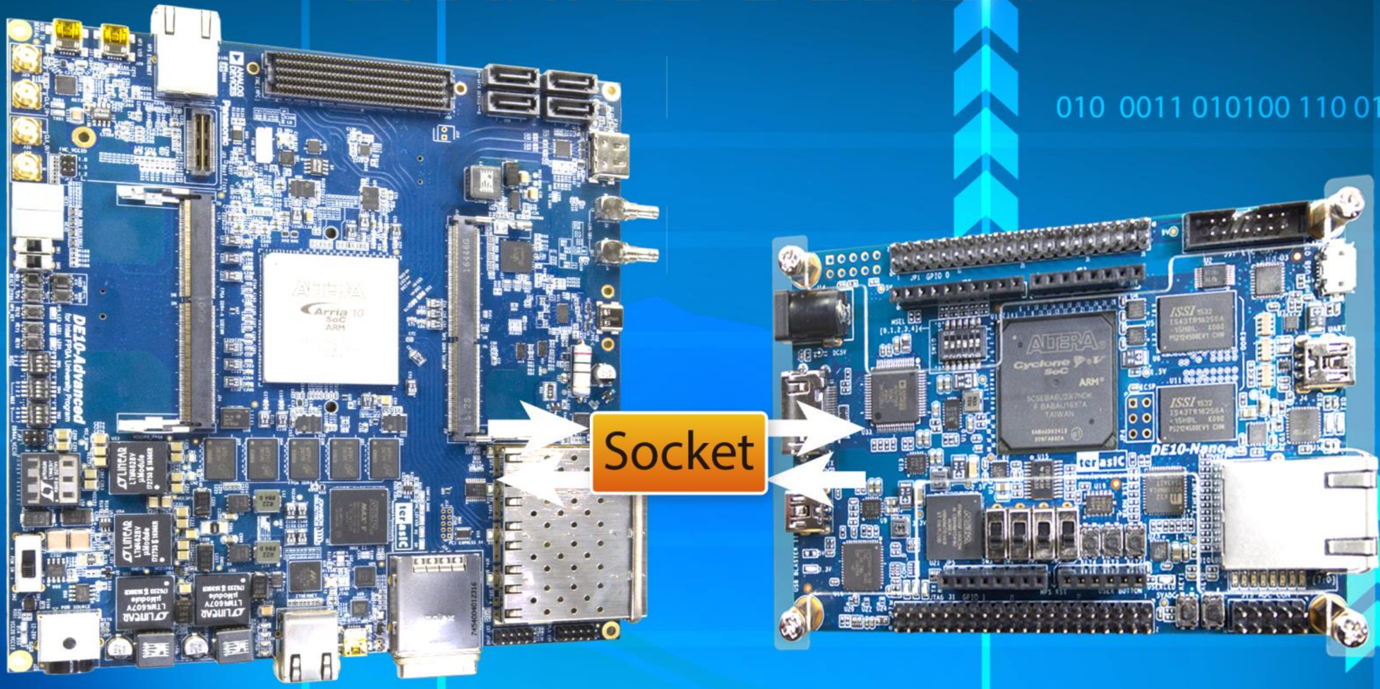
Network Socket
Example Design



VPX-A5SoC

DE0-Nano-SoC

DE10-Standard

DE10-Advanced

DE1-SoC

DE10-Nano

Contents

This document will describe how two remote application processes communication via socket in client-server model. Based on these design examples, developers can make their Linux Application Software, run on SoC FPGA boards and easily communicate with other hosts via a network socket. The example kit includes two folders: Manual and Demonstration. The Manual folder includes the user manual. The Demonstration folder includes the source code and binary execution code of both client and server project.

1. Sockets

Sockets are the fundamental technology for programming software to communicate on the transport layer of networks shown in **Figure 1**. A socket provides a bidirectional communication endpoint for sending and receiving data with another socket. Socket connections normally run between two different computers on a LAN, or across the Internet, but they can also be used for interprocess communication on a single computer.

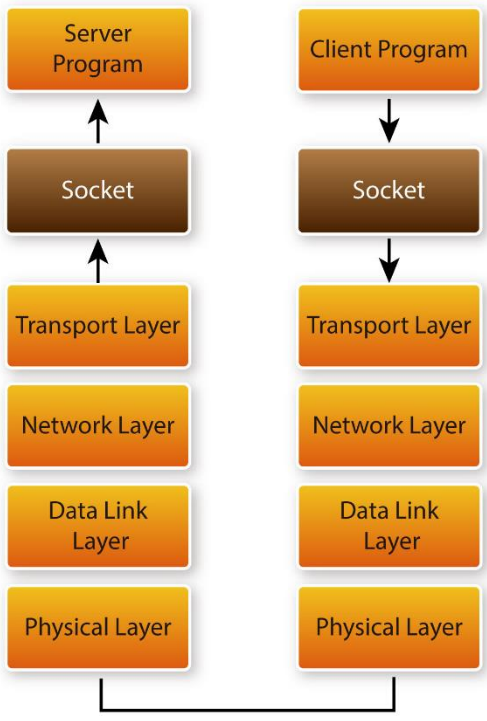


Figure 1 The two processes communicate on a network via a socket.

2. Client Server Model

Most interprocess's communication uses the client server model. These terms refer to the two processes which will be communicating with each other. One of the two processes, the client, connects to the other process, the server typically to makes a request for information. A good analogy is a person who makes a phone call to another person.

Notice that the client needs to know of the existence of and the address of the server, but the server does not need to know the address of (or even the existence of) the client prior to the connection being established.

Notice also that once a connection is established, both sides can send and receive information.

The system calls for establishing a connection which is somewhat different for the client and the server, but both involve the basic construct of a socket. A socket is one end of an interprocess communication channel. The two processes each establish their own socket. **Figure 2** shows the communication diagram between the client and server.

The steps involved in establishing a socket on the client side are as follows:

1. Create a socket with the socket() system call
2. Connect the socket to the address of the server using the connect() system call
3. Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

The steps involved in establishing a socket on the *server* side are as follows:

1. Create a socket with the socket() system call
2. Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
3. Listen for connections with the listen() system call
4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.

5. Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

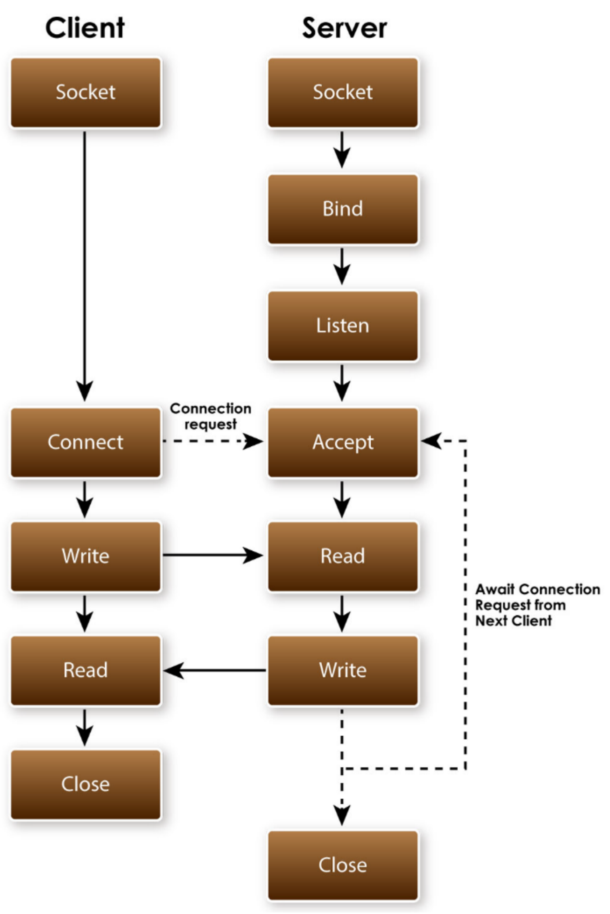
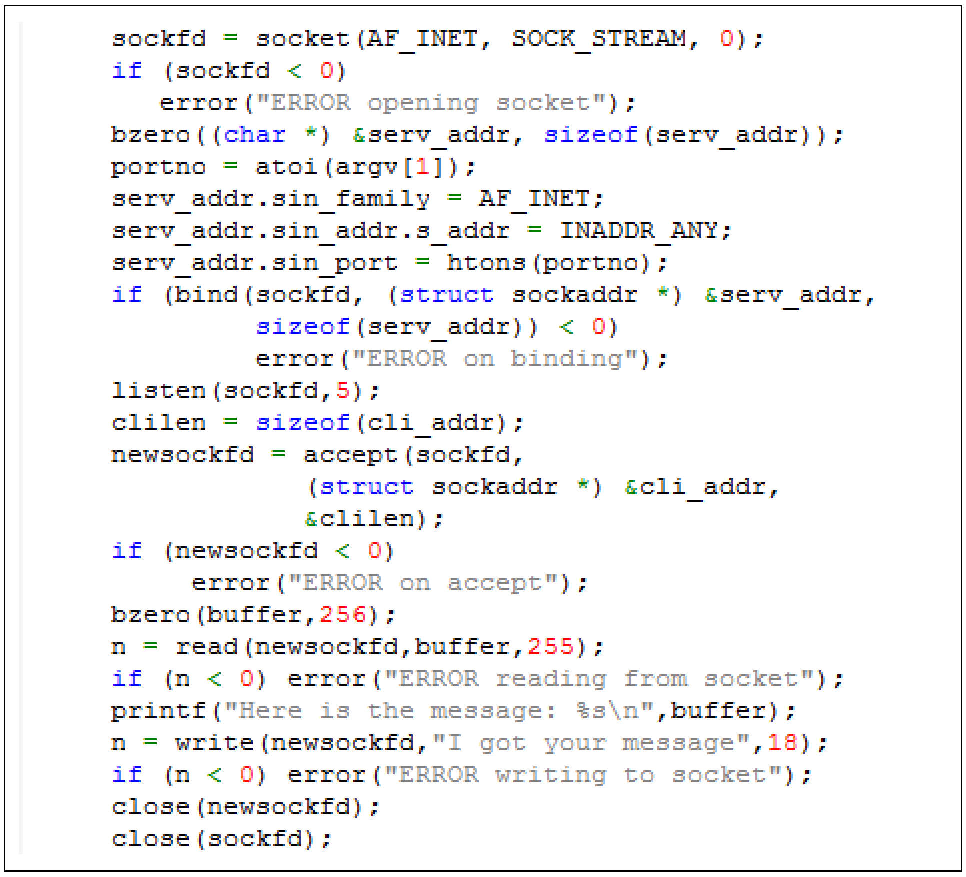


Figure 2 Client Server Communication

3. Example Code Explanation

The example design contains two projects. One is socket server project, and one is socket client project. The SOCK_STREAM socket type is used in the design. The Linux Socket Library is used to provide socket functions, so remember to include the socket API header file – socket.h.

The major function of socket server program is to create a socket server based on the given port number and the waiting a client to request to establish a connection. When a connection is established, the server is waiting for an incoming text message. When a message is received, it will show the receiver message on the console terminal, then send the message “I got your message” to the client socket, and then close the server program. **Figure 3** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK_STREAM socket, **bind** API is used to bind the socket to any incoming address and a specified port number. For connection, **listen** API is used to make the socket as a passive socket that is, as a socket that will be used to accept the incoming connection, and **accept** API is used to accept the incoming connection. The **accept** blocks until a client connects with the server. Data receiving and sending is implemented by the **read** and **write** API, and **close** is used to close the socket.

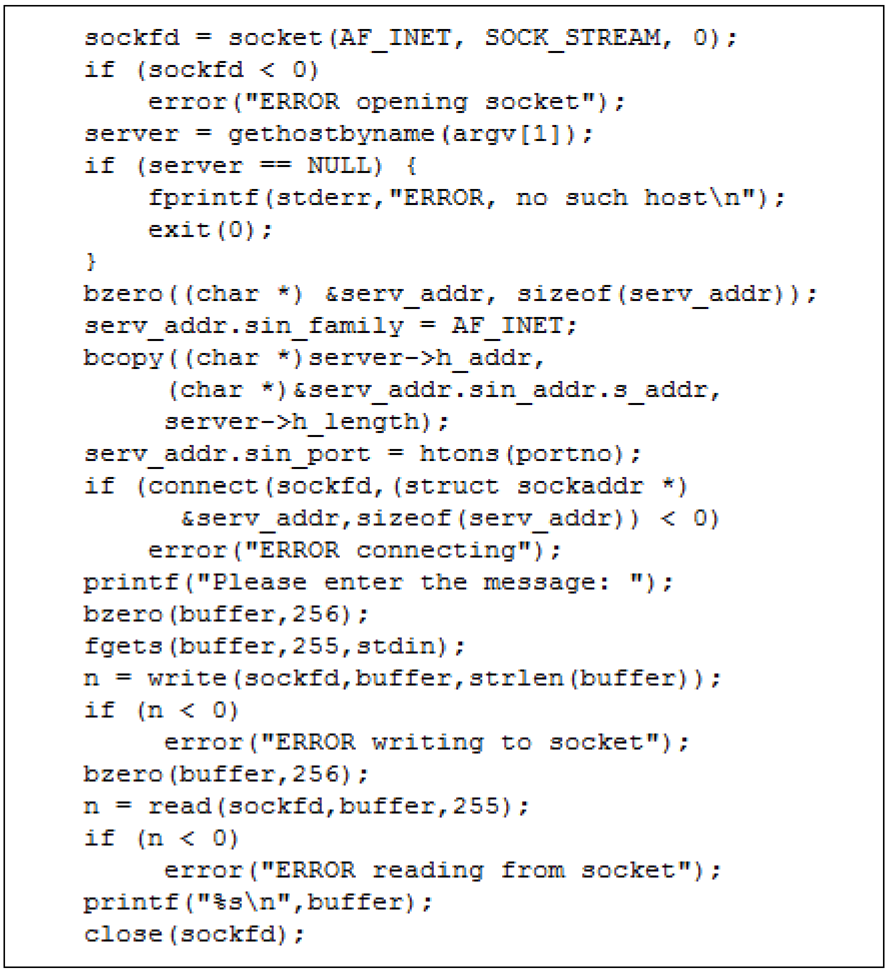


```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
   error("ERROR opening socket");
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
if (bind(sockfd, (struct sockaddr *) &serv_addr,
         sizeof(serv_addr)) < 0)
         error("ERROR on binding");
listen(sockfd,5);
clilen = sizeof(cli_addr);
newsockfd = accept(sockfd,
            (struct sockaddr *) &cli_addr,
            &clilen);
if (newsockfd < 0)
     error("ERROR on accept");
bzero(buffer,256);
n = read(newsockfd,buffer,255);
if (n < 0) error("ERROR reading from socket");
printf("Here is the message: %s\n",buffer);
n = write(newsockfd,"I got your message",18);
if (n < 0) error("ERROR writing to socket");
close(newsockfd);
close(sockfd);
```

Figure 3 Socket Sever Code

The major function of the socket client program is to create a connection based on given hostname (or IP address) and host port. When a connection is established, it will show “Please enter the message: ” message on console terminal to ask users to input a message. After get users input message, the message is sent to a remote socket server via the socket. If the remote server socket received the message, it will return a message “I got the message”. The client program will show the received message on the console terminal and exit the program. **Figure 4** shows the socket relative code statement. In the program, **socket** API is used to create a SOCK_STREAM socket,

connect API is used to connect the remove socket sever based on the given hostname (or IP4v Address) and port number. Data receiving and sending is implemented by **read** and **write** API, and **close** is used to close the socket.



```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr,"ERROR, no such host\n");
    exit(0);
}
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char *)server->h_addr,
      (char *)&serv_addr.sin_addr.s_addr,
     server->h_length);
serv_addr.sin_port = htons(portno);
if (connect(sockfd,(struct sockaddr *)
       &serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR connecting");
printf("Please enter the message: ");
bzero(buffer,256);
fgets(buffer,255,stdin);
n = write(sockfd,buffer,strlen(buffer));
if (n < 0)
     error("ERROR writing to socket");
bzero(buffer,256);
n = read(sockfd,buffer,255);
if (n < 0)
     error("ERROR reading from socket");
printf("%s\n",buffer);
close(sockfd);
```

Figure 4 Socket Client Code

4. Setup Demo

Here we show the procedure to execute the socket client-server communication demonstration. In this setup procedure, the server program is running to Intel SoC FPGA board and the Socket Client is running on MS Windows as shown in **Figure 5**. The DE10-Nano SoC FPGA board and associated Linux Console BSP are used. Putty is used as a terminal for the Linux on DE10-Nano. Note, the two Windows PC on the block diagram can be use the same one.

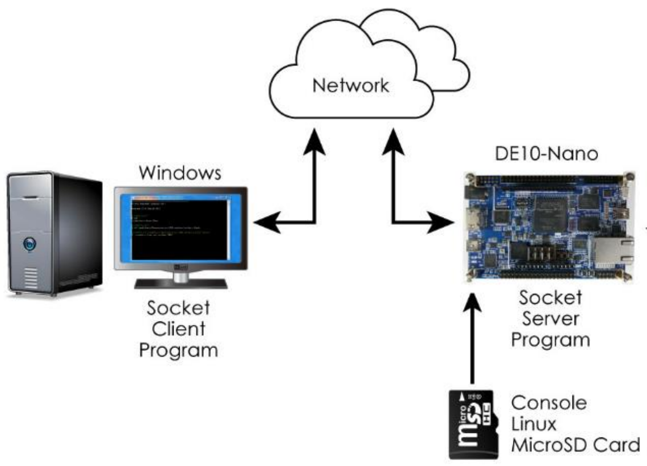
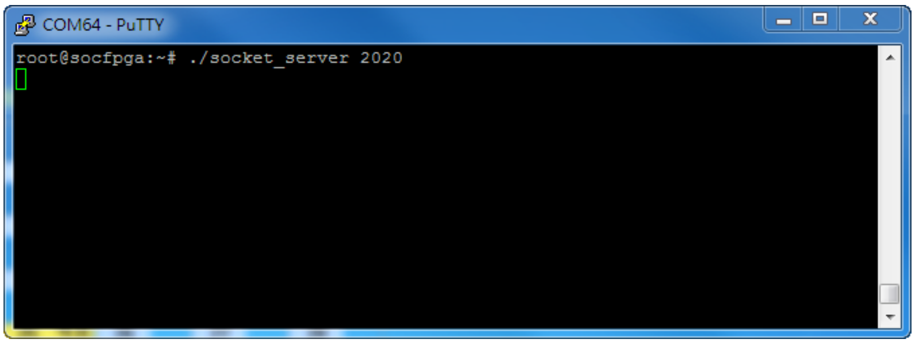


Figure 5 Demo Setup Block Diagram

Below shows the procedure to start the socket server program on the DE10-Nano SoC Board. The procedure is similar for other SoC Board.

1. Connect the DE10-Nano to Network via RJ45 port.
2. Connect the DE10-Nano UART-to-USB port to your PC Windows.
3. Power on the DE10-Nano board.
4. Copy the client server file (Demonstration/arm/socket_server/socket_server) in the example kit to the DE10-Nano board. You can use "scp" command under SoC EDS Shell to to copy the file.
5. In Windows, launch the Putty to connect DE10-Nano via the UART-to-USB port.
6. In Putty, type "./socket_server 2020" to launch the client server program with port number 2020 as shown in **Figure 6**. The port number can be any value between 2000 and 63500.

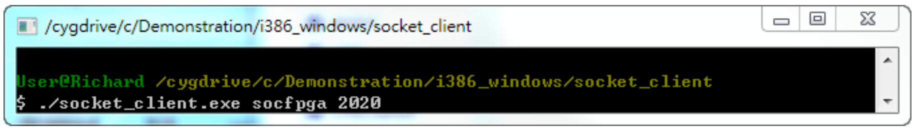


```
root@socfpga:~# ./socket_server 2020
```

Figure 6 Start Socket Server on the SoC FPGA Board

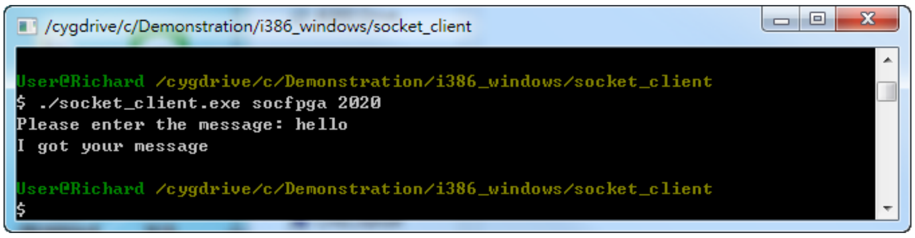
Here is the procedure to start the socket client program and communicate with the client server program:

1. Make sure the SoC EDS is installed on your MS Windows and the Windows is connected to a network.
2. Copy the client program (i386_windows/socket_client/socket_client.exe) in the example kit to your Windows.
3. Launch the SoC EDS Command Shell.
4. In the command shell, change the current directory to the directory where socket_client.exe is located.
5. Then, type "./socket_client socfpga 2020" to launch the client program to connect to the host server socfpga with port number 2020 as shown in **Figure 7**. "socfpga" is **hostname** of the Linux where the socket server is running. Also, the IPv4 Address can be used instead of the **hostname**.
6. If connection is established successfully, a prompt message "Please enter the message." will appear. Type "hello", then an echo message "I got your message" will be sent from the client server and shown on terminal as shown in **Figure 8**. At the same time, the socket server program will dump the received message at which point it is terminated as shown in **Figure 9**.



```
User@Richard /cygdrive/c/Demonstration/i386_windows/socket_client
$ ./socket_client.exe socfpga 2020
```

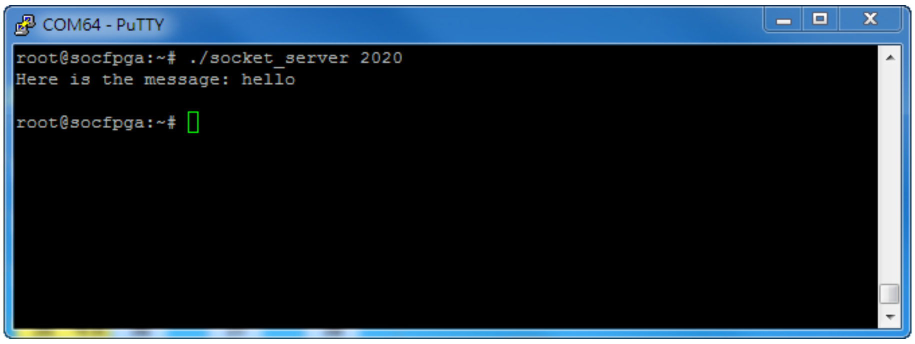
Figure 7 Start Client Program



```
User@Richard /cygdrive/c/Demonstration/i386_windows/socket_client
$ ./socket_client.exe socfpga 2020
Please enter the message: hello
I got your message

User@Richard /cygdrive/c/Demonstration/i386_windows/socket_client
$
```

Figure 8 Send Message in Client Program



```
root@socfpga:~# ./socket_server 2020
Here is the message: hello

root@socfpga:~#
```

Figure 9 Sever dumps received message

5. Source Code

The source code of the design example is located in the Demonstration folder as shown in **Figure 10**. The Demonstration folder contains three platform subfolders: **arm**, **i386_linux** and **i386_windows**. The project under the **arm** folder is designed for SoC FPGA board. The project under **i386_linux** folder is designed for Linux running on i386 PC. The project under **i386_windows** folder is designed for SoC EDS Shell running on i386 PC. Each platform subfolder contains socket_client and socket_server project folders.

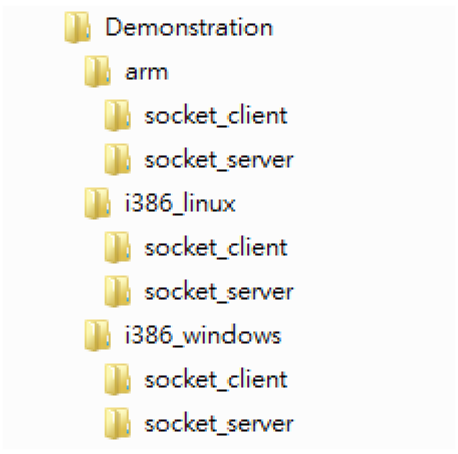


Figure 10 Source Code Folder Tree

The socket_client project includes a *Makefile* and a source file *main.c*. For different platforms, the *Makefile* content is different, but the *main.c* content is the same. The socket_server project has the file project architecture.

6. Build Program

It is easy to build the demo project just by using the “make” command. Here are the procedures to build the binary file for **arm** and **i386_window**s platforms:

1. Make sure the SoC EDS is installed.
2. Launch the SoC EDS Command Shell.
3. Go to the directory where the project source code is located

4. Type “make” to build the project.
5. For the client project, socket_client is generated for **arm** platform and socket_client.exe is generated for the **i386_windows** platform. For the server project, the socket_server is generated for the **arm** platform and the socket_server.exe is generated for the **i386_windows** platform.

Here are the procedure to build the binary file for **i386_linux** platform:

1. Make sure the GCC is installed on your Linux.
2. Go to the directory where the project source code located
3. Type “make” to build the project.
4. For the client project, the socket_client is generated. For the server project, the socket_server is generated.