

```

c:\Temp\sample\E:\src\window1
#include "Dsign.h"
UNIT_Lint_struct std_unit = {1900,0,'0','1','X','\x11','\x13'};

void main ( void )
{
  lint_module();
  lint_unit (std_unit);
  send_string ("My Message");
}

void send_string (char *str)
{
  int k;
  char c;
  k = 0;
  while ( (c = str[k++]) != '\0' )
  {
    while ( ! send_char (c) ) ;
  }
}

```

## D.SignT Software Solutions

Hardware and software form an integral unit in any realtime application. This is especially true on a DSP system, where blocking functions or excessive data copy instructions will almost certainly lead to realtime violations, resulting in data loss and erroneous results.

Therefore we carefully fine-tune hardware and software of a D.Module to provide optimum performance and user-friendliness. Our goal is to provide a consistent and intuitively usable programming interface for the D.Module hardware, encapsulating hardware dependences and avoiding common programming pitfalls - the key note of the D.Module.BIOS.

Field maintenance and serviceability, as well as convenient upgrade paths, are additional musts for a competitive end product. D.Modules do meet these requirements already at the board-level: The built-in firmware supports program uploads, logic reprogramming, and configuration changes via a RS232 or USB connection - or even network-based if an Ethernet daughter card is attached.

Maintenance is further improved by the Module Config File, a text file, which allows to define installation-specific data like network settings or algorithm parameters. Should modification be required, simply change the Module Config File, no recompilation of the application program is required.

The D.FAT file system for SD and MMC cards finally provides a convenient method for mass data storage and offline data exchange between the DSP system and a PC.

### D.Module.BIOS

The D.Module.BIOS functions are an integral part of the module, permanently stored in the Flash Memory and booted at power-up or reset. To use the D.Module.BIOS in your program, only a header file must be included, containing prototypes, pre-defined parameters, and call addresses. The BIOS handles initialization and programming of all the module's peripheral devices. Because this programming interface is consistent throughout the D.Module family, upgrading (and downsizing) to different modules is much simplified. The table below shows an excerpt of the available functions.

The software engineer doesn't have to struggle with hardware configuration options, timing constraints, and figure out the profundities of each device - instead the BIOS let's him concentrate on his actual task: implementing a signal processing algorithm.

Since our boards are targeted at industrial applications we have to ensure many years of delivery, service, and replacement. Yet several components of a DSP system are reliant on consumer mass market conditions. This is especially true for memories and communication devices. Some parts may become obsolete in future and need to be replaced by similar, but slightly divergent, parts, requiring modified initialization or programming. Consequently this leads to software adaption, resulting in wasteful administration of multiple hardware and software versions.

The D.Module.BIOS copes with this situation: Hardware changes will not become visible to customers, their effects are handled in the BIOS without having to change the application program code.

Device	Functions	Example
Board	initialize and configure	DM2_busConfig (DM2_BUS_SYNC);

Device	Functions	Example
Interrupts	configure	DM2_intMap (DM2_INT_USBCPU, 4)
Boot Loader	boot load program from Flash Memory	DM2_bootload (0x30000);
UART	initialize, configure, data transfer	DM2_uartWriteStr (h_uart0, "hello world!", 0);
USB	initialize, configure, data transfer	cnt = DM2_usbReadBlock (dstbuf, ENDPOINT2);
Flash	erase, program, read, Intel-Hex upload	DM2_flashWriteBlock (0x30000, srcbuf, 1024);

 [back to top](#)

## D.Module.Setup Utility

Field maintenance and update options are frequently demanded for industrial systems, mostly required for installing bug fixes and adaptation to installation changes.

The D.Module firmware includes a Setup Utility which readily meets these requirements. Communicating via a RS232 serial link or USB (no need for specialized emulators or programming cables!) you are able to upload program and data files, change configuration files in the module's Flash Memory, reprogram the reconfigurable logic devices, and execute diagnostic functions. If your system is expanded with a D.Module Ethernet Daughter Card this functionality is available via network too.

Since the Flash Memory capacity of the D.Modules is sufficient to store multiple programs you are able to include diagnostic programs or calibration routines, which are not part of the default application, but can be invoked by a service technician on demand. Data and parameter files stored in ASCII or binary format help to simplify adaptations. Only the parameter set needs to be replaced, no need to modify the program code itself.

Function	Description
Intel-Hex Upload	upload executable programs to the Flash
ASCII, Binary and Configuration File Upload	upload data and parameter sets
Intel-Hex, ASCII and Configuration File Download	download parameter, log and report files
CPLD Program File Upload	reprogram the User-CPLD (e.g. to accommodate interface changes)
read and write memory	basic debugging
load and execute programs	execute programs other than the default application, e.g. diagnostic or calibration programs

 [back to top](#)

## D.Module.Configuration File

The Configuration File is an ASCII file stored in the module's Flash Memory. It is used to store system configuration and start-up options. User-defined sections can of course be added too. Below is a sample configuration file.

The most obvious advantage is to store installation-dependent data in the Configuration File. Any program using the UART or the Ethernet adapter can read it's settings from the Config file, resulting in only one central data base to maintain. End-customers can make changes on their own without access to the program code.

We strongly encourage you to add your own user-defined sections for installation-dependent parameter storage, e.g. calibration data, signal processing coefficients, etc. Should one of these parameters need adjustment during product lifetime, only the Module Configuration File needs to be changed. The

program code itself is not affected - invaluable especially after upgrading the highly optimizing Code Generation Tools, where new versions are prone to produce different code size and different optimization results, which may involve expensive software validation cycles.

### A Sample Module Configuration File

```
[DSP] ; DSP Configuration
  core clock = 720000000 ; DSP core clock in Hz (480, 600, or 720 MHz)
  bus clock  = 133000000 ; DSP EMIF clock in Hz (83, 100, 125, or 133 MHz)
  endian     = 0         ; DSP endianness (0:little, 1:big)
  mac        = 1         ; DSP Ethernet MAC (0:disabled, 1:enabled)

[UART0] ; UART 0 communication parameters
  baud rate  = 115200
  data bits  = 8         ; 5, 6, 7, or 8
  stop bits  = 1         ; 1 or 2
  parity     = N         ; N:none, O:odd, E:even, 1:forced-1, or 0:forced 0
  flow control = X       ; N:none, X:Xon/Xoff, or R:RTS/CTS
  xon code   = 0x11     ; hex and dec notation allowed, typically 0x11
  xoff code  = 0x13     ; hex and dec notation allowed, typically 0x13

[UART1] ; UART 1 communication parameters
  baud rate  = 9600
  data bits  = 8         ; 5, 6, 7, or 8
  stop bits  = 1         ; 1 or 2
  parity     = N         ; N:none, O:odd, E:even, 1:forced-1, or 0:forced 0
  flow control = X       ; N:none, X:Xon/Xoff, or R:RTS/CTS
  xon code   = 0x11     ; hex and dec notation allowed, typically 0x11
  xoff code  = 0x13     ; hex and dec notation allowed, typically 0x13

[APPLICATION] ; Application program
; boot address = 0x30000 ; hex and dec notation allowed
  boot address = 0x20000 ; hex and dec notation allowed

[ETH] ; Network Settings
  mode = A                ; A=autonegotiation
  hostname = dsp.company.net
  IP = 192.168.1.199
  DNS = 192.168.1.1
  gateway = 192.168.1.1

[END]
```

[↑ back to top](#)

## D.Module.TCP/IP Stack

---

TCP/IP communications on a DSP system greatly differs from the well-known Personal Computer requirements. A DSP does not need to handle hundreds of simultaneous connections as does a PC based server. On the other hand, latencies and resource consumption are much more critical to avoid compromising signal processing performance.

The majority of available protocol stacks rest upon an underlying operating system which provides buffer space and schedules events. Many DSP application's realtime constraints however are too demanding to tolerate the overhead of an operating system. A common solution is to "outsource" TCP/IP to a companion processor. However, protocol overhead for interprocessor communications is not negligible and about the same order of magnitude as the UDP protocol. Data transfer time to/from a companion processor is comparable to a direct transfer to/from a network controller. The "pro" side of this approach is reducing the DSP's burden of handling the TCP protocol. On the "con" side, the higher level protocols based on TCP (HTTP, FTP) require complex interactions of DSP and companion processor, which will use

up the advantages.

We therefore decided to develop our own TCP/IP stack for the D.Module and DSK Ethernet Adapter boards. It has been carefully tailored to the requirements of DSP systems. Code and data memory size are minimized, and no additional resources like DSP interrupts or timers are required. The D.SignT.TCP/IP protocol stack can be used in a linear C program, just as running as a task in a realtime operating system, e.g. Texas Instruments DSP/BIOS. The stack supports the following protocols:

Protocol	Usage
ARP	Address Resolution Protocol, resolves the IP address to a hardware MAC address. No user action is required. If an address is unknown, an ARP request is generated automatically.
IP	Internet Protocol. All data transferred by DNS, DHCP, ICMP, UDP and TCP is automatically packed into IP packets.
ICMP	Internet Control Message Protocol. The D.SignT.TCP/IP protocol stack responds to "ping" requests to test a connection
UDP	User Datagram Protocol, provides a one-to-one or one-to-many connectionless data path. Data transmitted via UDP is not guaranteed to reach its destination. This protocol has very low overhead and is especially useful for transmitting non-critical data like audio and video streams, or any other realtime data where retransmissions cannot be tolerated.
TCP	Transmission Control Protocol, provides reliable, connection-oriented, one-to-one connections. All data is acknowledged by the receiver and retransmitted automatically if required. This protocol should be used for critical data like software uploads, commands, etc.
DHCP	Dynamic Host Configuration Protocol. This protocol has been developed to ease maintenance of a TCP/IP network. A DHCP server manages the allocation of IP addresses and provides additional network configuration data like gateways, DNS servers etc. The TCP/IP stack integrates the client functions required to obtain an IP address, DNS server, and gateway.
DNS	Domain Name System. This protocol allows to use symbolic host names instead of numerical IP addresses. The TCP/IP stack integrates the client functions to query a DNS server to resolve a host name.
FTP	File Transfer Protocol, based on TCP. A FTP server library exists, which allows to upload programs and parameters to the module's Flash Memory, or download log files and data from the DSP. The FTP server is widely configurable: users, passwords, directories, files, and access restrictions are maintained in a simple data structure.
HTTP	Hypertext Transfer Protocol, based on TCP. We do provide a HTTP server framework, which passes GET and POST parameters to a user-defined callback function, hence providing the required flexibility for dynamic data. The DSP can send static HTML pages and images as well as inserting the current value of variables, generate images from data acquisition buffers, etc. on demand.

The following table lists basic TCP/IP stack functions:

Function	Description
net_init	initialize the TCP/IP stack
net_set_gateway	configure the default gateway for connections outside the local network
socket_open	create and open a new socket
socket_close	close a socket
set_socket_option	specify non-default socket options, e.g. disable UDP checksum
install_icmp_socket	install a socket for ICMP messages (ping)
net_send	non-blocking send function
net_send_ready	blocking send function (binary data)
net_send_string	blocking send function (ASCII streams)

Function	Description
net_recv	non-blocking receive function
net_recv_ready	blocking receive function
net_isq	main networking function, must be called periodically in your program main loop, from a timer interrupt, or as a task in a RTOS
connect	establish a TCP connection
shutdown	shutdown a TCP connection
gethostbyname	host name resolution
accept	test if a TCP socket is connected

The D.SignT TCP/IP stack is available for the following D.Modules, DSKs and EVMs:

DSP-System	Ethernet Daughtercard	Software Package Order Code
D.Module2.C6747	not required	DS.TCP/IP-C6747
D.Module2.DM642	not required	DS.TCP/IP-DM642
D.Module.C6713	D.Module.91C111	DS.91C111
D.Module.C6203	D.Module.91C111	DS.91C111
D.Module.C6201	D.Module.91C111	DS.91C111
D.Module.C6701	D.Module.91C111	DS.91C111
D.Module.VC33	D.Module.91C111	DS.91C111
D.Module.21065	D.Module.91C111	DS.91C111
TMS320VC5502 EVM	DSK.91C111	DS.DSK91C111
TMS320VC5509 EVM	DSK.91C111	DS.DSK91C111
TMS320VC5509 DSK	DSK.91C111	DS.DSK91C111
TMS320VC5510 DSK	DSK.91C111	DS.DSK91C111
TMS320C6413 EVM	DSK.91C111	DS.DSK91C111
TMS320C6416 DSK	DSK.91C111	DS.DSK91C111
TMS320C6455 DSK	not required	DS.TCP/IP-DSK6455
TMS320DM642 EVM	not required	DS.TCP/IP-EVMDM642
TMS320C6711 DSK	DSK.91C111	DS.DSK91C111
TMS320C6713 DSK	DSK.91C111	DS.DSK91C111
TMS320C6747 / OMAP-L137 EVM	not required	DS.TCP/IP-EVM6747
eZdsp TMS320F2812, R2812	eZdsk.91C111	DS.eZdsk91C111
eZdsp TMS320F28335	eZdsk.91C111	DS.eZdsk91C111

Please [contact](#) us or our international resellers for price, delivery, and volume discount information

## D.FAT File System for SD and MMC Cards

D.FAT is a file system for SD and MMC cards and provides standard compliant data exchange between the DSP system and a Personal Computer. D.FAT currently runs on C6713 and C6203 D.Modules with attached D.Module.91C111.

D.FAT installs low-level drivers for the Texas Instruments stdio runtime library and allows to access files on the memory card using high-level C functions like fprintf() and fscanf().

Function	Description
FAT System	supports FAT12 and FAT16 cluster size 512 bytes 8.3 directory and file names
FAT commands	getcwd, chdir, mkdir, rmdir, dir, ren, del
Stdio Support	fopen, fclose fread, fgetc, fgets, fscanf fwrite, fputc, fputs, fprintf fsize, ftell, fgetpos, fseek
File Open Modes	r, w, a, r+, w+, a+, b

Please [contact](#) us or our international resellers for price, delivery, and volume discount information

 [back to top](#)

© D.SignT 1998-2009 comments: [webmaster@dsignt.de](mailto:webmaster@dsignt.de) [Impressum / Imprint](#) last change: 2009-09-17



Kane Computing Ltd  
7 Theatre Court, London Road,  
Northwich, Cheshire, CW9 5HB, UK.  
Tel: +44(0)1606 351006  
Fax: +44(0)1606 351007/8  
Email: [sales@kanecomputing.com](mailto:sales@kanecomputing.com)  
Web: [www.kanecomputing.co.uk](http://www.kanecomputing.co.uk)