



# Configuring an Optimal TCP/IP Stack

## Introduction

The ZiLOG TCP/IP Software Suite (ZTP) supports a large number of Internet protocols. Many applications do not require all of these protocols. This Technical Article explains how to generate an appropriate ZTP configuration for ZiLOG's eZ80<sup>®</sup> family of microprocessors by means of selecting applicable ZTP libraries.

The results show that each configuration fits within the eZ80F91 MCU's internal Flash memory. The document is structured into the following sections:

- Introduction
- ZTP Protocols and Libraries
- Code and Data Sizes
- Application Protocols

## ZTP Protocols and Libraries

The ZiLOG TCP/IP Software Suite includes multiple libraries that can be linked to a project. These libraries correspond to protocols in the TCP/IP stack. When creating a custom project, the user is only required to include the libraries specific to a particular application. However, some of the TCP/IP protocol libraries are dependent upon other protocol layers. If a dependent library is not linked to a project, it is necessary in some cases to include a stub file for the project to build properly.

Other libraries are completely independent of lower-level protocol libraries and can either be included in or excluded from a project without requiring the addition of a stub file<sup>1</sup>. The stub files are located in the `..\ztp\conf` directory and can be used in place of the corresponding full protocol library. These stub files and their corresponding library files are listed in Table 1.

**Table 1. Stub and Library File Association**

Stub Files	Corresponding Library Files
icmp_stub.o	ICMP.lib
igmp_stub.o	IGMP.lib
net_stub.o	NET.lib
tcp_stub.o	TCP.lib
tty_stub.o	TTY.lib
udp_stub.o	UDP.lib

Note: \*For PPP only.

---

<sup>1</sup>Many drivers can be omitted from ZTP by including a driver stub instead of the driver library itself.

**Table 1. Stub and Library File Association**

Stub Files	Corresponding Library Files
arp_stub.o*	ARP.lib
pppsnmp_stub.o*	SNMP.lib

Note: \*For PPP only.

Some higher-level application layers depend on the TCP or UDP layer; this topic is discussed in the ZiLOG TCP/IP Software Suite Programmer’s Guide (RM0008). Whenever stub files are used to configure the ZTP stack without TCP or UDP, any of the associated TCP or UDP application libraries must be excluded.

Table 2 shows the minimum linker configuration (libraries and stubs) required for a set of basic projects—OS only, OS with shell, UDP only, TCP only, or both TCP and UDP. Each of these basic projects can be configured to operate over PPP or Ethernet (or both). When using Ethernet, it is also necessary to include a library specific to the Ethernet NIC used in the hardware platform.

Table 2 also lists the libraries and object stubs that are included when developing a project using an eZ80<sup>®</sup> Development Platform equipped with an eZ80F91 Module. For each project, the indicated libraries and object stubs (shown in shaded cells) must be included<sup>2</sup>.

**Table 2. Available Libraries and their Project Configurations**

Available Libraries	ZTP Configuration							
	OS Only	OS with Shell	TCP only w/ Ethernet	TCP only w/ PPP	TCP/UDP w/ Ethernet	TCP/UDP w/ PPP	UDP only w/ Ethernet	UDP only w/ PPP
ARP.lib			ARP.lib	arp_stub.o	ARP.lib	arp_stub.o	ARP.lib	arp_stub.o
debug.lib								
DGRAM.lib			DGRAM.lib	DGRAM.lib	DGRAM.lib	DGRAM.lib	DGRAM.lib	DGRAM.lib
ez80190.lib	ez80F91.lib	ez80F91.lib	ez80F91.lib	ez80F91.lib	ez80F91.lib	ez80F91.lib	ez80F91.lib	ez80F91.lib
ez80L92.lib								
ez80F91.lib								
ez80F92.lib								
ez80F93.lib								
HTTP.lib								
ICMP.lib			icmp_stub.o	icmp_stub.o	icmp_stub.o	icmp_stub.o	icmp_stub.o	icmp_stub.o
IGMP.lib			igmp_stub.o	igmp_stub.o	igmp_stub.o	igmp_stub.o	igmp_stub.o	igmp_stub.o
IP.lib			IP.lib	IP.lib	IP.lib	IP.lib	IP.lib	IP.lib
NET.lib	net_stub.o	net_stub.o	NET.lib	NET.lib	NET.lib	NET.lib	NET.lib	NET.lib

Note: \*This library resides in the eZ80<sup>®</sup> Compiler Library directory. The crtd.lib file is the debug version. The fdummy.lib file (1 byte) is also included by the eZ80<sup>®</sup> compiler when the floating point library is not checked.

<sup>2</sup>Not applicable to the application-level protocols (such as HTTP, TELNET, and SNMP).

**Table 2. Available Libraries and their Project Configurations (Continued)**

Available Libraries	ZTP Configuration							
	OS Only	OS with Shell	TCP only w/ Ethernet	TCP only w/ PPP	TCP/UDP w/ Ethernet	TCP/UDP w/ PPP	UDP only w/ Ethernet	UDP only w/ PPP
netapp.lib								
shell.lib		shell.lib						
SNMP.lib				pppsnmp_stub.o		pppsnmp_stub.o		pppsnmp_stub.o
SYS.lib	SYS.lib	SYS.lib	SYS.lib	SYS.lib	SYS.lib	SYS.lib	SYS.lib	SYS.lib
TCP.lib	tcp_stub.o	tcp_stub.o	TCP.lib	TCP.lib	TCP.lib	TCP.lib	tcp_stub.o	tcp_stub.o
TCPD.lib			TCPD.lib	TCPD.lib	TCPD.lib	TCPD.lib		
TTY.lib	tty_stub.o	TTY.lib	tty_stub.o	tty_stub.o	tty_stub.o	tty_stub.o	tty_stub.o	tty_stub.o
UDP.lib	udp_stub.o	udp_stub.o	udp_stub.o	udp_stub.o	UDP.lib	UDP.lib	UDP.lib	UDP.lib
XC.lib	XC.lib	XC.lib	XC.lib	XC.lib	XC.lib	XC.lib	XC.lib	XC.lib
crt.lib*	crt.lib	crt.lib	crt.lib	crt.lib	crt.lib	crt.lib	crt.lib	crt.lib
eZ80_Website.lib								
Acclaim_Website.lib								
CS8900A.lib			F91_EMAC.lib		F91_EMAC.lib		F91_EMAC.lib	
RT8019AS.lib								
F91_EMAC.lib								
emac.lib								
PPP.lib				PPP.lib		PPP.lib		PPP.lib

Note: \*This library resides in the eZ80<sup>®</sup> Compiler Library directory. The crt.lib file is the debug version. The fdumy.lib file (1 byte) is also included by the eZ80<sup>®</sup> compiler when the floating point library is not checked.

ZTP includes libraries to support the EMAC devices listed in Table 3. Custom Ethernet drivers can be created for NIC that use the [Device Driver Kit](#) (DDK), available with the `emac.lib` library.

► **Note:** ZiLOG no longer supports the RT8019AS driver, and currently does not ship eZ80<sup>®</sup> Development Platforms using the RT8019AS controller. It is provided strictly for reference.

**Table 3. Network Interface Cards (NICs) and their Libraries**

NIC	Library
eZ80F91 w/ on-chip EMAC	FP1_EMAC.lib
Cirrus Logic CS8900A Crystal EMAC	CS8900A.lib
RealTek RTL8019AS EMAC	RT8019AS.lib
Other EMAC using DDK	emac.lib

The ZiLOG TCP/IP Software Suite also includes libraries that support the eZ80<sup>®</sup> devices listed in Table 4. It is necessary to include the library specific to the MCU that is used.

**Table 4. ZTP-Supported eZ80<sup>®</sup> MCUs**

ZiLOG Microcontroller	Library
eZ80190	eZ80190.lib
eZ80L92	eZ80L92.lib
eZ80F91	eZ80F91.lib
eZ80F92	eZ80F92.lib
eZ80F93	eZ80F93.lib

Additionally, the sample website library can be used for each of the two eZ80<sup>®</sup> families, as shown in Table 5.

**Table 5. eZ80<sup>®</sup> Website Libraries**

eZ80 <sup>®</sup> Family	Website Library
eZ80 <sup>®</sup> MPU	eZ80_Website.lib
eZ80Acclaim! <sup>™</sup> MCU	Acclaim_Website.lib

## EMAC Library Files

The F91\_EMAC.lib and ez80cs.lib library files support the eZ80F91 EMAC and the Crystal EMAC. One of these libraries must be added to an Ethernet-based project. Neither of these libraries is required to be included in a PPP-only project. The added library must correspond to the proper EMAC on the eZ80<sup>®</sup> Development Platform. The library can be added via either the **Settings Options** dialog in ZDSII or to the source files by using the **Add to Project** command from the ZDSII **Project** menu.

Follow the brief sequence below to add a library.

1. Choose **Settings...** from the **Project** menu of ZDSII. The **Settings Options** dialog box appears.
2. Click the **Linker** tab and select the **General** category.
3. Enter the appropriate changes in the **Object/Library modules:** field or click the ... button to select an object/library file from an appropriate path.

## Source Files For Any Project

At a minimum, all ZTP projects should include the source files boot.asm, EZ80\_HW\_Config.c, ipw\_ez80.c, and main.c. Samples of these files can be found in any of the ZTP project directories (for example, ..\ztp\demo or ..\ztp\PPPDemo). When using an eZ80<sup>®</sup> Development Platform under normal circumstances, the EZ80\_HW\_Config.c file does not require modification<sup>3</sup>.

<sup>3</sup>EZ80\_HW\_Config.c may require modification for a custom hardware platform that utilizes an eZ80<sup>®</sup> microprocessor.

The contents of `main.c` are dependent on the TCP/IP protocol layers required in the user's application. At minimum, `main.c` must include the following:

```
void
main
(
    void
)
{
}
```

The **main** routine above performs no function. When code is added to `main.c` to interface with TCP or UDP, or to possibly start an optional protocol layer in ZTP, it is necessary to include header files from the `..\ztp\includes` directory. The required list of header files is specific to the user's application. For the projects defined in Table 2, the following set of header files is a good starting point.

```
#include <kernel.h>
#include <bootinfo.h>
```

### Source Files For PPP Projects

For projects utilizing the Point-to-Point Protocol (PPP), the user must explicitly initialize PPP by calling the **ppp\_init** function. The **ppp\_init** function should be called just after the Ethernet driver is initialized (if Ethernet is required in the user's application). The `main.c` file in the `..\ztp\PPPDemo` directory provides an example. This `main.c` file includes a **netconfig** function near the top of the file. The **netconfig** function, shown below, can be found in the `main.c` file in the `..\ztp\pppdemo` directory.

```
/*
 *-----
 * netconfig - set network configuration parms using BOOTP
 *-----
 */
void netconfig()
{
    init_ether(&Bootrecord, b_use_dhcp);
    ppp_init( SERIAL1, &ppp );
}
```

► **Note:** If the project uses both Ethernet and PPP, then the **netconfig** function shown above is appropriate. If the project does not require Ethernet, then the call to **init\_ether** can be omitted.

### Code and Data Sizes

For each configuration in Table 2, a ZDSII project was created and compiled using ZDSII-eZ80Acclaim! version 4.6.0. Application code for the TCP and UDP projects include Open and Control functions to set up a connection, Read and Write functions to echo a packet to the client, and the Close function to terminate the connection. The C Compiler was set to perform size optimization.

Tables 6 and 7 summarize information from the map file generated as a result of building these projects. Table 6 shows the number of bytes of code generated for the different segments within the projects. All

segments within Table 6 can be located in ROM or Flash because their contents do not require alteration during run-time. After each project was built, it was downloaded to internal eZ80F91 Flash memory and tested.. Table 7 shows the static RAM requirements (in bytes) for the various sections of each project. The sections shown in Table 7 cannot be placed in ROM or Flash because their contents change during execution of the project. In all cases, the numbers in the table are Base 10 (decimal) values.

► **Note:** The contents of Tables 6 and 7 do not account for the dynamic memory requirements of the ZTP stack.


**Table 6. ROM Code and Data Sizes for each Project Configuration**

Code Segment	OS Only	OS with Shell	TCP only w/ Ethernet	TCP only w/ PPP	TCP/UDP w/ Ethernet	TCP/UDP w/ PPP	UDP only w/ Ethernet	UDP only w/ PPP
.bootstrap	110	110	110	110	110	110	110	110
.startup	376	376	376	376	376	376	376	376
CODE	23353	39215	91988	99626	96176	103814	69574	77136
DATA COPY	5327	6144	5979	6696	6035	6752	5928	6645
STRSECT	1477	2603	3184	6070	3411	6302	3110	5974
TEXT	61	61	67	64	67	64	64	61
Total ROM	30704	48509	101704	112942	106175	117418	79162	90302

**Table 7. RAM Data Sizes for each Project Configuration**

Data Segment	OS Only	OS with Shell	TCP only w/ Ethernet	TCP only w/ PPP	TCP/UDP w/ Ethernet	TCP/UDP w/ PPP	UDP only w/ Ethernet	UDP only w/ PPP
.IRQVect	256	256	256	256	256	256	256	256
DATA	5327	6144	5979	6696	6035	6752	5928	6645
BSS	6231	8502	12228	11751	16626	16149	15322	14845
Total RAM	11814	14902	18463	18703	22917	23157	21506	21746

Tables 8 and 9 provide summaries of the memory requirements (in bytes) for each of the ZTP libraries that can be included in a project. The tables are separated to denote those libraries that are mandatory in any given project, and those libraries that are optional. In addition, Table 8 shows an approximation of the total amount of dynamic RAM the various protocol layers can require.

 **Caution:** Please be aware that the figures represented in Table 8 are dependent on many factors, such as the functions called by the application, the number of threads running in the OS,

the amount of data currently being transferred, etc., and may not match the memory requirements of a custom application. It is possible that different versions of ZTP and ZDSII will provide different numbers. Also, the linker in ZDSII only extracts the code from the library that supports the function called. These figures should not be interpreted as absolute maximum values; they should only be used as guidelines for considering which protocol layers to include within the custom application.

**Table 8. Mandatory Code and Data Sizes Along Protocol Boundaries**

Compile Time Memory Requirements					Run Time Heap Allocations <sup>1</sup>					
Library	Protocol	Fixed Code (CODE, TEXT)	Uninitialized Data (BSS)	Initialized Data (DATA, STRSECT)	# of Threads	Stack Size	BPOOL	Misc. <sup>2</sup>	ROM	RAM
crt	C Runtime	3662							3662	
XC	C Runtime	5120	3	237					5357	240
SYS	OS	15739	2837	509	1	2048			16248	6384
eZ80F91	Hardware Config., Serial	10584	47	6070					16654	6117
ARP	ARP	3243	156	376					3619	532
DGRAM	TCP/UDP	1907	222	162					2069	384
NET	BOOTP, DHCP, DNS, HDLC, RARP, SMTP, TFTP	29477	1150	4561	7	14336	81536		34038	101583
IP	IP	10137	155	207			725		10344	1087
<b>Total</b>		<b>79869</b>	<b>5560</b>	<b>12122</b>	<b>8</b>	<b>16384</b>	<b>82261</b>	<b>0</b>	<b>91991</b>	<b>116327</b>

Notes:

1. These values correspond to an inactive stack. When the stack starts processing data, the memory requirements increase.
2. The MISC heap memory allotment represents the 4.5% of dynamic memory usage spread throughout the stack that cannot be attributed to either stack or BPOOL allocations.

Table 9. Optional Code and Data Sizes Along Protocol Boundaries

Compile Time Memory Requirements					Run Time Heap Allocations					
Library	Protocol	Fixed Code (CODE, TEXT)	Uninitialized Data (BSS)	Initialized Data (DATA, STRSECT)	# of Threads	Stack Size	BPOOL	Misc.	ROM	RAM
F91_EMAC	EMAC	6297	180	477					6774	657
PPP	LCP, PAP, IPCP	16641	55	3541	1	2048			20182	5644
ICMP	ICMP	3899	104	42					3941	146
IGMP	IGMP	3167	163	31	1	2048			3198	2242
UDP	UDP	1948	160	54					2002	214
TCP		20345	77	479					20824	556
TCPD	TCP	6100	1283	105					6025	1388
HTTP	HTTP	7281		755	1	2500			8036	3255
SNMP	SNMP	26130	307	15169					41299	15476
netapp	SNMP, TELNET, TIME	1237		289	2	4096			1526	4385
shell	Command Interpreter	23292	264	6910	1	3072			30202	10246
TTY	Terminal Emulation	3809	2265	102					3911	2367
Total		120146	4858	27954	6	13764			148100	46576

Notes:

1. These values correspond to an inactive stack. When the stack starts processing data, the memory requirements increase.
2. The MISC heap memory allotment represents the 4.5% of dynamic memory usage spread throughout the stack that cannot be attributed to either stack or BPOOL allocations.

## Application Protocols

Adding code to `main.c` may require the inclusion of optional protocol layers and/or features of the ZTP stack. For example, to create a TCP-level application that transmits temperature control information to a monitoring station, the user might decide that it is helpful to also create an embedded website that facilitates remote viewing of the monitoring station data via a web browser. A project providing this feature would necessitate adding the optional HTTP service (or daemon). This section shows the optional components of ZTP that can be added to a project and the steps that must be taken to enable the feature. In most cases, the application must call a specific API to enable or use the feature.



If a function such as the above example is implemented in a library already included in the project, there is no more effort required to use the function. Where appropriate, the library containing these optional components has been identified to ensure that the user links all of the appropriate libraries.

### Launch A TELNET Daemon

A TELNET daemon can be launched using the following sequence.

1. Call `telnet_init()`
2. Link in `netapp.lib`

### Launch A Time Client

1. Call `timed_738_init()`
2. Link in `netapp.lib`

### Start A Shell On A Serial Port

A shell can be started on a serial port using the following sequence:

1. Include `shell.h`
2. Link in `shell.lib` and `tty.lib`
3. Open a TTY device on the appropriate serial port (typically `SERIAL0`). Ensure that the corresponding variable `b_xinu_uses_uart0` or `b_xinu_uses_uart1` is set to `TRUE` in the file `ipw_ez80.c`.
4. Call `shell_init()` and pass the device ID of the TTY driver to be used as a parameter.

The following code fragment provides an example of how to start a shell.

```
if ((fd=open(TTY, (char *)SERIAL0,0)) == SYSERR) {
    kprintf("Can't open tty for SERIAL0\n");
    return SYSERR;
}
kprintf("Starting up a shell on device %d\n", fd);
shell_init(fd);
```

### Add Network-Related Shell Commands

The user can add custom commands to the shell from the previous section by calling:

```
shell_add_commands(netcmds, nnetcmds);
```

For example, the following code fragment shows how to add a **mail** and a **tftp** command to the default shell.

```
char * mail_name = "mail"; // The string of characters you will type on the
                           console to execute the command
char * tftpdemo_name = "tftpdemo";
```

```

struct cmdent *mycmds;

/*
 * In this example memory is dynamically allocated at run time to store
 * the command structure.
 */
mycmds = getmem(sizeof(struct cmdent) * 1);
mycmds[0].cmdnam = mail_name;
mycmds[0].cbuiltin = TRUE;
mycmds[0].cproc = x_mail;
mycmds[0].cnext=NULL;

mycmds[1].cmdnam = tftpdemo_name;
mycmds[1].cbuiltin = TRUE;
mycmds[1].cproc = x_tftpdemo;
mycmds[1].cnext=NULL;

shell_add_commands(mycmds, 2);
    
```

## Initialize HTTP

To initialize a webserver, call `http_init()`. A few parameters must be passed into the call, but the default values shown below should suffice for a typical webserver.

1. `http_init(http_defmethods,httpdefheaders,website,80);`
2. Include `http.h`
3. Link in to `HTTP.lib`

## Initialize An SNMP Daemon

Network management protocols can be utilized by initializing an SNMP daemon. Follow the sequence below.

1. Include `snmp.h`
2. Call `snmp_init();`
3. Link in `SNMP.lib`

## Initialize A DHCP Client

Use of the DHCP client is controlled by a parameter on the call to **`init_ether`**. The first parameter to the **`init_ether`** call is the boot record (see the next section). The second parameter is a boolean expression. If this expression is `TRUE`, the DHCP client tries to obtain an IP configuration dynamically. If this expression is `FALSE`, the ZTP stack uses the static IP parameters specified in the boot record.

To enable DHCP, add the following code to `main.c`.

```

void
netconfig
(
    void
    
```

```
}  
{  
  init_ether(&Bootrecord, TRUE);  
}
```

A simpler way to enable or disable DHCP is to set the `b_use_dhcp` variable in the `ipw_ez80.c` file to either `TRUE` (enable) or `FALSE` (disable).

## Configure A Network

To modify the structure of the boot record, change the structure accordingly in the `main.c` file. The record is included at the top of `main.c`. See the `main.c` files in any sample project.

```
struct BootInfo Bootrecord = {  
  "192.168.1.1",      /* Default IP address */  
  "192.168.1.4",      /* Default Gateway */  
  "192.168.1.5",      /* Default Timer Server */  
  "192.168.1.6",      /* Default File Server - Not currently Used */  
  "",  
  "192.168.1.7",      /* Default Name Server */  
  "",  
  0xffffffff00UL     /* Default Subnet Mask */  
};
```

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

**ZiLOG Worldwide Headquarters**

532 Race Street  
San Jose, CA 95126  
Telephone: 408.558.8500  
Fax: 408.558.8300  
[www.ZiLOG.com](http://www.ZiLOG.com)

**Document Disclaimer**

ZiLOG is a registered trademark of ZiLOG Inc. in the United States and in other countries. All other products and/or service names mentioned herein may be trademarks of the companies with which they are associated.

ZiLOG is a registered trademark of ZiLOG Inc. in the United States and in other countries. All other products and/or service names mentioned herein may be trademarks of the companies with which they are associated.

©2004 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.