dSPACE Software Environment in the ARO-DURIP Facility

Technical Report of the ISIS Group
at the University of Notre Dame
ISIS-97-003
March, 1997

Xenofon D. Koutsoukos
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

Interdisciplinary Studies of Intelligent Systems dSPACE Software Environment in the ARO-DURIP facility

February, 1997.

Xenofon D. Koutsoukos

1 Introduction

This documentation is intended to give an overview of the basic DSP software, which is used to implement applications on the controller board DS1102. The document is organized into three sections. The first section describes the necessary settings of environment variables and paths to make dSPACE tools directly accessible from a user's local environment. The second section is an operation tutorial covering the basic steps of establishing the connection between a host workstation and the client dSPACE box, downloading and executing application programs on the DS1102, and monitoring the operation of the DSP controller. A simple example describing the basic steps of the procedure is given. The final part gives an overview of the basic software, which is used to write C programs for the DS1102 controller board.

2 Environment Variables and Paths

To make the dSPACE tools directly accessible some paths and environment variables have to be added to a user's local environment. Log in as a "normal" user and insert the following shell code fragment at an appropriate position into your local start-up file (*.cshrc*, or *.login* file).

```
#
# 'dspace.sh'
# this shell script should be included in every user's login script
# generated: Mon Dec 9 11:30:23 EST 1996
#
# NOTE: This script is for 'csh' shell users !
#

set path=( $path /usr/Dspace/dsp_cit/bin /usr/Dspace/dsp_cit/c31)

setenv DSPACE_ROOT       /usr/Dspace/dsp_cit
setenv DSPACE_CONFIG_DIR /usr/Dspace/dsp_cit/matlab/local
```

```
setenv TI_ROOT          /usr/Dspace/dsp_cit/c31
setenv C_DIR            /usr/Dspace/dsp_cit/c31
setenv A_DIR            /usr/Dspace/dsp_cit/c31

setenv ND_PATH     /usr/Dspace/dsp_cit/bin
setenv CITFILES    $HOME/citfiles
setenv DSP_NAME_SERVER rottweiler

# either set or augment MATLABPATH.

set MP_DEFINED = (`env | grep MATLABPATH`)
if ( "$MP_DEFINED" == "" ) then
  setenv MATLABPATH "$DSPACE_CONFIG_DIR":"$DSPACE_ROOT"/matlab/mlib
else
  setenv MATLABPATH "$MATLABPATH":"$DSPACE_CONFIG_DIR":"$DSPACE_ROOT"/matlab/mli
b
endif

# end of 'dspace.sh'
```

You will find the proposed shell code fragment *dspace.sh* in the *usr/Dspace/dsp_cit/bin* directory. Finally, create the *citfiles* subdirectory, be sure that you are consistent with the definition of the *CITFILES* envinonment variable. For more information refer to [1].

3 Operation Tutorial

The procedure to execute an application on the dSPACE controllers consists of the following steps:

- Start the DSP Daemon to establish the connection between the host computer and the client DSP controller.
- Register the DSP boards at the daemon.
- Compile and download the file to the DSP controller.
- Monitor the results using the *trace* software.
- Terminate the DSP Daemon.

In order to compile and download application programs written in C, to a dSPACE DS1102 processor board without invoking all the necessary tools separately, the batch shell script file *down31* has been developed.

The script file *down31* is invoked by using the following syntax:

```
down31 c_file [options] [-h]
```

where the options are the following

- **-b board name** If *down31* is invoked without this option the default board name *ds1102* is used.
- **-c client** specifies the target network client.
- **-d** Disables downloading. The specified C source file is compiled and linked only.
- **-g** Symbolic debug information is generated.
- **-o** Enables full optimization.
- **-l slave_appl** The specified slave-DSP application object module is loaded with the main-DSP object module.
- **-h** A list of available options is displayed on the screen.

The command line options may be specified in an arbitrary sequence while the C source filename (without a filename extension) must always be the first command line parameter.

Important: If the shell script file *down31* is invoked before the corresponding DSP board is registered at the DSP daemon, the board will not be accessible by the application. The DSP boards need to be registered only once and this can be done using the *dcont* daemon controller described later in the section.

For more information about the shell script file *down31* look at [2].

3.1 DSP Daemon Environment

The DSP daemon environment is mainly intended to provide capabilities for quasi concurrent accesses of programs to dSPACE DSP hardware. The daemon runs as a background process and controls the network connection to a DSP hardware client. All application programs must access the DSP hardware via a network connection to the daemon, which monitors and serializes all accesses to that client. The interface to the daemon is the *CLIB API*, which provides C language functions to be called by host application programs and allows to easily access the DSP hardware.

The DSP daemon allows simultaneous accesses by several host applications to a single DSP board. The use of sevelal DSP boards is also supported. Another feature of the daemon is the mirroring of some DSP board's registers into the host computer's memory. This concept saves many accesses to the board's registers and increases communication performance between the host computer and the DSP board. This feature enables a host program to alter system parameters on-the-fly, i.e. while the DSP is executing the DSP program.

A DSP name server (*rpc.dspnamed*) keeps track of active DSP daemons. The main task of the name server is to provide information about which *dsp_daemon* is serving which DSP client hardware. The name server is started automatically on startup of the host system. (For the ARO-DURIP facility's LAN, the name server is *rottweiler*).

3.1.1 Starting dsp_daemon

The daemon may be invoked by typing the command line

```
dsp_daemon -c client [-p port]
```

-c client specifies the name of the dSPACE hardware client to which the daemon will connect. (The names of the DSP clients for the ARO-DURIP facility's LAN are *goofy1, goofy2, goofy3,* and *goofy4*). Alternatively, the client may be specified by its network address.

-p port specifies the host's TCP port. This option need not to be specified, the default port will be used.

3.1.2 Terminating dsp_daemon

Because dsp_daemon runs as a background process it is not possible to terminate it directly. Though it is recommended to use the DSP daemon controller (to be described later in the section) to terminate a daemon, it may sometimes be necessary to use the deinstallation option *-u* with another call of the daemon:

```
dsp_daemon -c client [-p port] -u
```

If the daemon process id cannot be determined, the daemon deinstallation will fail. In this case, you can terminate the daemon manually using the UNIX *kill* command.

3.1.3 dcont - The DSP Daemon Controller

The program *dcont* is used to control the DSP daemon. The command line syntax is

```
dcont [options]
```

with the following options

-a register all DSP processor board specified in the initialization file *dspace.ini*.

-b board register DSP processor board with name *board*. An entry with that name must already exist in the initialization file *dspace.ini*.

-s server use the name server on host *server*. This option overrides the setting for a name server by the environment variable *DSP_NAME_SERVER*. (For the ARO-DURIP facility's LAN, the setting for the environment variable *DSP_NAME_SERVER* is *rottweiler* and this option need not be used).

-c client connect to the DSP network *client* its daemon. If no daemon is available for the specified client, a daemon process will be started automatically.

-d board delete entry for DSP processor board from file *dspace.ini*.

-k terminate the daemon for the specified client.

-l list all processor boards and host applications currently registered at the daemon.

-r reset the daemon for the specified client.

-h display short help about available options.

The file *dspace.ini*, located in the directory specified by the environment variable CITFILES, contains the information necessary to recognize and initialize the complete DSP hardware system. It is not recommended and it is not needed to edit the file by hand. If the file is corrupted, the DSP software will report errors and exit.

A daemon can be reset by entering *dcont -c client -r*. A reset is necessary if an application using the DSP daemon has terminated without freeing all resources it has allocated from the daemon.

To terminate a DSP daemon type the command line

```
dcont -c client -k
```

Please note, that the daemon will terminate even if there are still applications registered.

3.1.4 dsp_ping - A Network Utility

The *dsp_ping* program checks the reachability of a network client at the IP level of the network protocol. The calling syntax is:

```
dsp_ping client [timeout]
```

The default value for the *timeout* option is 20 seconds.

For more information about the DSP daemon envoronment refer to [3].

3.1.5 An Example

The application example program *trctst31.c* is distributed with the dSPACE software. It can be found into the demo directory */usr/Dspace/dsp_cit/demo1102/std*. The demo program performs the simulation of a 2nd-order spring-mass-damper system by using an Euler integration. The procedure to execute this application on the *goofy1* dSPACE controller is the following: (You must first copy the *trctst31.c* file into you working directory.

- dcont -l -c goofy1 -b ds1102
- down31 trctst31 -c goofy1

The DSP daemon controller *dcont* has to be invoked only once, thus if you want to download another application, first reset the DSP controller and use the *down31* program again.

- ld31 -c goofy1
  and select the Reset option.
- down31 filename -c goofy1

When you finish your work, you must reset the DSP controller properly and the DSP daemon must be terminated.

- ld31 -c goofy1
  and select the Reset and Quit DSP option.

- dcont -c goofy1 -k

Finally, if for some reason the necessary tools for compiling and downloading an application must be invoked separately, the sequence of the command is the following

- dcont -l -c goofy1 -b ds1102
  (start the DSP daemon)

- cl30 trctst31 -z -o trctst31.out -l rts30.lib -l ds1102.lib
  (compile and link)

- rm trctst31.obj
  (remove the file generated by the compiler)

- mv trctst31.out trctst31.obj
  (rename the file generated by the linker)

- ld31 trctst31 -c goofy1 -b ds1102
  (download the program)

3.2 The Texas Instruments TMS320 floating-point C compiler

The TMS320 floating-point C compiler operates in two passes: the first pass parses the code, and the second pass produces a single assembly language source file. To invoke the shell program to compile and assemble *filename.c*; enter:

```
cl30 filename
```

If you want to inspect the assembly language output, use the -k option to retain the assembly language file:

```
cl30 filename -k
```

If you use the -z option, the output will be an *executable* object module. The following examples show the two ways of creating an executable object module:

- Use the *lnk30* command and link the object file with the libriaries *rts30.lib* and *ds1102.lib*:

  ```
  lnk30 -c filename -o filename.out -l rts30.lib -l ds1102.lib
  ```

  This example uses the -c linker option because the code came from a C program. The -l option tells the linker that the input files *rts30.lib* and *ds1102.lib* are object libraries. The -o option names the output module *filename.out*; if you don't use this option, the linker names the output module *a.out*.

- In this example, use the -z option, which tells the shell program to run the linker. -z is followed by linker options.

  ```
  cl30 filename -z -o filename.out -l rts30.lib -l ds1102.lib
  ```

Refer to [4] for more information about the Texas Instruments C compiler.

dSPACE software environment provides the linker command file *ds1102.lnk* to be used for customization of the linker options. It is automatically used for linking if you use the script *down31*. It defines where to place the different sections created by the C compiler in the DS1102's memory. It may be used to instruct which object modules and libraries are to be linked. See [3] for the linker command file *ds1102.lnk*.

3.3 The Program Loader

This section describes the LD31 program necessary to load and execute application programs with the controller board DS1102. The program loader LD31 is invoked by typing

```
ld31 [obj_file] [options]
```

The command options (-b board_name, -c client, -l slave_appl) have the usual actions.

If a main DSP object module is specified in the command line, LD31 is invoked in the command mode. Otherwise it enters the menu mode and prompts for further actions. One of the following choices can be selected from the main menu.

- Load object module
- Restart DSP

- Reset DSP
- Select slave application
- Quit LD31 and disable DSP
- Quit LD31

The use of the "Quit LD31 and disable DSP" option is recommended before terminating the DSP daemon which establish the connection between the host computer and the client DSP processor.

Important: *obj_file* is the filename of the executable module created as it was described in the previous section. The object filename must be used without an extension, *.obj* extension is assumed. In the previous examples an extension *.out* was used for the executable module, while an *.obj* extension is used for the output of the compiler. Thus, after the creation of the executable module you must invoke the following commands:

```
rm filename.obj
mv filename.out filename.obj
```

The program loader is described in [5].

3.4 Real-time TRACE module

This section describes briefly the dSPACE TRACE module. TRACE provides real-time capabilitites for any application running on a DS1102 DSP board. It allows all signals and parameters represented as single precision float and integer variables in the processor board's memory to be recorded and displayed graphically. Thus it provides an extremely good insight into any application while data acquisition is executed in real-time on the dSPACE processor board.

TRACE needs several input files containing information on the DSP program currently running as well as some configuration files. Some of the files are global files existing only once in the environment while others contain application-specific information. TRACE also creates some output files to store various information.

TRACE needs at least two input files that are specific to each DSP application. These are the trace file (file extension is *.trc*) and the DSP programs linker map file (file extension is *.map*).

The trace file is an ASCII file and describes which variables of the DSP program are to be traced. All variables to be traced must be global variables in the C program. For a detailed description of the syntax of the *.trc* file refer to chapter 5 of [6].

After TRACE has determined the structure and the variable names from the trace file, it must obtain the addresses of the variables. These are retrieved from the *.map* file. The linker map file is an ASCII file generated by the linker together with the DSP program file.

If an error occurs while either the trace or the linker map file is being traced, an error file is generated (*.err*). This is a copy of the trace file with error messages.

The layout of one plot window is called a template, and can be saved in a template file (the file extension is *.tpl*). An experiment file is an other application-specific file. It offers the opportunity of saving the whole experiment environment of the current TRACE session to disk. TRACE of course allows traced data to be saved to disk and load them into memory later. It supports three file formats, the MATLAB binary format (file extension is *.mat*), the table data ASCII format (file extension is *.thi*), and an intermediate file format (file extension is *.idf*) which is used only when TRACE streams data to disk.

For more information about files used by the TRACE module refer to [6].

4 C Environment for the DS1102 Controller Board

This section is intended to give an overview of the basic DSP software, which is used to write C programs for the DS1102 controller board.

Initialization of the TMS320C31 DSP on the DS1102 is performed by the STARTUP code which must be linked to every DS1102 application program and it is included in the object library *ds1102.lib*. The STARTUP code will be linked automatically if the linker command file *ds1102.lnk* is used for linking or if the shell script file *down31* is used to compile and download an application program.

The directory *dsp_cit/c31* contains a set of functions for initialization of the TMS320C31's on chip timers and to access the DS1102 peripherals. All functions have been compiled with the highest optimization level and collected into a library named *ds1102.lib*. For a complete function reference look at [3].

The directory */usr/Dspace/dsp_cit/c31* also contains appropriate header files (*.h*) containing function prototypes, definitions, and macros to make the implementation of application programs more convenient. These header files can be included in application programs when functions from the library are used.

4.1 Application Demo Program

The application example program *trctst31.c* is distributed with the dSPACE software. It can be found into the demo directory */usr/Dspace/dsp_cit/demo1102/std*. The demo program performs the simulation of a 2nd-order spring-mass-damper system by using an Euler integration. The input and output statements are implemented as macros and can be easily substituted or customized in order to match your individual requirements. A square-wave signal generated by the application program itself is supplied as an input signal by default. Thus the program can be used without accessing any peripheral hardware.

```
/* TRCTST31.C ********************************************************

  DSP application demo program for the DS1102 processor board

  Simulation of a continuous 2nd-order spring-mass-damper system.
  The parameters are located at fixed addresses in memory
  for on-the-fly modification by the host.

  (C) 1993-1995 dSPACE GmbH

  $RCSfile: trctst31.c $ $Revision: 1.2 $ $Date: 1996/09/02 16:03:26 $
*********************************************************************/

#include <brtenv.h>                     /* basic real-time environment */


/*-------------------------------------------------------------------*/
/* I/O macros (may be modified for individual requirements) */

/* square-wave input signal (Tp = 200 * DT) */
#define input(u) {                      \
  static int i = 0;                     \
  if (++i >= 100)                       \
  {                                     \
    u *= -1;                            \
    i = 0;                              \
  } }

/* empty output macro */
#define output(y)

/* further I/O examples follow as comments */

/* input via DS1102 on-board ADC channel 1 */
```

```
/*
#define input(u)                        \
  ds1102_ad_start();                    \
  u = ds1102_ad(1)
*/

/* output via DS1102 on-board DAC channel 1 */
/*
#define output(y) ds1102_da(1, y)
*/


/*---------------------------------------------------------------------*/
#define DT 1.0e-4                       /* 100 us simulation step size */

#define NSTATES 2                                  /* number of states */
float x[NSTATES], x_dot[NSTATES];                    /* state matrices */

#define D_DEF 16.0                                  /* initial damping */
#define C_DEF 20000.0                    /* initial spring coefficient */
#define M_DEF 0.04                                     /* initial mass */

float *d = (float *) (DP_MEM_BASE);                         /* damping */
float *c = (float *) (DP_MEM_BASE + 1);          /* spring coefficient */
float *m = (float *) (DP_MEM_BASE + 2);                        /* mass */

float u = 0.5;                                          /* input value */

/* variables for execution time profiling */
#define TMR0 0                                    /* timer0 definition */
float exec_time;                                     /* execution time */
unsigned long count0;                             /* timer0 time count */

/* error flag for CHKERR at last dual-port memory location */
volatile int *error = (int *) (DP_MEM_BASE + DP_MEM_SIZE - 1);


/*---------------------------------------------------------------------*/
void derivatives()
{
  float fc, fd;

  fd       = *d * x[1];
  fc       = *c * (x[0] - u);
  x_dot[0] = x[1];
```

```
  x_dot[1] = (-fd - fc) / *m;
}


/*------------------------------------------------------------------*/
void integrate_with_euler()
{
  int i;

  derivatives();
  for (i = 0; i < NSTATES; i++)
    x[i] = x[i] + x_dot[i] * DT;
}


/*------------------------------------------------------------------*/
void isr_t0()                       /* timer0 interrupt service routine */
{
  begin_isr_t0(*error);                                /* overload check */
  service_trace();                                     /* call TRACE code */
  count0 = count_timer(TMR0);                /* save current timer count */

  input(u);                                            /* get input value */
  output(x[0]);                    /* output result (x[0] is output) */
  integrate_with_euler();                        /* update state matrices */

  exec_time = time_elapsed(TMR0, count0);  /* calculate execution time */
  end_isr_t0();                     /* end of interrupt service routine */
}


/*------------------------------------------------------------------*/
void main()
{
  int i;

  *d = D_DEF;                                  /* initialize parameters */
  *c = C_DEF;
  *m = M_DEF;
  for (i = 0; i < NSTATES; i++)                /* clear state variables */
    x[i] = 0.0;

  init();                                /* initialize hardware system */
  *error = NO_ERROR;                           /* initialize error flag */
  start_isr_t0(DT);                  /* initialize sampling clock timer */
```

```
  while (*error == NO_ERROR)                    /* background process */
    service_cockpit();                           /* call COCKPIT code */
}
```

An explanation of the function and macro calls can be found in [3].

4.2 CLIB - Host-DSP Interface Library

Many applications require data transfer between the host and the DSP, i.e. for on-line parameter modification, data acquisition, application program control. etc. To simplify control of the processor board and access to the DSP memory, a set of basic functions was created. The function references and detailed descriptions of all DSP processor board control and access functions and definitions are given in [7].

To call any of the DSP interface functions in an application program, the header file *clib.h* must be included into that program. The same applies to auxiliary functions contained in other modules (see [7] for details). The appropriate object library must be linked to the application program using the options of your C compiler (e.g. -lclib for the cc compiler).

For the application demo program described before the demo host application *pctrl.c* can be found in the directory */usr/Dspace/dsp_cit/demo1102/clib*. The host demo program will alter the spring, mass and damping parameters on-the-fly, i.e. while the DSP is executing the DSP program. The *trctst31* object file must be downloaded first. Then the demo host application *pctrl* can be started to modify the parameters of the simulation on the DSP.

References

[1] General Software Installation Guide for UNIX Workstations, dSPACE, Paderborn, Germany, 1994-95.

[2] DS1102 Software Environment Reference Guide, dSPACE, Paderborn, Germany, 1994-95.

[3] DSP Daemon Environment User's Guide, dSPACE, Paderborn, Germany, 1994-95.

[4] TMS320 Floating-Point DSP Code Generation Tools Getting Started Guide, Texas Instruments, 1996.

[5] DS1102 Program Loader User's Guide, dSPACE, Paderborn, Germany, 1994-95.

[6] Real-Time TRACE Module User's/Reference Guide, dSPACE, Paderborn, Germany, 1994-95.

[7] Host-DSP Interface Library CLIB User's Guide, dSPACE, Paderborn, Germany, 1994-95.