                                 |     3.5 |       3.5 |1252
cc/pp.c/pp_nextchar                             |     3.3 |       3.3 |1189
cc/pp.c/pp_checkid                              |     3.2 |       3.2 |1150
cc/lex.c/next_basic_sym                         |     2.7 |       2.7 |979
libc.lib/strcmp                                 |     2.3 |       2.3 |812
libc.lib/DummySemWait                           |     2.2 |       2.2 |784
libc.lib/sub_vfprintf                           |     1.7 |       1.7 |617
```

Figure 10 Example output from the execution profiler

The execution of the user program is monitored by a profiling kernel. The presence of the profiler kernel will slow the execution of the program by less than 5%.

4.2 Utilization monitor

The utilization monitor shows in graphical form the utilization of the processor over the time of the program execution. This is displayed by an interactive program that draws a chart of processor

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

execution against time using X Window System and OSF/Motif on a Sun-4 or Microsoft Windows on a PC.

As with the execution profiler, the user program is monitored by a profiling kernel. The kernel will slow the program by less than 5%.

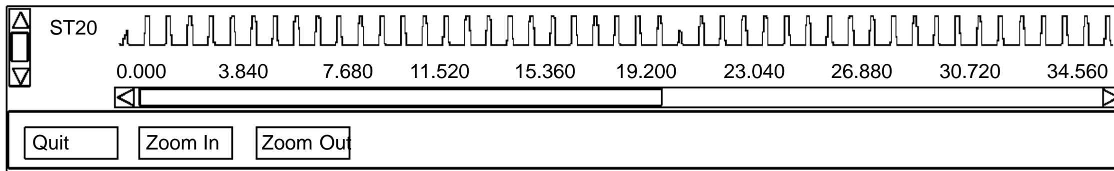


Figure 11 Example X-Windows display from the utilization monitor

4.3 Test coverage and block profiling tool

This tool monitors test coverage and performs block profiling for an application which has been run on target hardware.

This tool is able to:

- provide an overall test coverage report;
- provide per module test coverage reports;
- accumulate a single report from multiple test runs;
- provide a detailed basic block profiling output by creating an annotated program listing;
- provide output that can be fed back into the compiler as a part of its optimization process.

The application program (compiled with the appropriate compiler option) is run and accumulates the counts in the memory of the target processor. The tool is used to extract the results and save, accumulate or display them. This application writes the counts into the code area, so the tool cannot be used with code running from ROM.

```
Writing coverage file ”square.v” – 40% coverage
Writing coverage file ”comms.v” – 14% coverage
Writing coverage file ”app.v” – 75% coverage
Writing coverage file ”control.v” – 36% coverage
Writing coverage file ”feed.v” – 33% coverage
Writing coverage file ”sum.v” – 40% coverage
Total coverage for bootable 39% over 1 run
```

Figure 12 Example test coverage summary report

The following is an example of the contents of a coverage file:

```
|/*
| * facs.c
| *
| * generate factorials
| *
| */
|
|#include <stdio.h>
|#include <stdlib.h>
|#include <process.h>
|#include <channel.h>
|#include <misc.h>
|#include ”comms.h”
|
|#define TRUE 1
|#define FALSE 0
|
|
```


Downloaded from Elcodis.com electronic components distributor

```
                  |        /*
                  |         * compute factorial
                  |         *
                  |         */
                  |
                  |int factorial(int n)
96                |{
                  |  if (n > 0)
74                |    return ( n * factorial(n-1));
                  |  else
22                |    return (1);
                  |
                  |}
                  |
                  |int main()
1                 |{
                  |   Channel *in, *out;
                  |   int going = TRUE;
                  |
                  |   in = get_param(1);
                  |   out = get_param(2);
                  |
                  |   while (going)
27                |   {
                  |      int n, tag;
                  |
                  |      tag = read_chan (in, &n);
                  |      switch (tag)
                  |      {
                  |         case DATA: {
22                |            send_data (out, factorial(n));
                  |            break;
                  |         }
                  |         case NEXT: {   /* start a new sequence */
4                 |            send_next (out);
                  |            break;
                  |         }
                  |         case END: {    /* terminate */
1                 |            going = FALSE;
                  |            send_end (out);
                  |         }
                  |      }
                  |      }
                  |   }
                  |}
                  |
```

```
######################
# Summary of results #
######################
Source file       : facs.c
Number of runs    : 1
Processors        : All
>From linked unit : facs.lku

Top 10 Blocks!!

Line 25 - 96 times
Line 27 - 74 times
Line 42 - 27 times
Line 29 - 22 times
Line 49 - 22 times
Line 53 - 4 times
Line 34 - 1 time
Line 57 - 1 time

Total number of basic blocks 8
Basic blocks not executed    0
Coverage 100%
```


Downloaded from Elcodis.com electronic components distributor

5 Host interface and AServer

The host interface is provided by the AServer. This can be used simply as an application loader and host file server, invoked by the **irun** command. The INQUEST tools have their own commands which in turn load **irun** in order to load the application. The AServer may also be used to customize the host interface if required.

5.1 The application loader – irun

irun performs three functions, namely:

1. to initialize the target hardware;
2. to load a bootable application program onto the target hardware via the hardware serial link;
3. to serve the application, i.e. to respond to requests from the application program for access to host services, such as host files and terminal input and output.

These steps are normally performed when **irun** is invoked.

5.2 AServer

The AServer (Asynchronous Server) system is a high performance interface system which allows multiple processes on a target device to communicate via a hardware serial link with multiple processes on some external device. The AServer software acts as a standard interface which is independent of the hardware used. A simple example is shown in Figure 13, in which the external device is the host.

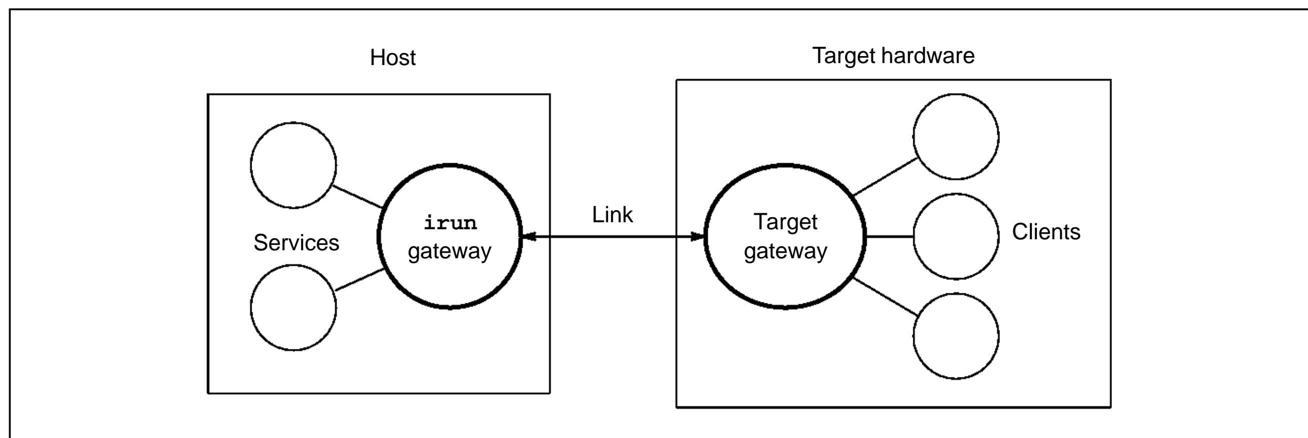


Figure 13 A simple software host-target interface

The AServer is a collection of programs, interface libraries and protocols that together create a system to enable applications running on target hardware to access external services in a way that is consistent, extensible and open. The software elements provided are:

- a target gateway which runs on the target;
- an **irun** gateway which runs on the host;
- an **iserver** service which runs on the host;
- an **iserver** converter which runs on the target;
- a library of interface routines for use by client and service processes;
- simple example services.

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

5.3 AServer features

This type of architecture offers a number of advantages:

1. The AServer handles multiple services.

   A number of services may be available on one device, handled by a single gateway. For example, new services may be added without modifying a standard server.

2. The AServer handles multiple clients.

   Any process on any processor in the target hardware may open a connection to any service. The process opening the connection is called the client. Several clients may access the same service. The gateway will automatically start new services as they are requested.

3. Services are easy to extend.

   The AServer enables users to extend the set of services that are available to a user's application. The AServer provides the core technology to allow users to create new services by providing new processes. For example, the **`iserver`** service provides terminal text i/o, file access and system services, which may be expanded by adding new AServer service processes such as a graphics interface.

4. AServer communications can be fast and efficient.

   The communications over the link between gateways use mega-packets, which make efficient use of the available bandwidth. Messages between the client and the service are divided into packets of up to 1 kbyte. The packets are bundled into mega-packets to send over the hardware serial link. Packets from different clients and services can be interleaved to reduce latency.

5. AServer communications are independent of hardware.

   When an AServer connection has been established the process can send data messages of arbitrary length to the service it is connected to, receive data messages of arbitrary length and disconnect from the service. The gateways are responsible for building and dividing mega-packets and complying with hardware protocols.

SGS-THOMSON MICROELECTRONICS

Downloaded from Elcodis.com electronic components distributor

6 ST20 Toolset product components

6.1 Documentation

- Toolset User Guide
- INQUEST Debugger Tutorial
- Toolset Reference Manual
- Language and Libraries Reference Manual
- INQUEST User and Reference Manual
- Delivery Manual
- AServer Programmers' Guide

6.2 Software Tools

- **`icc`, `ilink`, `ilibr`** – ANSI C compiler, linker and librarian
- **`ilist`**, **`imap`** – Binary lister program and memory map lister
- **`icconf`**, **`icollect`** – Configuration tools
- **`ieprom`**, **`imem450`** – EPROM and ST20450 memory interface programming tools
- **`irun`** – Application loader
- **`inquest`** – Interactive and post-mortem debugger
- **`imon`**, **`iprof`**, **`iline`** – Execution analysis tools
- AServer – Customizable host interface

6.3 Software libraries

- Full ANSI library plus multi-tasking support
- Reduced library for embedded systems
- Run-time start-up support library
- Debugging support library
- AServer library

6.4 Operating requirements

6.4.1 Sun-4 Toolset

The Sun-4 ST20 Toolset package will run on a Sun-4 SPARC-based workstation, server or compatible, running SunOS 4.1.3 or Solaris 2.4 or later. To support the graphical user interface, the environment must include an X Window System server (X11R4 or later) such as an X terminal or SPARCstation running OpenWindows 3.

The toolset is designed to operate in conjunction with an IMS B300 Ethernet Gateway acting as interface to the ST20 development hardware, such as a ST20450 Development Board.

Downloaded from Elcodis.com electronic components distributor

6.4.2 PC Toolset

The PC ST20 Toolset will run on an IBM 386 PC or IBM 486 PC or compatible, running MS-DOS 5.0 or compatible. To support the graphical user interface, the environment must include Microsoft Windows 3.1.

The PC Toolset is designed to operate in conjunction with any of the following hardware development systems or compatible hardware:

- a ST20 Evaluation Board using the PC parallel port;
- a ST20450 Development Board and a PC parallel port interface;
- an Ethernet connection to an IMS B300 Ethernet Gateway and ST20 development hardware, such as a ST20450 Development Board.

6.5 Distribution media

Sun-4 software is distributed on 60 Mbyte QIC-24 $^{1}/_{4}$ inch data cartridges, in tar format. PC software is distributed on 3.5 inch high density (1.44 Mbyte) diskettes.

7 Support

ST20 development products are supported worldwide through SGS-THOMSON Sales Offices, Regional Technology Centers, and authorized distributors.

8 Ordering Information

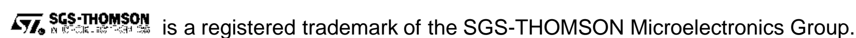
Description	Order number
ST20 Toolset for 386 PC.	ST20–SWC/PC
ST20 Toolset for Sun 4.	ST20–SWC/SUN

Downloaded from Elcodis.com electronic components distributor

Information furnished is believed to be accurate and reliable. However, SGS-THOMSON Microelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of SGS-THOMSON Microelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. SGS-THOMSON Microelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of SGS-THOMSON Microelectronics.

E 1995 SGS-THOMSON Microelectronics - All Rights Reserved

IMS and DS-Link are trademarks of SGS-THOMSON Microelectronics Limited.



is a registered trademark of the SGS-THOMSON Microelectronics Group.

X Window System is a trademark of MIT.

OSF/Motif is a trademark of the Open Software Foundation, Inc.

Windows is a trademark of Microsoft Corporation.

SGS-THOMSON Microelectronics GROUP OF COMPANIES

Australia - Brazil - France - Germany - Hong Kong - Italy - Japan - Korea - Malaysia - Malta - Morocco -
The Netherlands - Singapore - Spain - Sweden - Switzerland - Taiwan - Thailand - United Kingdom - U.S.A.

Downloaded from Elcodis.com electronic components distributor