SMART SPEED BUMPS

William J. Ohley, Marshall Feldman
Christopher Hunter and Frederic Bauhaud
University of Rhode Island

August 2002

URITC PROJECT NO. 536114

PREPARED FOR

UNIVERSITY OF RHODE ISLAND
TRANSPORTATION CENTER

1. Report No	2. Government Accession No.	3. Recipient's Catalog No.
URITC FY99-14	N/A	N/A

4. Title and Subtitle	5. Report Date
Smart Speed Bumps	August 2002
	6. Performing Organization Code: N/A

7. Authors(s)	8. Performing Organization Report No.
William J. Ohley, Marshall Feldman, Christopher Hunter and Frederic Bahuaud	N/A

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
University of Rhode Island, Dept. of Electrical and Computer Engineering, 4 East Alumni Ave., Kelley Annex, Kingston, RI 02881 (401) 874- 5813 email address: ohley@ele.uri.edu	N/A
	11. Contract or Grant No.: URI 536114
	13. Type of Report and Period Covered: Final

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
University of Rhode Island Transportation Center 85 Briar Lane Kingston, RI 02881	A study conducted in cooperation with U.S. DOT

15. Supplementary Notes

N/A

16. Abstract

In this project, a scale-model of a Smart Speed Bump, which can inflate or deflate on emergency vehicle request was designed. The design was also discussed in student led focus groups.

An embedded system that commands such a speed bump was developed, based on a *68HC11 evaluation board*. This board is interconnected to another board that features a Seiko iChip. The iChip is a component that performs all the TCP/IP communication and therefore, allows easy Internet connection for any embedded system. The connection between the embedded system and the real inputs and outputs was not implemented in this project phase. The program that runs the Smart Speed Bump was written in C language, with state machine architecture. It is compiled with *IAR embedded workbench*. An email report engine based on the iChip was developed with *Seiko Development Kit*. Then, it was integrated in the Smart Speed Bump program.

The focus groups were formed through students in the Department of Community Planning. Several groups questioned the effectiveness of speed bumps in general, noting that they often observe drivers slowing down to go over speed bumps and then speeding up again. A second effectiveness concern dealt with traffic congestion. The focus groups were afraid that widely used speed bumps would increase traffic congestion.

17. Key Words	18. Distribution Statement
Speed, Inflation, Speed Control, Remote, Computer Networks, Emergency vehicles, Focus Groups	No restrictions. This document is available to the Public through the URI Transportation Center, 85 Briar Lane, Kingston, RI 02881

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	86	N/A

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized (art. 5/94)

Table of Contents

Appendices

List of Figures

LIST OF TABLES

GLOSSARY

68HC11A1: Motorola basic micro controller
CD-ROM: Compact Disc Read Only Memory
CPU: Central Processor Unit
EPROM: Erasable Programmable Read Only Memory
EVB: Evaluation Board for 68HC11
ISA Bus: Industry Standard Architecture Bus
kbyte: 1 kilo byte = 1 024 bytes
MCU: Micro Controller Unit
PC: Personal Computer
RAM: Random Access Memory
ROM: Read Only Memory
RS232: Serial link standard
RTI: Real Time Interrupt
S^2B: Smart Speed Bump
SCI: Serial Communication Interface
SDK: Seiko Development Kit
SRAM: Static Random Access Memory

*.a07: File containing the code generated by the linker
*.c: File containing C language programs
*.r07: File containing the code generated by the compiler
API: Application Programming Interface
ASM: Assembly. Low-level computer language
C: High-level computer language
DOS: Disc Operating System
FTP: File Transfer Protocol
HTTP: Hyper Text Transfer Protocol
ISP: Internet Service Provider
POP3: Post Office Protocol 3
PPP: Point to Point Protocol
RTOS: Real-Time Operating System
SMTP: Simple Mail Transfer Protocol
TCP/IP: Transmission Control Protocol / Internet Protocol

ESPEO: Ecole Supérieure des Procédés Electroniques et Optiques
IAR: Software Company that makes C cross compilers
URI: University of Rhode-Island

INTRODUCTION

In order to reduce car speeding on back roads, the transportation authorities install speed bumps. But sometimes, the police or the firemen lobby against the installation of speed bumps because they slow them down, although they have an emergency. Dr. William Ohley [1] thought of a smart speed bump (S^2B), which would slow down any vehicle but emergency vehicles. It is an inflatable speed bump that can be commanded from emergency vehicles. The idea was patented and funds were found to carry out the project.

Dr. Ohley hired an intern [2] in the electrical engineering department of the University of Rhode Island (URI). The goal of this research was to define the structure of the smart speed bump system, write the specification of a scale-model speed bump and then, design the embedded system of the scale model. The tools required to develop the system were not all available so I also had to consider the purchase of some software.

This report was written to both explain the smart speed bump project and present the work that we have done.

In a first part, the overall project is explained. Then, the specification of a small-scale design is described. The third part is a short tutorial about the evaluation board that was used for the project. The forth part explains how to connect the speed bump to Internet in order to send emails. The fifth part is about the hardware design of the embedded system, using two existing boards plus one homemade board. The sixth part is about the programming of the embedded system. It also relates how a C cross compiler was chosen to carry out the programming. Finally, the last part proposes a list of improvements that can be done on the actual design.

Throughout this report, '*' symbol is used to describe a negation (NOT). If RESET is a signal, RESET* means that it is active low.

Paths, where different files are located, are given in this report. All these paths are valid only for Dr. Ohley's computer.

1. Project overhead

The main feature of the S^2B is the capability to inflate or deflate. The underlying idea is as follows: if an emergency vehicle approches a S^2B, the driver can press a button in his vehicle. This operation sends a message to the S^2B, which automatically deflates. Hence, the emergency vehicle does not have to slow down, since there is no speed bump anymore ahead. A short while later, the S^2B automatically inflates and returns into normal operation mode. The emergency vehicle request is sent from a small radio emitter, to a receiver installed next to the S^2B.

Of course, different S^2B's can be installed in a town. They constitute a network that can be remotely monitored and operated, from a server. Furthermore, any S^2B has the capability of sending reports to the server, in case of problem, failure, or for statistical purposes. A S^2B network management software is a user friendly interface that allows an operator to do so. The internet network can be used in order to transmitt data. As shown on figure 1, the S^2B network is composed of 3 different portions:

- Speed bumps
- A central server
- Remote controls

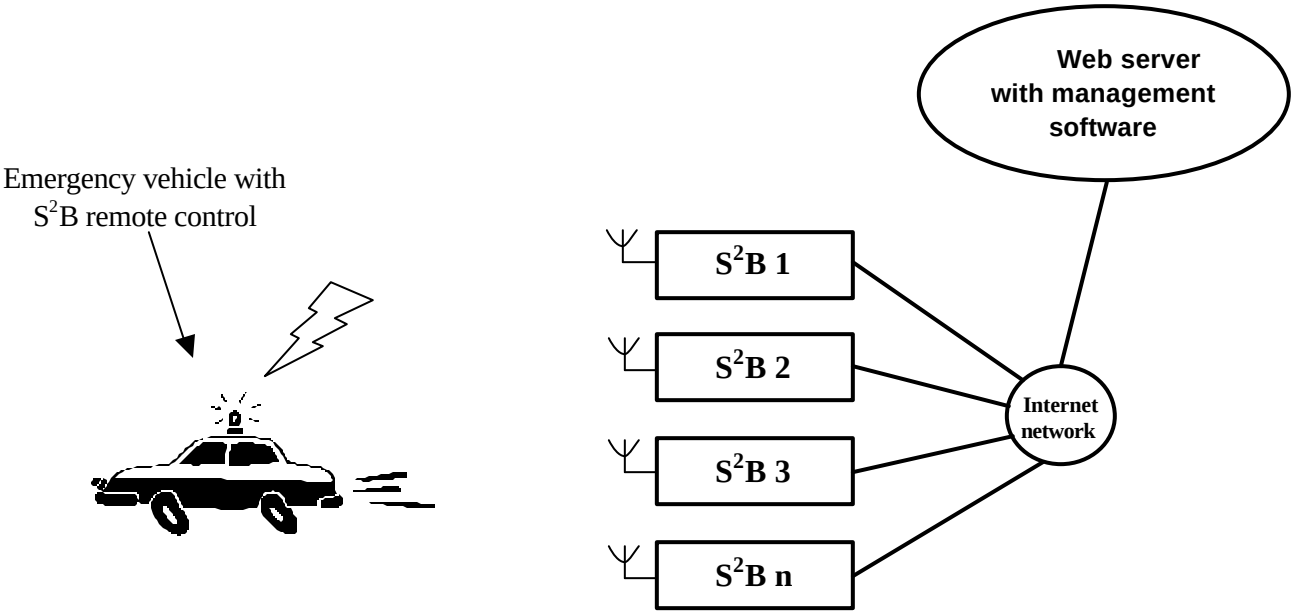


Figure 1: Main constituents of the overall S^2B project

Below, the goal of each constituent of the S^2B system is described more accurately:

Web server:

- Command each S^2B operation, independently
- Monitor S^2B's. Trigger an alarm in case of S^2B failure
- Receive e-mail reports from S^2B's, in case of failure
- Collect daily statistics from S^2B's.

On-site S^2B:

- Receive requests from emergency vehicles
- Inflate or deflate S^2B
- Monitor problems and failures (power, actuators, sensors, flashing lights…)
- Perform Internet connectivity
- Send email reports in case of S^2B failure or for statistics
- Switch in server mode for remote operation or configuration
- Allow manual command of inflation/deflation, for maintenance.

Remote control:

- Send an emergency request to all speed bumps in a radius of 1mile.

The sketch of figure 2 focuses on the on-site part of the S^2B system. Its main constituents are:

- An inflatable speed bump, to slow down vehicles
- A flashing light, to warn coming vehicles
- An antenna, to receive emergency vehicle requests
- An embedded system, to control the S^2B operation
- A pressure gauge, to know the pressure in the speed bump
- Actuators and air pumps, to inflate and deflate the speed bump.

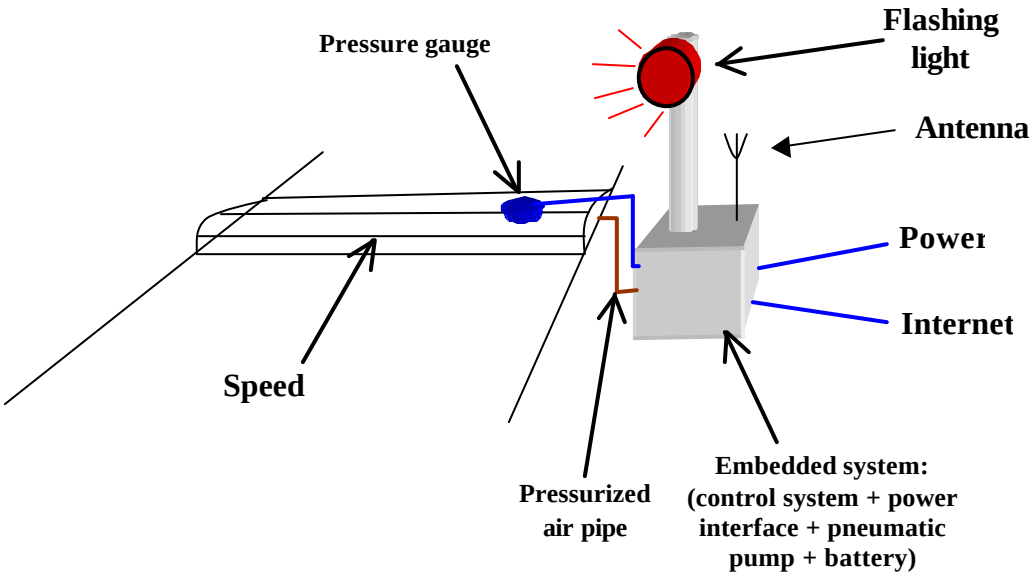


Figure 2: Sketch of an on-site S^2B

As a first step in the development of a S^2B, a scale-model has to be designed. The management software will not be designed and the overall system will only feature one speed bump. The scale-model will be used for demonstration to the police, firemen, and other people. It will be used to work with these potential users in design process. The scope of this report is the design of the embedded system of a scale-model of a S^2B.

In the next parts of this report, the on-site part of the S^2B is the only one that is considered. Hence, the words “Smart Speed Bump”, or “S^2B”, will stand for the on-site part of the overall project.

2. Specification for a scale-model design

The first specification of the project is to… define a specification because there isn't any! Follows a description of what can be done, providing that materials that are already available have to be used.

2.1 Functionalities of the scale-model

A scale-model design, including a single S^2B has to be built. As first priority, the development work will be concentrated on the embedded system. It must be modular and easily upgradeable, failsafe and highly reliable. It must perform the following functions:

- Receive requests from emergency vehicles
- Inflate or deflate S^2B
- Monitor problems and failures (power, actuators, sensors, flashing lights…)
- Perform Internet connectivity
- Send e-mail reports in case of S^2B failure
- Allow manual command of inflation/deflation, for maintenance.

The control system of the S^2B is very simple. Most of the work to be carried out concerns the Internet connectivity. The latter must be implemented and tested to send emails that can be read with a classic email box (Netscape messenger or Microsoft Outlook, for example).

2.2 Use of existing materials

As the scale-model design will be used for demonstration, materials that are already available at the university should be used. Follows a list of some devices that can be used in order to implement the first prototype:

- 68HC11 evaluation board (EVB)
- Pneumatic pump and pressure gauge of an intraaortic balloon cardiac assist system
- LED instead of the flashing lights.

The 68HC11 EVB is used each year by Dr. Ohley to teach microcontrollers. It is a classic development board (1986) but it still works fine. It has sufficient memory and is fast enough for our requirements. It will be used for the embedded system. A program will perform all the S^2B control system and provide Internet connection. The EVB is further described in part 3.

The intraaortic balloon cardiac assist system is a medical device that was designed by Dr. Ohley. It features an air pump that inflates and deflates a long balloon. Inflation and deflation can be commanded by an external source. A built-in pressure gauge yields a signal that is accessible via an external connector. So this device can be used easily to simulate an inflatable speed bump.

The use of the intraaortic balloon cardiac assist system was part of the initial specification. Unfortunately, it was not available during my internship. As a consequence, the characteristics of the inputs and outputs were not known. One way to do something anyway was to simulate inputs with switches, and outputs with LEDs. This solution was retained, and it is still the way it works, so far.

2.3 Inputs and outputs of the scale model

Considering the needs for the S^2B, inputs and outputs of the embedded system were defined. They are as listed in table 1 and 2.

Inputs:

Name	Physical origin	Utility
High pressure in speed bump	1 switch simulating a high pressure signal from a pressure gauge	Stop inflation when the required pressure is reached. Detect punctures
Low pressure in speed bump	Same switch as above, but in the other position	Stop deflation when pressure is low enough
Emergency request	1 push button simulating a radio signal from an emergency vehicle	Command the deflation of the speed bump
Inflation button	1 push button	Manual command of inflation, for maintenance
Deflation button	1 push button	Manual command of deflation, for maintenance
Modem incoming stream	Modem	Reception of data from Internet

Table 1: Inputs of the embedded system of the small-scale design

Outputs:

Name	Physical device	Utility
1 pneumatic pump	2 LEDs simulating the pump (inf & def)	Allow inflation or deflation of the speed bump
1 flashing light	1 LED simulating the flashing light	Flash when the speed bump is inflated
Modem outgoing stream	Modem	Emission of data to Internet, for email report in case of problem or failure of the S^2B

Table 2: Outputs of the embedded system of the small-scale design

Before starting the design of the scale-model, it was necessary to study the 68HC11 evaluation board. It was also required to find information on how to connect an embedded system to the Internet. That is what is explained in the next two parts of this report.

3 The 68HC11 evaluation board

This board is used for the embedded system of the S^2B. Follows a quick description of its main characteristics and how to operate it.

3.1 Hardware description

The EVB was designed to debug and evaluate user code in a target system environment. The EVB emulates the single-chip mode of operation, even though it operates in the expended multiplexed mode of operation. The EVB was designed along with a monitor/debugging program called BUFFALO. This program is contained in an EPROM, external to the microcontroller (MCU). The user program is contained in an external RAM, after download from an external host computer. Hence, user program is lost in case of power failure.

The EVB specification is described in table 3:

Characteristics	Specifications
MCU	MC68HC11A1FN
PRU	MC68HC24FN
ACIA	MC68B50
I/O ports: Terminal Host computer MCU extension	 RS232C compatible RS232C compatible HCMOS-TTL compatible
Temperature: Operating Storage	 0 to 50 degrees C -40 to +85 degrees C
Relative humidity	0 to 90% (non-condensing)
Power requirements	+5 Vdc, 0.5 A maximum +12 Vdc, 0.5 A maximum -12 Vdc, 0.5 A maximum
Dimensions: Width Length	 17.8 cm 11.75 cm

Table 3: Specification of the EVB [3]

A block diagram of the EVB, a component location diagram, a pin assignment of connector P1, a memory map and an interrupt vector jump table are provided in appendix C.

The EVB is used with a RS232 terminal and a power supply coming from room 101, Kelley building. Target boards with 6 displays connected to PORTB are also available for experimentations.

Connector P1 is used for target system to EVB interconnection. Connector P2 is used to connect the terminal. This terminal is used as an I/O device for user programs. Connector P3 is used to connect the serial port COM1 of the host computer. This computer is used to download programs in the EVB RAM.

Before intending any serial communication, make sure that jumper 5 (J5) is set to 9600 bauds.

3.2 Operating instructions

3.2.1 Downloading a program in memory

The downloading operation enables the user to transfer information from a host computer to the EVB, using the *LOAD T* command. The load command moves data information in S-record format (Motorola), from the external host computer to the EVB user RAM. The extension of S-record format files is either *.s09* or *.a07*. These files are generated by a C cross compiler, such as *IAR workbench for 68HC11*, described in more details in chapter 6.2.

A terminal emulator, such as Hyperterminal (C:\program files\accessories\hyperTrm.exe), is used to transfer files. It must be configured as follows:

Connect using:	direct to COM1
Port settings:	9600 bauds
Data bits:	8
Parity:	none
Stop bit:	1
Flow control:	hardware

The default settings of other fields are OK, so they don't need to be changed.

Save these parameters in a new profile called *68HC11evb.ht*. Then, when loading this profile, the link with the EVB is automatically opened. Turn on the EVB or reset it. The BUFFALO welcome message should appear in the Terminal window. It is as follows:

BUFFALO 2.5 (ext) - Bit User Fast Friendly Aid to Logical Operation

Press *Enter* and a prompt should be displayed as follows:

>

That means that the BUFFALO operating system is now ready to receive commands and interpret them. The first step is to download the user program in RAM. To do so, type *LOAD T* and then, press *Enter*. Now, the program waits for the user to send a S-record file. In the menu, choose *Transfer/Send text file…*. Select the file to be sent and click on the *open* button. The download procedure is started. If it was successful, the following should be displayed in the window:

done

>

If any other character appear, the file was not downloaded properly: the program should be reconsidered because it contains errors.

3.2.2 Debugging

Table 4 lists a few commands that can be used to execute and debug a program. The whole set of instructions is available in the M68HC11EVB user's manual [3], p 4.4.

Command	Description
g <address>	Execute program. <address> is the starting address where user program execution begins (C010 in our design).
md <address1> <address2>	Memory display from <address1> to <address2>
mm <address>	Memory modify.
rm	Register modify (P, Y, X, A, B, C, S).
br <address>	Breakpoint set

Table 4: BUFFALO program commands

4 Connecting the 68HC11 to Internet

The S^2B features an Internet connection. This part explains how this can be easily implemented by using Seiko iChip. There is also a quick description of the Seiko Development Kit (SDK). Finally, a program of an email report engine is explained.

4.1 Internet strategy

The S^2B has the capability of sending data reports to a server and being remote configured. Telephone lines are convenient to transport data since they are available in all streets. As the data to be transmitted needs to be highly reliable, an Internet protocol, such as TCP/IP should be used.

Hence, an email communication engine that can be embedded inside a device has to be designed. It dialups, connects to an Internet service provider (ISP), and sends emails over the Internet. In this first version of the S^2B, reception of emails is not implemented.

TCP/IP is a rather complicated stack of protocols that ensures reliable data transmission between two peers. Post Office Protocol 3 (POP3) and Simple Mail Transfer Protocol (SMTP) are used to receive and send emails between a mail server and a mail host. These protocols are briefly explained in the next chapter.

Two different strategies were thought of in order to develop an email engine:

1. The 68hc11 is linked directly to a modem. Therefore, the appropriate program has to manage all handshaking signals with the modem and all TCP/IP. This can be achieved in C programming but C++ classes, like Winsocket, cannot be used, since the code generated is far too big to be uploaded in an embedded target. All the code has to be written. This is very complicated and time to design would be very long!
2. The 68HC11 is interfaced with a modem via an Internet connection chip, like the S7600A, from Seiko. This chip, called iChip, manages all modem handshaking and TCP/IP, making Internet connection of embedded systems much easier. However, iChip is new and not many people know how to use it.

The second solution was chosen. A Seiko Development Kit (SDK) was ordered for $199, together with a US Robotics 56k external modem, for $110.

Sending and receiving emails over the Internet requires an ISP. The computer service of URI has an ISP service that allows dialup line connection. An account was opened on **etal** server. This account will be valid one year and shall be renewed afterwards. The connection parameters are as follows:

Phone number: 874-8900 or 4-8900 if on campus
User name: fred@etal
Password: 1176
Email address: fred@etal.uri.edu
IP address: 131.128.1.21
POP3 port: 110
SMTP port: 25
HTTP port: 80

4.2 TCP/IP, SMTP and POP3 [4]

A basic knowledge of TCP/IP is required in order to implement an email engine, using the Seiko S7600A iChip. In our case, a mail has to be sent.

First, there is a protocol for mail. This defines a set of commands which one machine sends to another, e.g. commands to specify who is the sender of the message, who is the recipient, and then the text of the message. However this protocol assumes that there is a way to communicate reliably between the two computers. SMTP and POP3 simply define a set of commands and messages to be sent (c.f. Appendix D for a list of the commands). They are designed to be used together with TCP and IP. TCP is responsible for making sure that the commands get through to the other end. It keeps track of what is sent, and retransmits anything that did not get through. If any message is too large for one datagram, e.g. the text of the mail, TCP will split it up into several datagrams, and make sure that they all arrive correctly. Since these functions are needed for many applications, they are put together into a separate protocol, rather than being part of the specifications for sending mail. TCP can be thought of as forming a library of routines that applications can use when they need reliable network communications with another computer. Similarly, TCP calls on the services of IP. Although the services that TCP supplies are needed by many applications, there are still some kinds of applications that don't need them. However there are some services that every application needs. So these services are put together into IP.

This strategy of building several levels of protocol is called "layering". SMTP, POP3, TCP, and IP are separate "layers", each of which calls on the services of the layer below it. Generally, TCP/IP applications use 4 layers:

- An application protocol such as SMTP, POP3, FTP or HTTP
- A protocol such as TCP that provides reliable delivery of datagrams
- IP, which provides the basic service of getting datagrams to their destination
- Point-to-Point protocol (PPP), needed to manage a specific physical medium.

The Internet Protocol address (IP address), and the Port number are the only parameters needed in order to access another computer. The IP address looks like 131.128.1.21.

Most systems have separate programs to handle file transfers, remote terminal logins, mail, etc… When connecting to 131.128.1.21 to send an email, SMTP must be used to establish the connection. Having “well-known sockets” for each server does this. TCP uses port numbers to keep track of individual conversations. User programs normally use more or less random port numbers. However specific port numbers are assigned to the programs that sit waiting for requests. For example, if a mail has to be sent, a program using SMTP will be started. It will open a connection using some random number, say 1, for the port number on its end. However it will specify port number 25 for the other end. This is the official port number for the SMTP server.

4.3 The iChip s7600A

The iChip contains TCP/IP Protocol stacks that act as an accelerator between a microcontroller and Internet. It is a completely self-contained, drop-in solution for any device requiring networking connection. It provides a high connect speed with low power consumption, integrating full TCP/IP, PPP and UDP protocols, and 10 kbytes of on-chip SRAM that operates

as a buffer. The S7600A also supports a microcontroller interface via a register set, and connection to the physical transport layer interface, i.e. the modem. It is represented in figure 3.

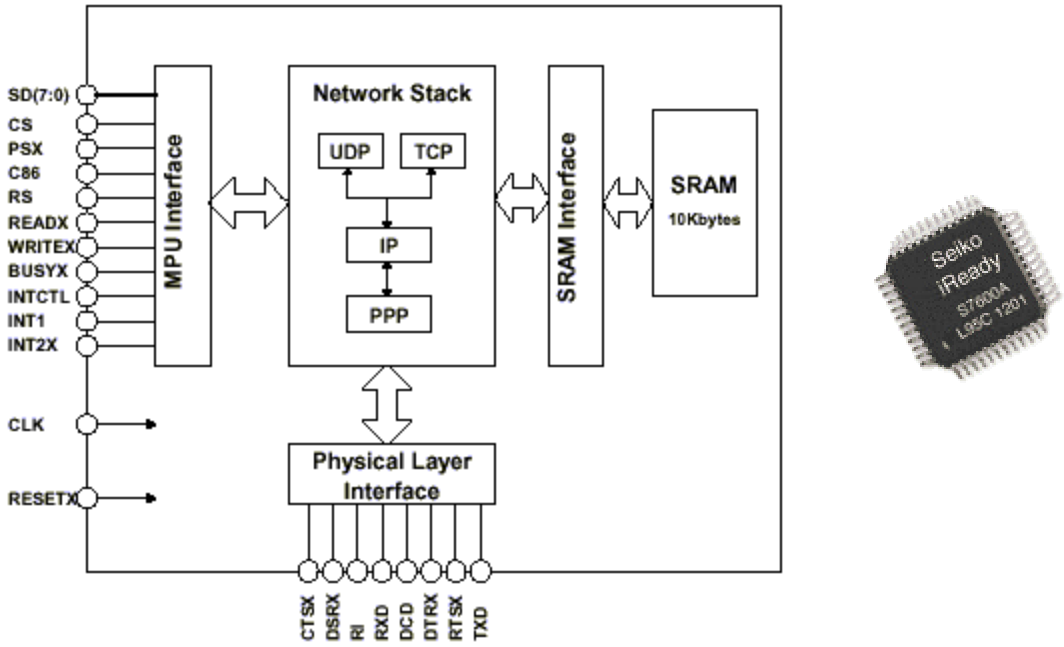


Figure 3: Block diagram of the Seiko S7600A iChip [5].

The iChip is powered with Vdd = +3.3V. Its I/O can vary between 0 and +Vdd. It can be operated up to 5MHz. The interface with the MCU can be serial or parallel. It is only available in surface mount packages (48 LQFP). The complete pin description is provided in Appendix E, as well as a description of the register set, and an explanation of a R/W cycle.

Not much information is available about the iChip. However *Mike's Seiko S7600 web page* [6] provide many useful documents and a forum of iChip developers.

4.4 The Seiko Development kit (SDK)

4.4.1 Hardware specification

The SDK contains everything that is needed to interface between a desktop PC and a modem in order to perform software development and to demonstrate Internet connectivity. It comprises an ISA bus interface and two S7600A iChips. The first iChip is interfaced to the ISA bus for software development on a PC and routed to a DB9 connector, for the modem connection. The second one is routed to a standard 50-pin header and mounted in a sea of holes, providing prototyping capabilities. The kit also comes with a DB9-DB25 modem cable and a CD-ROM containing datasheets, SDK board schematic, application programming interfaces (APIs) in C source code, and a sample program. This is shown in figure 4.

Figure 4: The Seiko Development Kit with its package [7]

The board and its drivers are installed on Dr. Ohley's PC. The US Robotics external modem is connected to the board with the DB9-DB25 cable. The modem is also connected to a telephone socket.

The S7600A SDK board is equipped with several connectors. The ISA bus connector CN1 is provided to interface the board to an ISA bus for PC. The CN3 provides direct access to the second S7600A signals for user's appreciation. The pin assignment of CN3 is provided in appendix F.

The S7600A SDK board includes the glue logic to interface between S7600A and ISA bus in a CPLD. This logic is simple and provides registers to access the S7600A. Table 5 shows the mapping of the I/O registers in the PC memory map.

I/O address	Description	Read/Write	Notes
0x0430	Set index register address	R/W	
0x0431	Read/Write data at S7600A index register	R/W	
0x0432	Busy signal of S7600A (BUSYX)	R	MSB, D7

Table 5: I/O address mapping of the SDK

4.4.2 Software design

The SDK comes with a set of APIs in C source code [16]. Any software application can use them to control the functionality of the iChip, at a high level of abstraction. They are platform, processor, and operating system independent. Furthermore, they are designed to support usage in highly integrated embedded systems that run in a multitasking environment [7]. The ItaskAPI provides a small real-time operating system (RTOS) that would meet our requirements.

Unfortunately, installing the APIs, setting the environment, and then building the application is not trivial. And there is very little information available. After many tries, and a phone call to George Ho [8], the designer of the SDK, it was decided not to use the APIs provided. They are designed for complicated systems, so they are not required in our case.

Instead, a C program given as an example can be used as a starting point. It can be found in the iChip CD-Rom and is called *S7600r1.c*. The header file named *custom.h* is also needed.

The program is supposed to connect a daytime server, over the Internet, and display the time sent by the server. A project called *"s7600a server connection"* was created with Microsoft

Visual C++ 6.0, in order to compile the files. The compilation worked, but not the program. The dialup procedure worked, but not the TCP connection. Indeed, after connecting to the ISP, the program expected the server to use PPP. But URI server uses Telnet. The program was modified to ask the server to switch from Telnet to PPP mode. Then, it worked fine. That confirms that the entire communication link is properly set and that any data can be received from the Internet, using the iChip. However, sending data still had to be implemented.

The next program that was written is an email report engine. A new visual C++ project was created and called *"s7600a email report engine"*. The algorithm of the program is represented by a flowchart, in figure 5.

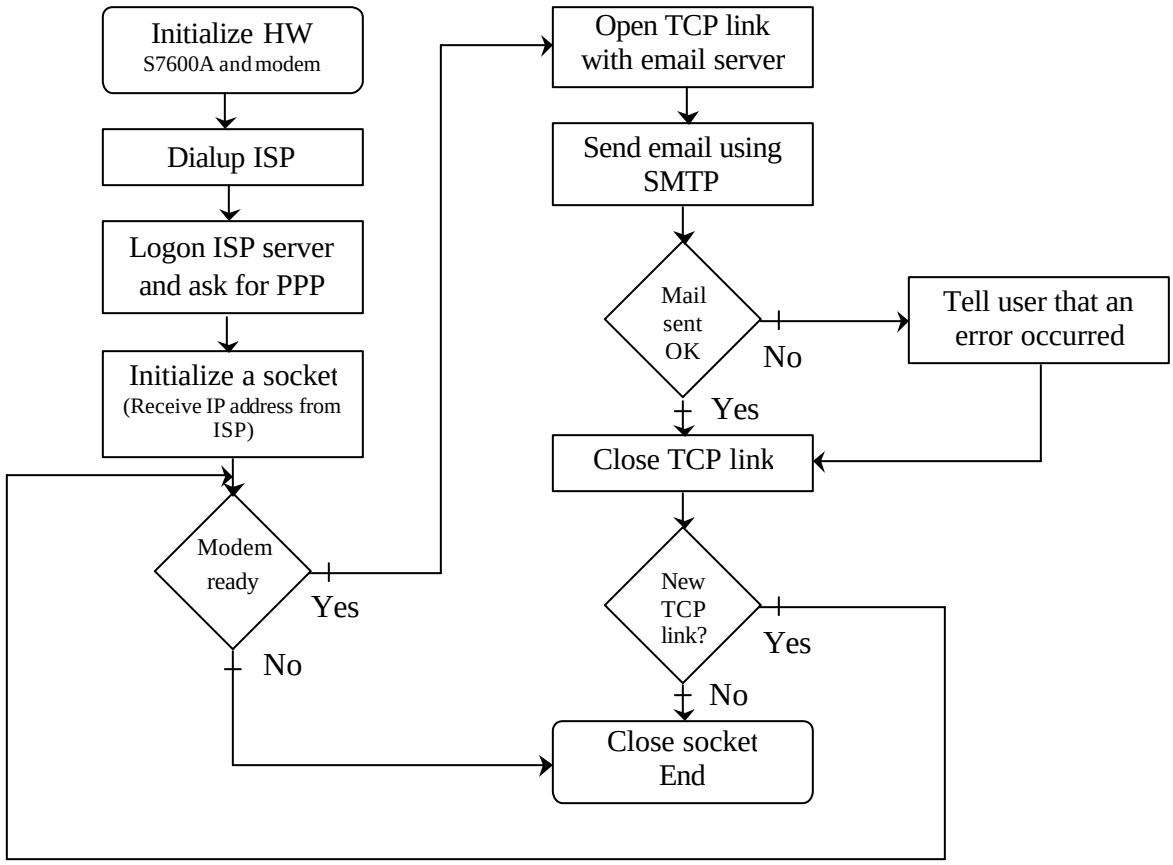


Figure 5: Algorithm of the email report engine.

Function declarations are in *custom.h* and function definitions are in *S7600A.c*. Table 6 lists all the functions that were required to write this program, and it explains their goal. A listing of this program is available in Appendix G.

Function name	Function description
void SendString(char* st)	**Send string on serial port** st: string to be sent to the modem
void DialUp(unsigned long BAUD , char * isp_num)	**Dialup to the service provider** BAUD: modem transmission bit rate Isp_num: phone number of the ISP
void Telnet(char* name, char* password)	**Telnet connection to remote server** name: username for ISP account password: password for ISP account
void HwSocketInit(char * name, char * password)	**PPP connection** name: username for email account password: password for email account
void HwSocketClose(void)	**PPP disconnect**
void abort(void)	**Close PPP connection**
int HwTcpOpen(void)	**Require TCP connection to the server as a client**
int HwTcpClose(void)	**Force TCP to CLOSE status**
int HwTcpState(int)	**Obtain TCP State**
int HwTcpSend(char* text)	**Send data to socket**
int HwTcpRcv(void)	**Display Data from Socket**
int nstep(void)	**Return FALSE if user pressed any key**
void x_write(BYTE addr, BYTE data)	**1 byte data write driver for S-7600A SDK Board for ISA BUS** addr: select register of iChip data: data to write in the iChip register
BYTE x_read(BYTE addr)	**1 byte data read driver for S-7600A SDK Board for ISA BUS** addr: select register of iChip return value of register
void timer(int sec)	**Timer** sec: time in seconds
int SendMail(char* from, char* to, char* subject, char* message)	**Send email report** from: email address of the sender to: email address of the recipient subject: subject of the email message: email message

Table 6: List of the functions and explanation of their I/O.

This program works fine as long as everything works OK during connection. Indeed, there are only very few tests that are performed during connection process. For example, if logon to server fails, or if an email could not be sent properly, this is not detected so no particular action is taken and normal execution of the program carries on.

To run the program, make sure the modem is on and properly connected to both the SDK board and a phone line. Then, click “execute” in Visual C++. The modem dials-up to an ISP, and send an email. Most of the parameters are initialized in custom.h (ISP phone number, password, username, sender, recipient, message, subject) and can be easily changed. The IP address of the email server can be changed in the function called HwTcpOpen().

5 Design of the embedded system

The core of the embedded system of the S^2B is the 68HC11 EVB. In order to connect it to Internet, it has to be interfaced with the second iChip, on the SDK board. Furthermore, inputs and outputs of the S^2B have to be connected to the EVB. This part explains how that was implemented.

5.1 Connecting SDK to EVB

5.1.1 Signals for Read/Write cycle of the iChip in 68k mode

The iChip supports two MPU interfaces: parallel and serial. In parallel mode, iChip can interface with x80 family MPU and 68k family MPU. In the scope of this project, the iChip must be configured in *68k family MPU mode*, using the parallel interface. This mode can be selected by pulling the **C86** and **PSX** input pins high. In this mode, the address and data are multiplexed into a single 8 bits bus. The Read/Write cycle is explained in appendix E, figures 20 and 21. The signals needed are listed in table 7:

iChip Signal name	EVB signal name	I/O	Description
CS	CE*	I	Chip selection input
RS	A0	I	Register selection input (INDEX or DATA)
WRITEX	R/W*	I	Read/Write selection input
READX	E	I	Enable input
SD7 to 0	D0 to D7	I/O	Data bus
BUSYX	PD5	O	Busy indicator output
CLK		I	Clock input

Table 7: Signals necessary for a R/W cycle. I/O are from the iChip point of view.

CS:

On the EVB, address decoding is accomplished via a MC74HC138 device and is segmented into 8 kbytes blocks. This device generates 5 CE* signals. One of them is used to select an optional 8 kbytes RAM. As no extra RAM is needed for the S^2B design, this CE* is used to select the iChip. Jumper header J3 of EVB must be put to enable CE*. The CE* signal is active low for addresses from 0x6000 to 0x7FFF. This is large enough to operate the iChip since there are only two registers to address. The CE* signal has to be inverted because the CS input of the iChip is active high. This is done with one NAND gate of a SN74LS00. The resulting hardware design is as shown in figure 6:

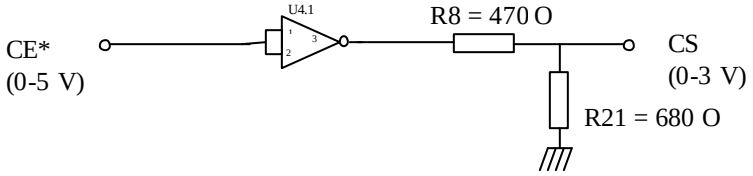


Figure 6: Shaping of a CE* signal from the EVB.

All I/O from the EVB are 0-5 V. But the iChip is a 3 V component so all input signals must be lowered to 3 V. This is why a simple voltage divider was added in output of the NAND gate. With R21 = 680 O and R8 = 470 O, the maximum load current for the gate is i = 5/(R21 + R8) = 4.3 mA. Output signals of the iChip do not need to be increased to +5 V, as

the EVB uses CMOS logic. A signal greater than 2.5 V is good enough to code a 1. So the voltage divider can even be used for bi-directional signals. In this case, if the iChip outputs a signal, it provides a load current of i = 3/R21 = 4.4 mA.

RS:

The Register Select signal is used to select the INDEX register or the DATA register of the iChip. As there are only 2 registers to address, the LSB of the EVB address bus can be used for this signal. A voltage divider is also required. Then, the memory location of these 2 registers becomes as follows:

INDEX: 0x6000
DATA: 0x6001

WRITEX, READX, D0-7:

These signals come straight from the EVB. A voltage divider has to be added for each signal.

BUSYX:

This output signal of the iChip is low during data write and read phases. The 68HC11 needs to monitor it before sending or receiving new data. This signal can be applied to an input port of the 68HC11. Pin 5 of port D was chosen. However, port D is a bi-directional I/O port [9]. It needs to be configured as an output port. This is done by resetting DDRD register.

CLK:

The iChip does not need to be synchronized with the MCU. The higher its operating frequency, the higher its power consumption. A frequency of 256 kHz is enough to generate a 56 k baud rate towards the modem. The maximum operating frequency is 5 MHz. The EVB clock rate is 8 MHz. This is too high so it cannot be used for the iChip. Instead, the PDI1051, 1 MHz built-in oscillator was chosen. This clock rate is interesting because it is also the clock rate of the first iChip, on the SDK. This choice makes portability of programs easier. Indeed, the values of the Clock and Baud rate registers of the iChip remain the same, whatever is the iChip used.

The oscillator output varies between 0 and 5 V: a voltage divider is required here as well.

5.1.2 The reset signal

The **RESET*** signal of the EVB can be used to reset the iChip (pin **RESETX**). This is a 0-5 V signal so it has to be lowered to 0-3 V. But this signal does not come from a buffer so a voltage divider cannot be added directly to it. Instead, the RESET* is applied to a succession of 2 NAND gates which are used as buffers. This solution was chosen because 3 NAND gates were not used in the SN74LS00 used for inverting CE*. Then, the voltage divider can be added. The resulting schematic is as shown in figure 7:

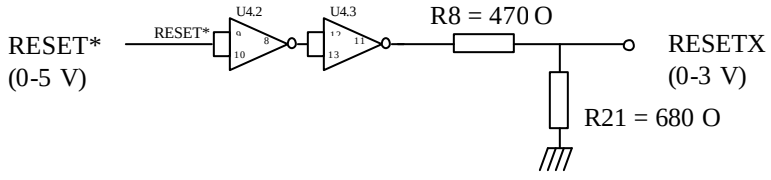


Figure 7: Two NAND gates are used to buffer RESET*

5.1.3 RS232 drivers

The iChip cannot directly drive a modem with RS232 signals. A RS232 driver must be added. 5 input drivers and 3 output drivers are necessary to interface all the handshaking signals between the iChip and the modem. Furthermore, all these signals must be 0-3 V between the iChip and the drivers. The ICL3241 is a device that meets these requirements. Unfortunately, it only exists in surface mount packages. It was ordered anyway and a SOIC to DIP adaptor was made. The ICL3241 also requires 4 external capacitors to generate the ±12 V required for the RS232 link.

Finally, the signals for the modem are connected to a DB9 connector, so that any modem can be easily installed/uninstalled.

5.1.4 Power supply

A +5 V power supply is available on the EVB. However, the iChip is a 3 V device. So a voltage regulator is required. The iChip power requirements are as low as 3 mW when transmitting and less than 0.5 mW when on standby. The load made by the voltage divider, on the BUSYX signal, needs about 4mA. The RS232 driver needs 1mA when operating. The oscillator needs 50 mA. A LED is added after the regulator and needs 15 mA. Hence, the voltage regulator must supply at least 70 mA. The ICL 3241 3.3V, 700 mA micropower low dropout voltage regulator, is sufficient for our needs.

5.2 Input/Output signals of the S^{2}B

5.2.1 Switches as inputs

The characteristics of the inputs were still not defined when this board was under development. It was decided that all the inputs would be simulated with switches. These switches are connected to Port D of the 68HC11. Port D must be initialized as an input by writing DDRD to 0. Furthermore, the serial communication mode must be disenabled or pin 1 of Port D will still be an output. This is done by resetting SCCR2. The 5 input signals are as described in table 8:

Switch name	Bit number of Port D	Description
BTN_MAINT	0	Put system in maintenance mode. Switch.
EM_REQUEST	1	Simulates an emergency vehicle request. Push button. Normally opened.
BTN_INF	2	Manually inflates the speed bump. Push button. Normally opened.
BTN_DEF	3	Manually deflates the speed bump. Push button. Normally opened.
HI_PRESSURE and LOW_PRESSURE	4	Simulates a high pressure signal from pressure gauge in one position, and a low pressure signal in the other position. Push button. Bistable.

Table 8: Port D is configured as an input port. Each switch is assigned to one bit.

Inverting Schmitt triggers (CD40106) shape each signal before it is input on the EVB. However, this is not an anti-bounce system so these switches cannot be used for all purposes, e.g. to count events. Pull-up resistors put in input of the Schmitt triggers force its outputs to 0 at rest. The schematic is given in figure 8:

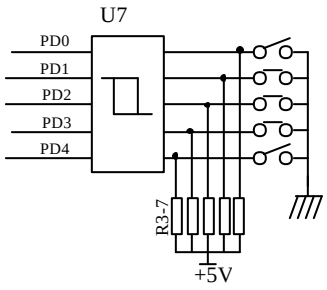


Figure 8: Circuitry used to simulate inputs of the S^{2}B

5.2.2 LEDs as outputs

As for the inputs, the characteristics of the outputs were still not defined when this board was under development. It was decided that all the outputs would be simulated with LEDs. They are connected to Port B of the 68HC11. The 8 output signals are described in table 9:

Switch name	Bit number of Port B	Description
INFLATE	0	Simulate pump that inflates the speed bump
DEFLATE	1	Simulate pump that deflates the speed bump
FLASH	2	Simulate flash lamp when speed bump is inflated
COMMU_LED	3	Turned on when system is in communication mode
MAINT_LED	4	Turned on when system in maintenance mode
TCP_LED	5	Turned on during TCP communication
MAIL_LED	6	Turned on when sending email
ERROR	7	Turned on when an error occurred

Table 9: Port B in an output port. Each bit is assigned to one LED.

The LEDs are in fact a bar graph, as shown on figure 9. A bussed resistor network drops the output voltage of the EVB, from 5 V to 1.5 V. Each resistor is 150 O.

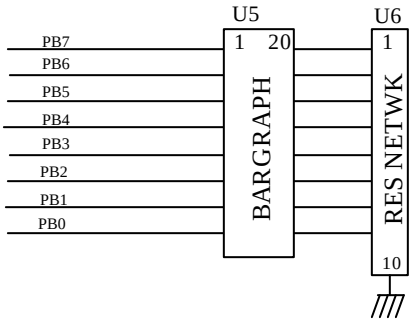


Figure 9: Circuitry of the output LEDs that simulate outputs of the S^{2}B.

5.3 Connectors and cables

The SDK has a connector (CN3) that provides direct access to the second iChip. It is a 50 contacts, pin connector. A similar connector (P2) is used on the interface board. An 18-inch, gray socket connector to socket connector ribbon makes the link between the SDK and the interface board. The pin assignment of CN3 is given in Appendix F.

The EVB has a connector (P1) that provides access to most of the signals needed. It is a 60 contact, pin connector. A similar connector (P1') is used on the interface board. A 5-inch, gray socket connector to socket connector ribbon makes the link between the EVB and the interface board. The pin assignment of P1 is given in Appendix C. Pins 50 to 60 are not connected or carry unwanted signals so the wires were cut. However, signals A0, CE* and D0-7 are only available at the optional RAM socket. Signal R/W* is available on pin 13 of U9 (MC68B50P). Hence, they were applied to wire 50 to 60 of the flat ribbon.

The wire assignment is as listed in Table 10:

Pin name on EVB	Pin number on P1'
A0 (from optional RAM)	50
R/W* (from MC68B50P)	51
CE* (from optional RAM)	52
D7-0 (from optional RAM)	53-60

Table 10: Pin interconnection between EVB and SDK

The wires going to the optional RAM terminate in an IC socket. Connections are made much easier that way. The red wire corresponds to pin 14 of the SDK socket.

5.4 Schematics, part list and costing of the board

5.4.1 Global architecture of the embedded system

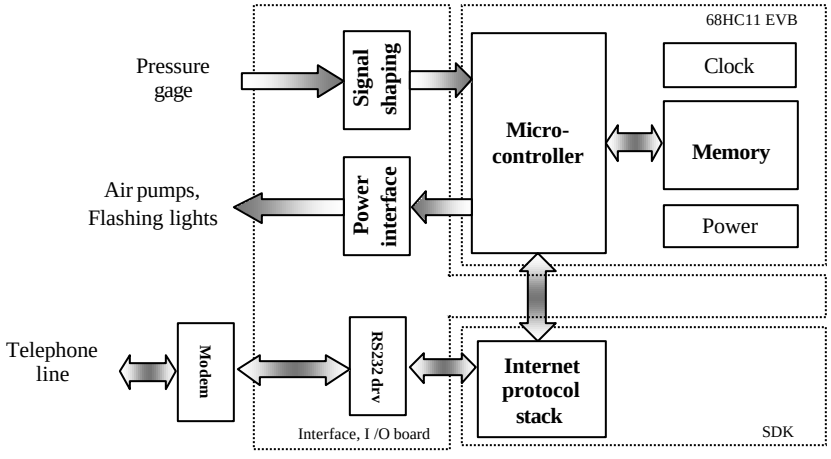


Figure 10: Architecture of the embedded system

The block diagram of figure 10 represents the architecture of the embedded system.

5.4.2 Schematics of the overall board

Figure 11 represents the overall schematic of the interface board. The 4 gray blocks called block1 represent voltage dividers. They are all the same.

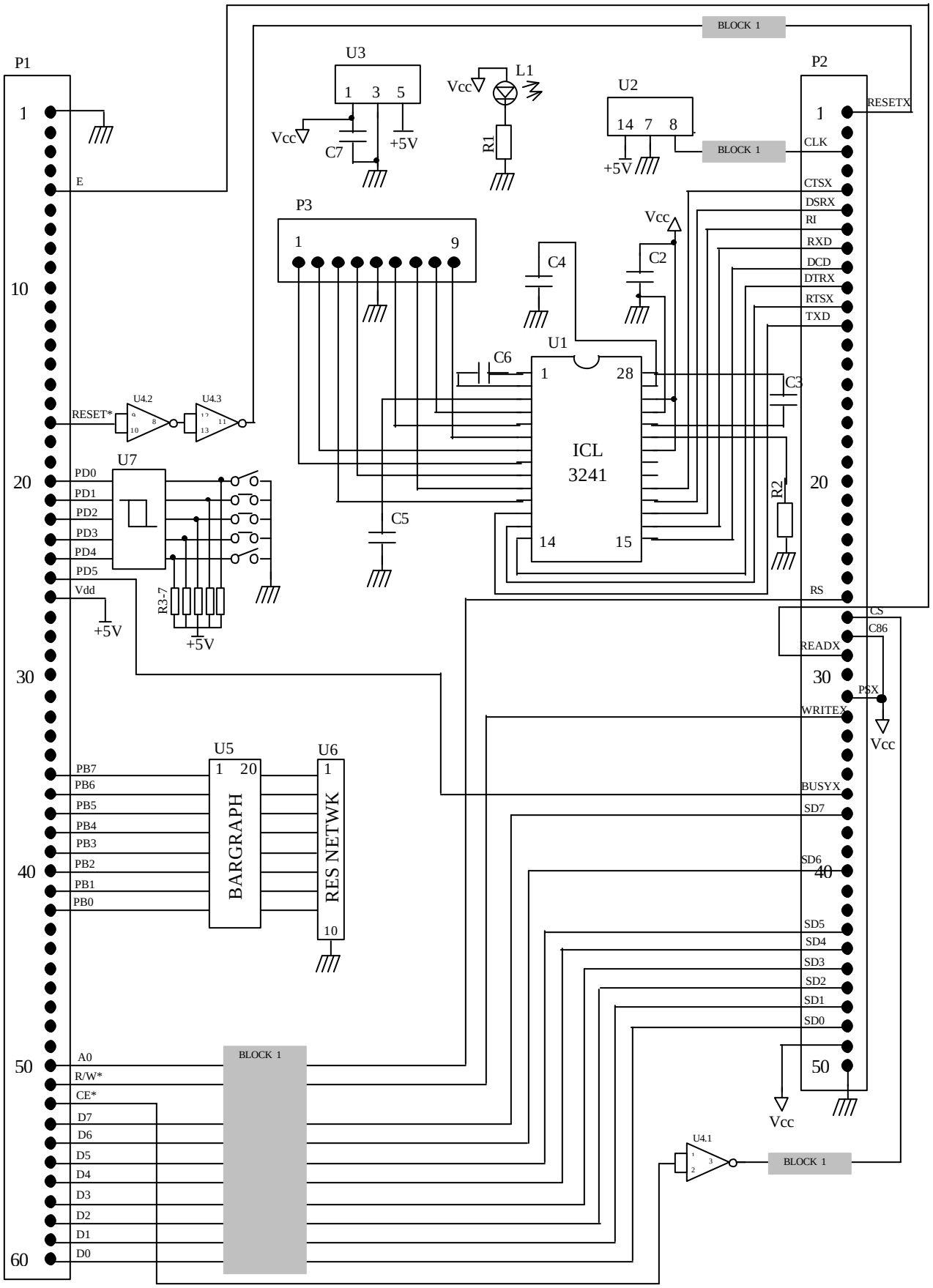


Figure 11: Schematic of the interface board

5.4.3 Part list

The items listed in table 11 were used to make the interface board. Prices come from Digi-Key catalog. All items without a Digi-Key part number were not ordered because they were already available.

Part No	Value	description	Digi-key Part No	Qty	rice eac	Total price
		Prototype board, 4.5x6.5 inches		1	$21.81	$21.81
		Socket connector to socket connector, 50 contacts, 18 inches, grey	A3AAG-5018G-ND	1	$9.38	$9.38
		Socket connector to socket connector, 60 contacts, 6 inches, grey	A3AAG-6006G-ND	1	$10.77	$10.77
P1'		Pin connector, 60 contacts	APK50G-ND	1	$11.97	$11.97
P2		Dual row straight header, 72 pins	S2012-36-ND	2	$1.85	$3.70
P3		Right angle, tin plated D-sub connector, 9 pins, male	A2096-ND	1	$2.72	$2.72
U1	ICL3241	3.3 V powered, 3 RS232 transmitters and 5 receivers, in DIP package		1	$4.00	$4.00
U2	PDI1051	oscillator 1MHz		1	$3.00	$3.00
U3	LT1129	3.3 V, 700 mA micropower low dropout voltage regulator	LT1129CT-33-ND	1	$4.00	$4.00
U4	SN74LS00N	Quad 2 input NAND gate, 14 DIP	296-1626-5-ND	1	$0.48	$0.48
U5		PC mount LED array, green, 2.1 V, 25 mA	67-1008-ND	1	$2.85	$2.85
U7		5 SCHMITT triggers inverters		1	$0.48	$0.48
L1		LED, 3 mm, green, 25 mA, 2.1 V	67-1056-ND	1	$0.18	$0.18
SW1,5		switch		2	$0.50	$1.00
SW2-4		push button, normally opened		3	$0.50	$1.50
C1-8	0.1 uF	Capacitor, 35 V	P2053-ND	8	$0.22	$1.73
C7	3.3 uF	Capacitor, solid tantalum, 10V DC	P2023-ND	1	$0.29	$0.29
R1	150	Resistor 1/4 W	150QBK-ND	1	$0.06	$0.06
R2-7	100k	Resistor 1/4 W	100KQBK-ND	6	$0.06	$0.34
R8-20	470	Resistor 1/4 W	470QBK-ND	13	$0.06	$0.73
R21-33	680	Resistor 1/4 W	680QBK-ND	13	$0.06	$0.73
U6	150	Resistance network, bussed, 150 Ohms, 4600 series, 9 res	4610X-1-151-ND	1	$0.39	$0.39
			TOTAL BOARD:			**$82.10**

Table 11: Bill of materials for the interface board and costing

The cost of components is about $82.00. But this is not the cost of the whole design: the SDK board was ordered for $199, together with a US Robotics 56k external modem, for $110. Hence, the overall cost of the embedded system is about **$400**.

5.5 Drivers for the iChip and the I/O

Three constants were declared in a file called *communication.h*, for easy access to the registers INDEX and DATA, and for access to the BUSYX signal:

```
#define INDEX (* (unsigned char *) (0x6000))     /* INDEX register of the S7600A */
#define DATA (* (unsigned char *) (0x6001))      /* DATA register of the S7600A */
#define BUSYX (PORTD & 0x20)                     /* BUSYX signal from S7600A*/
```

These registers are used by two C functions to read and write data in the iChip:

```
void x_write( BYTE addr, BYTE data )
```

1 byte data write driver

addr: select register of iChip
data: data to write in the iChip register

```
BYTE x_read( BYTE addr )
```

1 byte data read driver

addr: select register of iChip
returns value of the register

From an external point of view, only two registers are accessible in the iChip (INDEX and DATA). In fact, many more registers are indirectly accessible. To read a particular register, the first step is to write the index of this register in the INDEX register. Then, the data can be read/written in the DATA register. These functions are defined in a file called *communication.h*. A listing of these two functions is available in Appendix H. A header file containing all register declarations was written. It is called *s7600_reg.h*. It also contains constants that allow individual bit access of a few registers.
The header *file IO_s2b.h* contains constant declarations, which are very convenient to set, reset a LED, or monitor the state of an input. They are all listed below:

/*hardware descriptions of the outputs*/

```
#define S_INFLATE     (PORTB |= 0x80)     /*start inflation of speed bump */
#define S_DEFLATE     (PORTB |= 0x40)     /*start deflation of speed bump */
#define S_FLASH       (PORTB |= 0x20)     /*start flashing lights */
#define S_COMMU_LED (PORTB |= 0x10)  /*system in communication state */
#define S_MAINT_LED (PORTB |= 0x08)    /*system in maintenance state */
#define S_TCP_LED     (PORTB |= 0x04)     /*TCP established */
#define S_MAIL_LED    (PORTB |= 0x02)     /*sending email in progress */
#define S_ERROR       (PORTB |= 0x01)     /*error */

#define R_INFLATE     (PORTB &= 0x7F)   /*stop inflation of speed bump */
#define R_DEFLATE     (PORTB &= 0xBF)   /*stop deflation of speed bump */
#define R_FLASH       (PORTB &= 0xDF)   /*switch off flashing lights */
#define R_COMMU_LED (PORTB &= 0xEF)  /*system in communication state */
#define R_MAINT_LED (PORTB &= 0xF7)   /*system in maintenance state */
#define R_TCP_LED     (PORTB &= 0xFB)   /*not in ppp mode*/
#define R_MAIL_LED    (PORTB &= 0xFD)   /*no email being sent */
#define R_ERROR       (PORTB &= 0xFE)   /*no error */
```

/*hardware descriptions of the inputs*/

```
#define BTN_MAINT     (PORTD & 0x01)     /*maintenance state */
#define EM_REQUEST  (PORTD & 0x02)   /*simulate emergency vehicle request for deflation
*/
#define BTN_INF       (PORTD & 0x04)     /*force inflation of speed bump */
#define BTN_DEF       (PORTD & 0x08)     /*force deflation of speed bump */
#define HI_PRESSURE (PORTD & 0x10)    /* signal from pressure gauge 1=high pressure*/
#define LO_PRESSURE (PORTD & 0x10)   /* signal from pressure gauge 1=low pressure*/
```

These constants access port bits individually by means of masks.
In a C program, the following code is enough to set bit 4 of port B and hence, turn on the INFLATE LED. And it does so without changing the state of the other bits.

```
S_INFLATE;
```

The following code is used to turn off the INFLATE LED:

```
R_INFLATE;
```

The code for the inputs works in the same way. Here is an example of how the constant BTN_INF can be used:

```
if(BTN_INF != 0) return 1;
```

6 Programming the 68HC11

6.1 Choice of a C cross compiler

So far at URI, the programming of 68HC11 has always been done in assembly language. But that is not appropriate with the complexity of the S^2B design. It would be much faster and convenient to develop in C programming.

After a research over the Internet [10], 5 major software companies were contacted. They all offered interesting C cross compiler development tools, which main features are listed in appendix I IAR [11] proposed the most appropriate package. After a meeting with IAR sales manager, seven free licenses of IAR embedded workbench were given to URI. It is an ANSI C cross compiler for 68HC11, featuring a linker, a librarian, a debugger and a simulator. Hence, it will be used for the S^2B project, to program the 68HC11.

IAR embedded workbench software was received together with a set of 4 books and a dongle. The dongle is an electronic lock plugged in LPT1. It is required to run the software. IAR embedded workbench was successfully installed on Dr. Ohley's computer.

A few projects were created to check the working of the whole compiling/linking/ uploading/running procedure. They are available at the following location:
c:/my documents/ IAR C programs

6.2 Introduction to IAR embedded workbench

6.2.1 Start a new project

Run *IAR embedded Workbench* from the start menu of Windows.

The files in the Embedded Workbench are organized into projects. The first step is therefore to create a new project to specify which target processor is used, and to include a list of the files contained in the project.

Choose *New…* from the File menu. Select *project* and choose *OK*. Enter the name of the project, e.g. s2b, in the *Project filename box*. Set the *Target CPU family* to 68HC11. Choose *OK* to create the new project. A directory structure is created and displayed, with separated directories for object files, list files and executables.

The second step is to create a file and add it to the project.

Choose *New…* from the File menu. Select *source/text* and choose *OK*. Enter the code of the program. It is also possible to open an existing file with the command *Open…* from the *File* menu. If the file is saved with a .c or .h extension, Embedded Workbench provides immediate syntax checking, by using colored text. Next, choose *Project/Files* and select the file to add to the project. Click *Add*, then *Done*. The file that was just added should be displayed in the directory tree.

The third step is to compile and link the project. Many options can be set, for the whole target, for a group of files, or for a single source file. In our project, options are set for the entire Debug target by selecting the *Debug* folder icon in the Project window. Then, choose *Options…* from the *Project* menu. In *general* category, select *large* memory model. In xlink category, click *Output* to display output options. Here, two different settings are possible, depending on whether the output file will be used with the simulator (C-Spy), or uploaded in the EVB.

Set *Format* to *Debug info with terminal I/O* in the first case. Set it to *Other* in the second case, and set *output format* to *Motorola*.

Still in xlink category, select *Include* and check *override default* box, in *xcl filename* group. To choose or create the appropriate xcl file, refer to the next chapter.

To compile a file, choose *Compile* from the *Project* menu. If errors occur, edit the file and correct the errors. Else, the project can be built. To build the project, choose *Build* from the *Project* menu. Then, an output file is generated. It is located in the Debug/Exe folder. Depending on the link options that were selected earlier, the output file can be either used for simulation with C-Spy, or uploaded in the EVB, via a terminal emulator such as Hyper Terminal (c.f. chapter 3.2.1).

6.2.2 The lnk6811.xcl file

This file describes the address map of the target. The different memory locations are specified to suit the EVB memory map (c.f. appendix C). Follow the contents of the file lnk6811.xcl created to suit the EVB (available at c:/my documents/ IAR C programs).

```
-c68hc11
-Z(CODE)INTVEC=00C0-00FF
-Z(CODE)RCODE,CODE,CDATA0,CDATA1,CONST,CSTR,CCSTR=C010-CFFF
-Z(DATA)IDATA1,UDATA1,ECSTR,TEMP,CSTACK+200=D000-DFFF
-Z(DATA)IDATA0,UDATA0=0000-0035
cl6811
```

The first line defines what processor is used. The second defines the location of the interrupt vector table. The third line tells the linker where to locate program code and constants. The forth line does the same for variables and the stack. The fifth line defines the RAM segment used for direct addressing. The sixth line tells the linker what C library must be loaded.

The same file should be used by all the projects involving the EVB, so that improvements are automatically carried through each project.

6.2.3 The CSTARTUP.S07 file

On processor reset, execution passes to a run-time system routine called CSTARTUP, which normally performs the following:

- Initializes the stack pointer
- Initializes C file-level and static variables
- Calls the user program function main().

CSTARTUP is also responsible for receiving and retaining control if the user program exits, whether through *exit()* or *abort()*. A default CSTARTUP.s07 is provided at the following location: *c:/program files/iar/ew23/6811/src/lib*. If this file has to be modified, it must be reassembled. The whole procedure is explained in the 68HC11 C compiler programming guide [12].

6.3 Programming the S^{2}B as a state machine

6.3.1 Global architecture of the IAR Workbench project

A project called ***s2b v1.0*** was created with IAR Workbench. In this first version, the algorithm of the state machine is very simple, and many features are not developed yet. However, the project was structured to be easily upgradeable and modular.

The operation of the system was divided in 5 states:

- Main: wait for events to occur
- Operation: operate the speed bump in case of emergency request
- Maintenance: manually command the speed bump
- Communication: send email reports to a mail box
- Error handling: decide what to do in case of error or failure of the S^2B

The system is a state machine that will switch from one state to another, depending on the events occurring. The transitions between states are as described in figure 12. Upper case transitions correspond to physical events described in chapter 5.5.

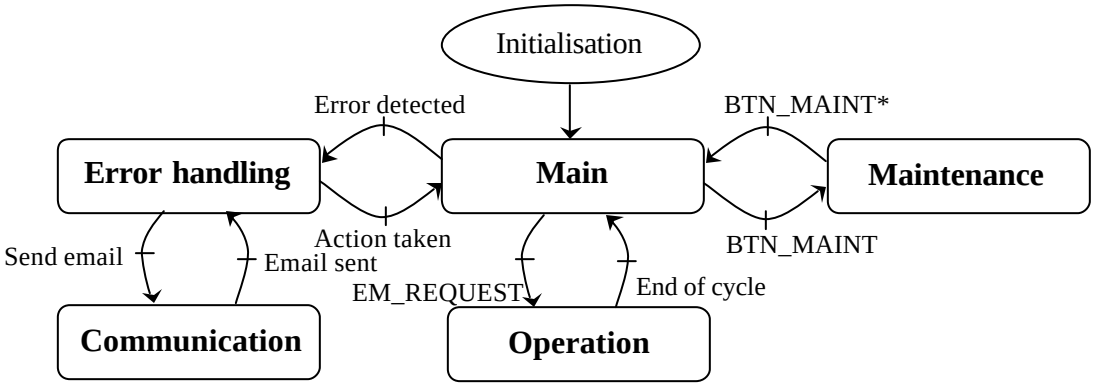


Figure 12: State diagram of the S^2B.

The architecture of the project matches the states of the state machine: the programming of each state is written in a different file, except *Error handling*, which is with the *main* state. By doing so, the programming of one state can be updated without altering the others. It is also convenient to find functions very quickly. Each C file is associated to a header file with the same name, where the functions are declared, as well as constants and global variables. The names of the files are:

- main.c and main.h
- operation.c and operation.h
- maintenance.c and maintenance.h
- communication.c and communication.h

A listing of all the files is provided in appendix H.
A double click on *s2b v1.0.prj*, in the explorer, opens the whole project in IAR Workbench.

6.3.2 The *main.c* file

This file contains 3 functions that are explained in table 12:

Function declaration	Description
void main(void)	Monitor inputs of the S^2B by polling port D. Switch to other states if events occur.
void init_system(void)	Initialize registers of the 68HC11and iChip. Inflate the speed bump.
void problem(int problem_id)	Perform an action, depending on the kind of error or failure that was encountered. Problem_id: identifies the kind of problem

Table 12: Declaration of the functions in main.h

The function *main()* calls the external functions *int maintenance()* and *int operation()*.
For further information about the working of the different functions, see appendix H. The listings of code are well documented.

6.3.3 The *operation.c* file

This file contains a single function called *int operation(void)*. The latter controls the speed bump in

case of emergency request. It returns an error message if a problem occurred.
This function is called by the main() function. The algorithm is as shown in figure 13:

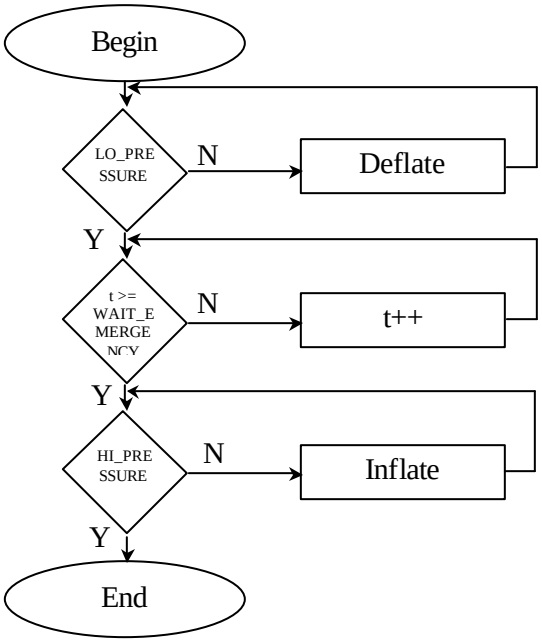


Figure 13: Operation of the S^{2}B, in case of emergency request

The constant WAIT_EMERGENCY is defined in operation.h. It corresponds to the time while the speed bump is deflated. It must be long enough for the emergency vehicle to run over the speed bump and go. At the end of the operation, the system returns in the main state.

6.3.4 The *maintenance.c* file

This file contains a single function called int maintenance(void). This function monitors push buttons used for maintenance. The buttons allow an operator to manually inflate of deflate the speed bump. The function returns an error message if a problem occurred. The algorithm of the function is as shown in figure 14:

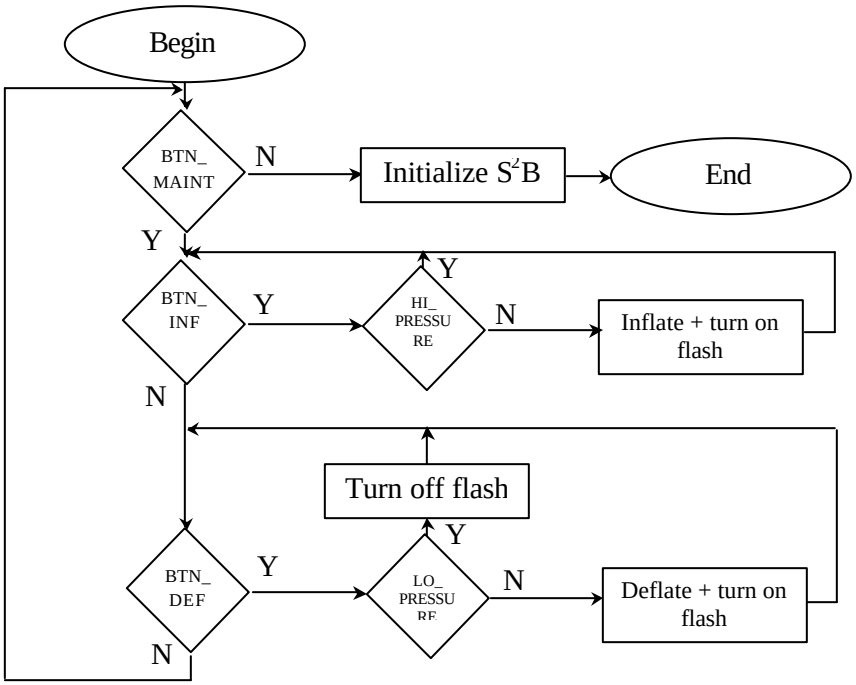


Figure 14: Algorithm of the maintenance mode

The *main()* function calls *maintenance()* if BTN_MAINT is switched in maintenance position. When BTN_MAINT is switched back to normal operation mode, *maintenance()* exits and the program goes back in *main()*.

6.3.5 The *communication.c* file

This file contains 15 functions that allow the S^2B to send email reports in case of problem with the S^2B. The message sent depends on the kind of problem that was encountered. Most functions are almost the same as the ones explained in chapter 5.4.2, apart from *x_read()*, *x_write* and *timer()*. For all the other functions, only minor changes were made. For instance, all the *printf* lines were removed. The function *int communication(int type_of_email)* replaces the old *main()* function. Its algorithm is illustrated in figure 15, on next page.

This function is called when the system is in error handling state, for some kind of problems only. Once an email is sent, the system returns in normal operation mode, i.e. in the *main()* function. If an error occurs during the communication state, the system returns in the error handling state, and the new problem is handled. Actually, the program is structured in such a way, but no error checking is implemented yet. So the program will never return error messages. In case of problem, the program will ignore it, or stay blocked in a loop. The error checking will be implemented in the next version of the program.

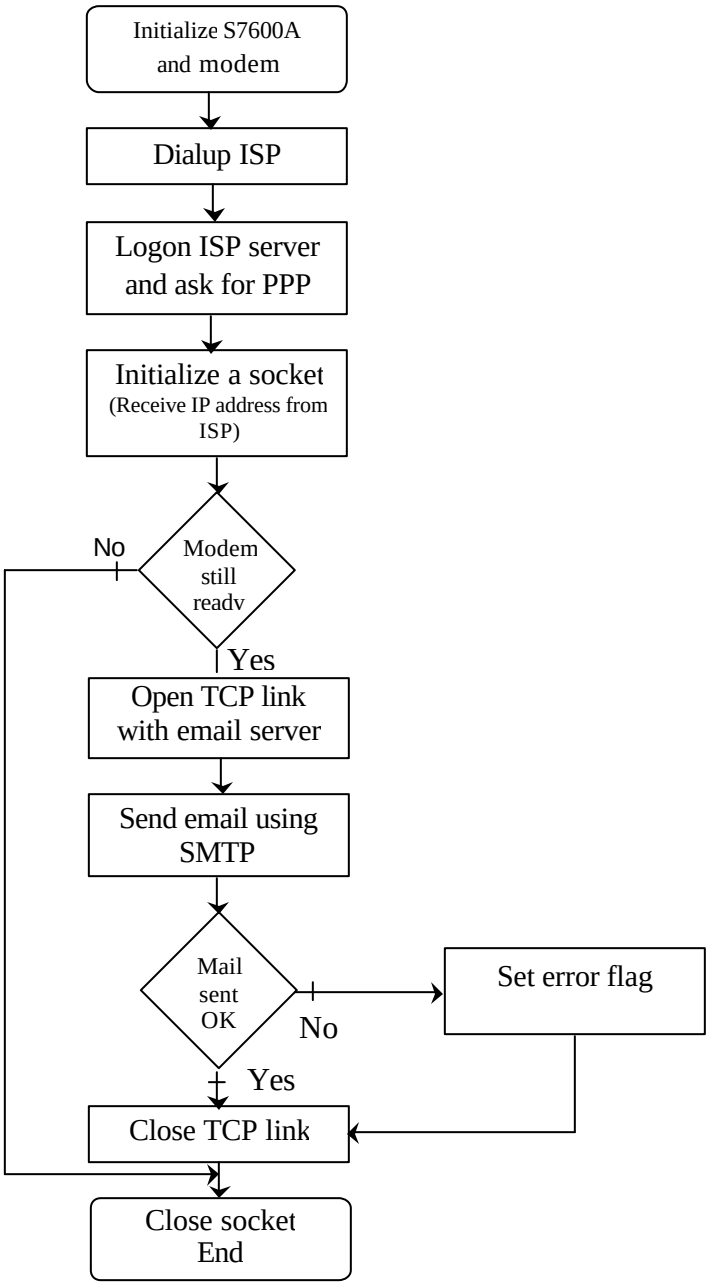


Figure 15: Algorithm to send an email, in communication state

The whole project, once compiled and linked, requires about 3kbytes of ROM, and 700 bytes of RAM. This can be checked in the *list* folder of the project, in the file called *s2b v1.0.lst*. The RAM requirements may vary, depending on the stack size that is chosen in lnk6811.xcl. The stack must be large enough to store at least one email message. Email messages are stored in the stack when the function *SendMail()* calls other functions.

7 Suggestions for further design and improvements

Suggestions for improvements of the overall design will not be developed here, since nothing was implemented. However, a first version of the embedded system was developed, with simulation of inputs and outputs of a S^2B in its real environment. This first version has many hardware and software imperfections. Some can be easily solved, while others would need more changes in the actual structure of the design. Follows a non exhaustive list of possible improvements for the next version of the S^2B embedded system:

Hardware improvements

1. The switches used for maintenance need an anti-bounce system. Indeed, they tend to bounce a lot. This is not a serious problem as long as the output pumps are simulated by LEDs. But it is likely to damage the pumps when they will be set up. Furthermore, the Schmitt triggers can be removed because they are useless.
2. Switches 1 and 2 are connected to bits 0 and 1 of port D. But these pins have another function: they can be used for RS232 link with a terminal. Hence, a serial link cannot be implemented in the actual configuration. And using a terminal would be very useful during the development of the program. Connecting the two switches to another input port would be enough to solve this problem.
3. The program of the S^2B is uploaded in RAM, on the EVB. Each time there is a power failure, or when the system is first set up, the program needs to be reloaded in memory. It would be interesting to replace this RAM by a flash memory. ROM would not do it because both program and variables are stored at RAM location.
4. A PCB could be designed for the embedded system. It would feature a 68HC11, memory, the iChip, a modem, a power interface for outputs and the signal shaping for inputs.
5. The power interface and the signal shaping were not developed because the characteristics of the inputs and outputs were unknown when this report was written. This needs to be done.

Software improvements

1. The main improvement is to add error checking everywhere in the program. In case of error or problem, the program should return an error message that will be computed in the error handling state. Furthermore, a software watchdog that would reset the system must be implemented, in order to avoid the program to stay blocked in a loop.
2. The timer function performs a software timer. A hardware timer, working with real time interrupts (RTI), would be much more accurate and reliable. It would also free the MCU so that it could do other tasks during this time.
3. The EM_REQUEST signal, which triggers the deflation of the speed bump, should be connected to an interrupt pin. It would allow the speed bump to go in the operation state, although it is in the communication state. In this case, the main program must be changed. An anti-bounce system must be added if a switch is still used to simulate an emergency request.

4. A daytime function could be developed. It would allow the speed bump to switch from one state to another, depending on the time. For example, the S^2B could deflate automatically by night, when the traffic is low. Also, it would be useful to report the time when a problem has occurred.
5. Statistics can be calculated from the pressure gauge. The speed bump could count the number of cars per day, their average speed, etc… and send a daily report to the email server. This can be done with an interrupt function that would be called each time there is an overpressure detected in the speed bump (count the number of vehicles running over the speed bump).
6. The Internet link is unidirectional. It would be great to add at state to the system where the embedded system is in server mode. In this configuration, the S^2B could be remote operated, updated, initialized…

CONCLUSIONS

The controller for an intelligent speed bump, which slows down normal vehicles but not emergency vehicles, has been implemented. This controller will be connected to a scale-model used for demonstration purposes.

First, the specification of a smart speed bump was developed. The hardware and software needs were identified. Then, the entire environment was set up to carry out the *Smart Speed Bump* project. The required materials were installed in Dr. Ohley's laboratory. A computer was connected to Internet and a phone line was set up. The diverse software used for this project was installed and configured. The 68HC11 evaluation board was installed together with a terminal. The Seiko Development Kit board and a modem were also installed and used to understand the working of the iChip. The latter allows easy Internet connection of any embedded system. Most of the work done during this internship concentrated on the design of the Internet connection. It works well and the design is efficient.

A hardware board was then designed to input and output signals to the 68HC11 evaluation board. This board was also required to interconnect the 68HC11 evaluation board and the second iChip of the Seiko Development Kit.

Finally, the software architecture was developed to perform the different operations of the Smart Speed Bump. This architecture is modular and is easily upgradeable. Additional features still need to be developed because many aspects of the S^2B are not implemented yet.

The intraaortic balloon cardiac assist system that we were supposed to use to simulate the speed bump was not available before the end of the research project, so we did not use it at all. We found it a little bit frustrating to design a system that only turns on and off LEDs.

REFERENCES

[1] Dr. William Ohley, professor of electrical engineering, URI, ohley@ele.uri.edu

[2] Frederic Bahuaud, electrical engineering student, ESPEO, frederic.bahuaud@fnac.net

[3] Motorola, M68HC11EVB evaluation board user's manual, revision 1, USA, 1986, 263p.

[4] Steven E. Newton , Introduction to the Internet Protocols, http://oac3.hsc.uth.tmc.edu/staff/snewton/tcp-tutorial/sec2.html,1994.

[5] Seiko Instruments Inc., Hardware specification s-7600A, revision 1.1, USA, Seiko, 1999, 55p.

[6] Mike Johnson, Mike's Seiko S7600 Webpage, http://www.mycal.net/wsweb/s7600/, 1999.

[7] Seiko, SDK board for ISA BUS specification, revision 0.2, USA, 1999, 13p.

[8] George Ho, iChip developer, Seiko, george.ho@seiko-la.com

[9] Motorola, M68HC11 reference manual, revision 1, USA, 1990, 350p.

[10] Compiler connection, Compiler connection search by company name, http://www.compilerconnection.com/companies/companies.htm, 1999.

[11] IAR, IAR homepage, http://www.iar.com/, 2000.

[12] IAR systems, 68HC11 C compiler programming guide, revision 8, USA, 1996, 278 p.

[13] J. Myers, Post Office Protocol - Version 3, http://www.cis.ohio-state.edu/htbin/rfc/rfc1939.html, 1996.

[14] Richard Beddingfield, Simple Mail Transfer protocol, http://www.provide.net/~bfield/polaris/topframe/top0120.htm , 1999.

[15] Seiko Instruments Inc., SDK board for ISA BUS specification, revision 0.2, USA, Seiko, 1999, 11p.

[16] Seiko Instruments Inc., Software development kit s-7600A, revision 1.1, USA, Seiko, 1999, 47p.

ACKNOWLEDGMENTS

Authors thank very much;

Mr. Raymond Sepe, Electro Standard Lab, 36 Western Industrial Drive, Cranston, RI 02921, 401-943-1164, not only for his generous monetary contribution but for his consultation expertise.

Dr. Rachid Harba, Professor of Electrical Engineering, University D'Orleans, France for his constant student intern support.

Mr. Matthew Caron for his help on the network and computer topics.

Ms. Phyllis Golden and **Ms. Alison Svenningsen**, for their administrative help, assistance and availability.

APPENDIX A

FOCUS GROUP FINAL REPORT

Dr. Marshall Feldman
Department of Community Planning and Landscape
University of Rhode Island
94 West Alumni Ave., Suite 1
Kingston, RI 02881-0815

UNIVERSITY OF RHODE ISLAND

SMART SPEED BUMPS:
FOCUS GROUP FINAL REPORT

Results from Focus Group Sessions
Designed to Solicit Input and to Provide
Community Outreach

Smart Speed Bumps:
Focus Group Final Report

Results from Focus Group Sessions
Designed to Solicit Input and to Provide
Community Outreach

Background

The original proposal included provisions for soliciting advice and opinions from decision-makers and citizens. This had three purposes: preliminary design, prototype refinement and implementation, and evaluation. Each purpose involved different groups and methods of data collection and analysis. They therefore comprised three distinct subtasks.

The preliminary design subtask solicited input from key decision makers in order to incorporate their ideas into the design and to address their concerns before the project went to far. The prototype refinement subtask sought input from ordinary citizens in order to refine the design and to develop an implementation plan. The final subtask involved evaluating the design through continuous feedback (through advertisements, etc.) and deliberate surveys of the public.

Unfortunately, funding limitations precluded us from implementing this plan as proposed. Instead, we conducted focus groups indirectly by having a class of URI undergraduates conduct focus groups in November 2000. This report summarizes that process and its results.

The Focus Group Process

The students did this as an assignment for a course, CPL 210: Introduction to Planning and Community. The course trained them in the rudiments of conducting focus groups, including assigned reading on the topic, design of the focus group protocol by the entire class under supervision of the instructor (Dr. Feldman), and a mock trial run of a focus group in class.

Three handouts were developed for this purpose (all three are included in the Appendix). The first, "Focus Groups," described the assignment and provided forms to assist in designing and conducting the focus group sessions. A second form, "Checklist for Focus Group Introductions," was used in the training to evaluate and to critique the students' performances as focus-group leaders. The third, "Focus Group Questions," was a list of questions and instructions developed by the class, under the instructor's guidance, to be used as a base for all the focus groups.

Students were instructed to conduct the focus groups either individually or in teams of two. Many of them elected to conduct the groups with family members over the Thanksgiving holiday. This resulted in an appropriately diverse sample of participants, although by no means was it a random sample representative of some larger population.

Results

Demographics

In all, the students conducted eleven focus groups, nine of which had six participants and two had five. Seven of the groups were conducted at URI and consisted primarily of college students; two had mainly "friends and relatives" and were therefore missed in terms of age and social status; one consisted entirely of "middle-aged adults" and one consisted of "professionals."

The students conducted seven focus groups at URI, two in Connecticut, and one in southern New Jersey-Pennsylvania area. The following table summarizes these demographics.

Size, Location, and Composition of Focus Groups

Group Size	Location	Composition
6	URI	College Students
6	CT	Middle-aged adults
6	URI	College Students
6	URI	College Students
6	URI	College Students
5		Friends and relatives
6	CT	Friends and relatives
5	URI	College Students
6	NJ/PA	Professionals
6	URI	College Students
6	URI	College Students

Findings

The findings from the focus groups fall into four categories: reactions, suggestions, issues, and needs for further research.

Reactions

Perhaps because the bulk of the focus group participants were college students, the idea of smart speed bumps-indeed, the idea of speed bumps altogether-was met with considerable animosity. Several participants said they saw no use for them, and many complained about speed bumps in general. A few questioned whether smart bumps provide much of an advantage over traditional speed bumps. Very few, if any, participants were unconditionally enthusiastic about the idea of smart speed bumps.

These findings should be taken with a grain of salt. In general, there was high correlation between the age and social status of the focus group and its animosity to the idea of smart speed bumps. Younger participants and college students were far more likely to criticize the idea and

to dismiss it than were the focus groups having a mixed composition or being made up of middle-aged adults or professionals. Given that the participants did not constitute a representative sample of the general population, these results may be highly biased. Furthermore, a major concern was the cost of smart speed bumps relative to their benefits. Since the focus groups did not have even ballpark cost estimate, they operated under considerable uncertainty regarding what turned out to be a major issue.

Suggestions

Two suggestions involved applications and, by implication, locations for using smart speed bumps. These included using smart speed bumps near schools and hospitals. In addition, some suggested that police use smart speed bumps in chases to delay or stop vehicles being pursued.

Other suggestions involved functional design. One was to have multiple heights so that the bumps would be "tunable." Another was to be sure to have lights and warning signs by the smart speed bumps so that drivers would be aware of them and whether they were up at a given instant.

Issues

Perhaps the most useful results from the focus groups were several issues and concerns that the groups identified. Even if theses are non-issues because of the design of smart speed bumps, they nonetheless constitute concerns among the general public that ought to be addressed, either through the design of the bumps or by better communicating design features. These issues can be grouped under several headings: effectiveness, functionality, administration, costs, and public receptiveness.

Effectiveness

Several groups questioned the effectiveness of speed bumps in general, noting that they often observe drivers slowing down to go over speed bumps and then speeding up again. While one might make the case that this lowers *average* speed, frequent slowing and accelerating might increase the rate of accidents. Moreover, the average speed will surely be much greater than the speed used to go over the speed bump, so it is almost certainly covered in the traffic engineering literature, but findings from this literature were unavailable to the focus groups.

A second effectiveness concern dealt with traffic congestion. The focus groups were afraid that widely used speed bumps would increase traffic congestion. Again, the traffic engineering literature indicates that this is not *necessarily* the case, but the groups did not have access to this literature. Moreover, any implementation of smart speed bumps would have to be engineered to obtain a level of traffic throughput appropriate for the application.

Functionality

Functionality was a major concern among the focus groups. Participants expressed concern about potential damage to vehicles and to animals possibly becoming trapped within the smart speed bumps' mechanism. This, and concerns about the effectiveness of speed bumps, led some participants to suggest that smart speed bumps not be used on regular city streets.

A second area of functional concern deals with reliability. Several participants expressed concern about possible malfunctions, electrical problems, and inclement weather. These issues must be addressed in the design.

Administration

Administration was a relatively minor concern, although several groups brought it up. This set of concerns deals mainly with who controls the smart speed bumps. In a positive vein, participants asked who would control the speed bumps and how such control might be coordinated across agencies. More negative concerns dealt with the possibility for deliberate interference with the operation of smart speed bumps. Such interference might come from vandals, criminals, or even terrorist intent on disrupting emergency services.

Costs

Costs were one of the most important and pervasive concerns among the groups. Without even ballpark cost estimates, many participants could not even decide if they were in favor of smart speed bumps or not. In addition to overall capital costs, participants raised questions about cost/benefit ratios and maintenance/operating costs. They also pointed to costs due to the disruption of traffic that would occur during installation and repair of smart speed bumps. Another related concern deals with the usable life of smart speed bumps, their reliability and the frequency of repairs.

Receptiveness

As mentioned above, several groups expressed hostility to the idea of speed bumps altogether. If this attitude is widespread, then there may be considerable opposition to smart speed bumps, particularly if the costs of the equipment, installation, and maintenance are high.

Needs for Further Research

These reactions and issues point directly to areas for further research. By far, the most important deals with costs and benefits. Further engineering research should focus on making the speed bump devices effective, efficient, and inexpensive. Further social research should aim at measuring the direct and indirect costs and benefits of smart speed bumps and comparing cost-benefit ratios to those of other, rival technologies.

A second area for further research concerns the design and implementation of smart speed bumps. For example, as mentioned above, participants questioned the reliability and safety of smart speed bumps in inclement weather, such as heavy snow. Further research should answer questions pertaining to such issues and address them.

APPENDIX B

FOCUS GROUP QUESTIONS

CHECKLIST FOR FOCUS GROUP INTRODUCTIONS

FOCUS GROUPS

FOCUS GROUP QUESTIONS

1. What do you think of speed bumps?
   (Now hand out a the description of Smart Speed Bumps)

2. What do you thinks of the Smart Speed Bumps?

3. What questions are crucial to deciding if they are a good idea?

4. How should they be used?

5. Where should they be used?

6. What suggestions do you have for improving the design?

7. What impact would Smart Speed Bumps have on your community?

CHECKLIST FOR FOCUS GROUP INTRODUCTIONS

Name:

_____

CHECKLIST FOR FOCUS GROUP INTRODUCTIONS			
	RATING		
CONTENT	**Change this**	**No change needed**	**COMMENTS**
Extends a welcome			
Introduces moderator and assistant			
Introduces study – why we're here			
Describes how participants were selected			
Welcomes all points of view			
Provides ground rules			
We're recording – confidentiality assured			
Invites questions			
Appropriate opening question			
Appropriate word choice			

DELIVERY	**Change this**	**No change needed**	**COMMENTS**
Relaxed and friendly			
Conveys sincerity and trust			
Smiles at some point			
Speed appropriate for participants			
Eye contact			
No distractions from content			

Suggestions for improvement

FOCUS GROUPS

Focus groups are a technique commonly used to involve citizens in community planning. They are a research method in which group discussions focus on a well-defined topic. Essentially focus groups are a way of listening to people and learning from them. In planning, focus groups are typically used to identify problems and needs, to find the best way to achieve a set of goals, to gain feedback on how a plan is being implemented, and to assess a plan. Conducting focus groups involves four steps:

1. Planning the focus group session, including most importantly what one needs to hear from participants.

2. Recruiting participants

3. Moderating the focus group as a conversation among people.

4. Analyzing and reporting what is learned from the focus group.

The Topic

For the purpose of this exercise we will conduct focus groups around a new innovation related to transportation planning: "smart speed bumps." Smart speed bumps are a computer-controlled traffic-calming device. Unlike conventional speed bumps, smart speed bumps can be raised or lowered. Typical applications might include lowering speed bumps to allow emergency vehicles to reach a destination more quickly, selective traffic calming at different times of the day depending on traffic patterns, interactive traffic calming reflecting the presence or absence of school children, or interactive traffic calming in response to vehicular speeds.

Assignment

Conduct a focus group addressing the viability of smart speed bumps. Ideally, teams of 2-3 people would conduct the focus groups. However, students often find it difficult to coordinate schedules, so you may do this assignment alone or in teams of two. If you do this in teams of two, I will expect higher quality.

Conduct your focus group and submit a report as described under "**Report,**" below. Your focus group should have at least six participants. Since we have no budget, you may have to compromise with the ideal group composition. *If so, be sure to discuss these compromises in your report.* Tape record your focus group session on an ordinary tape cassette and turn in your tape.

Due Date

In class, November 30.

Report

Submit a report on your focus group and how you conducted it. The report should have the following components:

Executive Summary: One paragraph summarizing what you did and what you discovered.

Table of Contents: List the parts of your report and the corresponding page numbers.

Part I: Planning

Problem Statement**:** One paragraph summarizing the purpose of the focus group

Description of the Target Population: One paragraph describing who should participate in the focus groups and why this group, rather than some other, should participate.

Strategy for Identifying Target Population: This should consist of two paragraphs. The first should explain how you would *ideally* find people belonging to the target population. Include discussion of any screening criteria in this paragraph. The second paragraph should discuss your *actual* strategy for selecting participants and if there are any other selection problems associated with your strategy.

Part II: Recruitment

Recruitment Process: One paragraph summarizing how you *actually* got members of the target population to participate and who actually did or did not wind up participating. This is different from your actual selection strategy. The selection strategy describes your *plan* for recruiting participants, the "Recruitment Process" describes what *actually happened*.

Recruitment Bias: One paragraph discussing what, if any, recruiting bias is present in your sample. Explain why you believe bias is or is not present. Make references as needed to items in the Appendix.

Part III: Moderating

Questions: Discuss any issues that came up with the questions. You may have decided to add questions to the ones we developed in class. If so, explain why. The questions themselves should be in the Appendix, and this section can refer to the questions there.

Do not use this section to discuss all replies to the questions, just issues such as ambiguities, misunderstandings, and so on. Typically this section will be one or two paragraphs long.

Field Notes: Include your field notes in the Appendix.

Written "Oral" Summary: Include a one-paragraph oral summary of the focus group.

Part IV: Analysis and Report

Top-Line Report: Prepare a Top-Line report as described in the handout, "Written Reports."

APPENDIX

Telephone Screening Script: Again, this will probably be simulated in that I don't expect you to actually recruit participants by telephone. Nonetheless, draft a telephone screening script similar to those on pages 58-60 in the handout.

Draft Invitation: This is a simulated exercise in that I don't expect you to actually use this invitation. However, draft a brief invitation that you could use if you were to send out formal invitations to participants in the focus group.

Recruitment Log: Using the form below, keep a log of all your attempts to recruit people to participate in the focus group. The completed form should be included in the appendix.

Questions: This should be a list of questions you asked when you moderated the focus group. If you decided to add any questions to the list developed in class, include them here.

Self-critique: Using the criteria for moderating a focus group (pp. 69-70 of the handout), rate yourself as a moderator. Use the **Criteria for Rating Moderators** form below.

Field Notes: Include your field notes. Use the standardized reporting form template below.

Tape Recording: You should record your focus group on a regular cassette tape. Include the tape with your report.

RECRUITING LOG			
Your name:			
Name & Address	*Dates/Times Contacted*	*Status*	*Notes*

Criteria for Rating Moderators

Before the focus group

☐ *Is familiar with the topic and goals of the sponsor*

☐ *Understands the purpose and objective of each question*

☐ *Has a sense of the amount of time needed for each question*

☐ *Anticipates the topics of discussion and potential areas of probing*

☐ *Is mentally and physically ready to moderate*

☐ *Has sufficient technical knowledge of the topic*

☐ *Welcomes participants and makes them feel comfortable before the session*

During the focus group

☐ *Delivers a smooth, comfortable introduction that is accurate and complete, including:*

- *a welcome*
- *a brief overview of the topic that defines the purpose of the group*
- *a description of the ground rules*
- *the opening question*
-

☐ *Establishes rapport with participants*

☐ *Asks the questions as intended, unless they have already been answered in another question*

☐ *Allows sufficient time for each question*

☐ *Keeps the discussion on track*

☐ *Keeps all participants involved*

☐ *Listens carefully; synthesizes information and feeds it back, probes for clarification, gets people to talk*

☐ *Seeks out both cognitive and affective domains; gets participants to tell both what they think and what they feel about the topic*

☐ *Moves smoothly from one question to another*

☐ *Handles different participants adeptly and conveys a sense of relaxed informality*
☐ *Avoids sharing personal opinions*

☐ *Finishes on time*

☐ *Brings closure to the group with a summary and invites comments on any missing points*

☐ *Goes to the door and thanks each person individually for coming, just as you would when guests leave your home*

After the focus group
☐ *Debriefs soon after the focus group with the assistant moderator*

☐ *Performs the analysis or provides insight into the analysis*

☐ *Reviews the report for accuracy*

Standardized reporting form template

This is just a template. The part labelled “Information about the Focus Group” should be included on the report. The part labelled “Q1” should be included on the report and repeated for each question. Remember to change the question number and to include the question itself.

Information About the Focus Group

Date of Focus Group	
Location of Focus Group	
Number and Description of Participants	
Moderator Name	
Asst. Moderator Name	

Responses to Questions

Q1. What do you think of speed bumps?

Brief Summary and Key Points	*Notable Quotes*
Comments and Observations	

Focus Group Evaluation
(Total points possible = 100)

Student's Name: _____

Executive Summary (5 points) points _____
Does it capture the substance of the report?
Is it clear and direct?

Table of Contents (2 points) points _____

Problem Statement (5 points) points _____
Does it summarize the problem clearly and concisely?

Description of Target Population (10 points) points _____
Is there a clearly identified target group?
Is the reason for choosing this target population clear?
Is the target population logical in the sense that choosing *this* target population follows logically from the problem statement?
Does it conform to the assignment?

Strategy for Identifying Target Population (10 points) points _____
Is the description complete in the sense that the reader can reproduce the strategy from reading the description?
Are screening criteria appropriate? Are they discussed clearly?
Are appropriate selection problems discussed? Is the discussion clear? Does it relate the problems to the strategy?
How effective is the strategy likely to be?

Recruitment Process (5 points) points _____
Is the description clear? Could a reader replicate the process?
Do we know who actually did or did not participate? Do we know why they did so?

Recruitment Bias (5 points) points _____

Is the description of recruitment bias clear?

Does the discussion show an understanding of the concept of recruitment bias?
Is the explanation of why bias is/is not present logical?
Are appendix materials used appropriately?

Questions (5 points) points _____

Are additional questions or changes to questions warranted?
Are problems with questions described clearly?
Were appropriate remedial measures taken?

Field Notes (15 points) points _____
Are field notes complete?
Are they clear?
Are they useful?

Written "Oral" Summary (10 points) points _____
Does the author come to some conclusion regarding the research?
Do the materials presented support the conclusions?
Top-Line Report (15 points) points _____

Does the top-line report reflect the field notes and tape?
Does it take due account of any problems?

Telephone Screening Script (5 points) points _____

Is the script clear?
Is it effective?
Does it relate properly to the target group?

Draft Invitation (3 points) points _____

Is the invitation truly inviting?
Does it give a fair overview of the nature and purpose of the focus group?

Self-critique (5 points) points _____

Does it seem consistent with the findings and rest of the report?

APPENDIX C: M68HC11EVB EVALUATION BOARD ARCHITECTURE

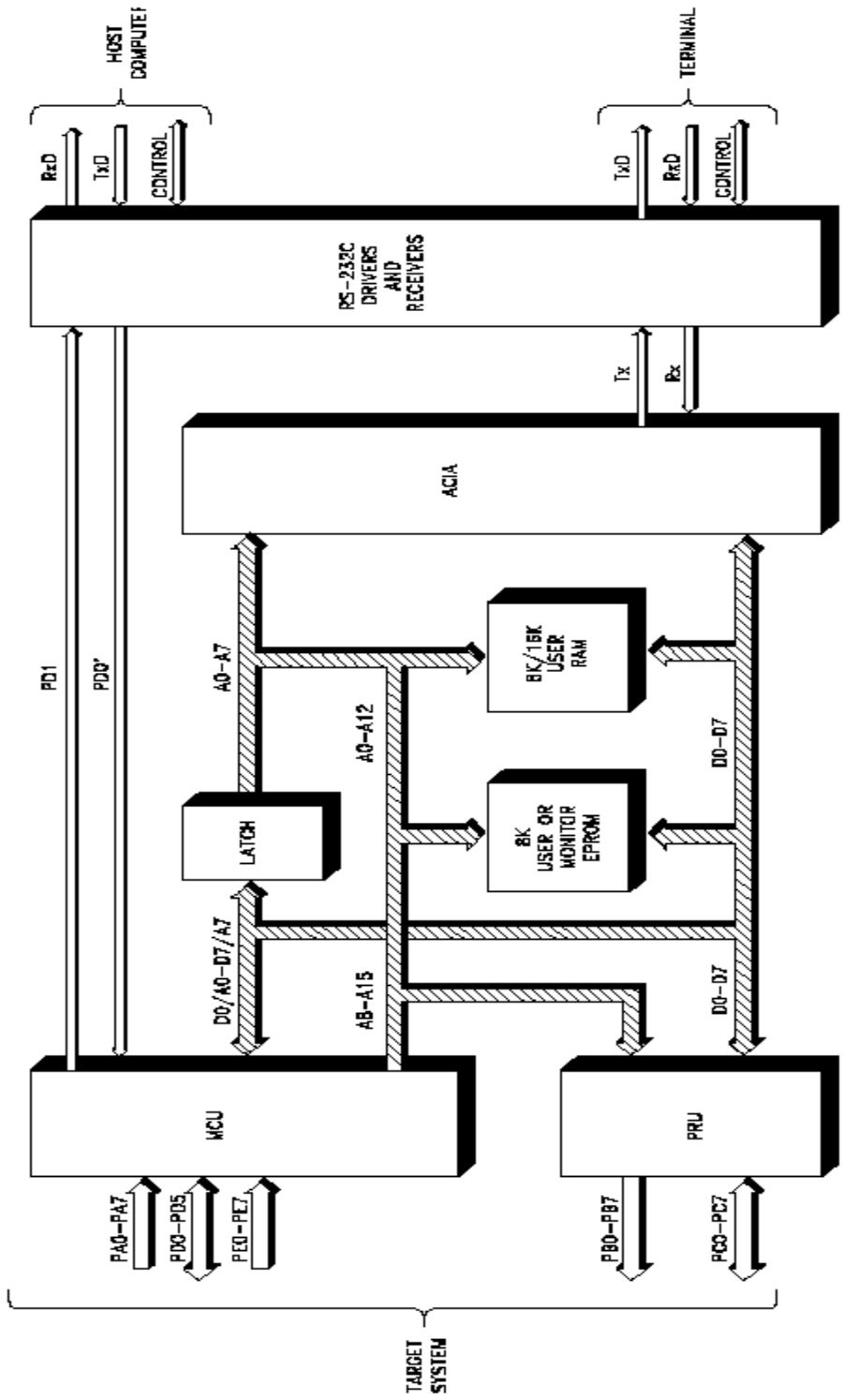


Figure 16: Architecture of the 68HC11 evaluation board (EVB)

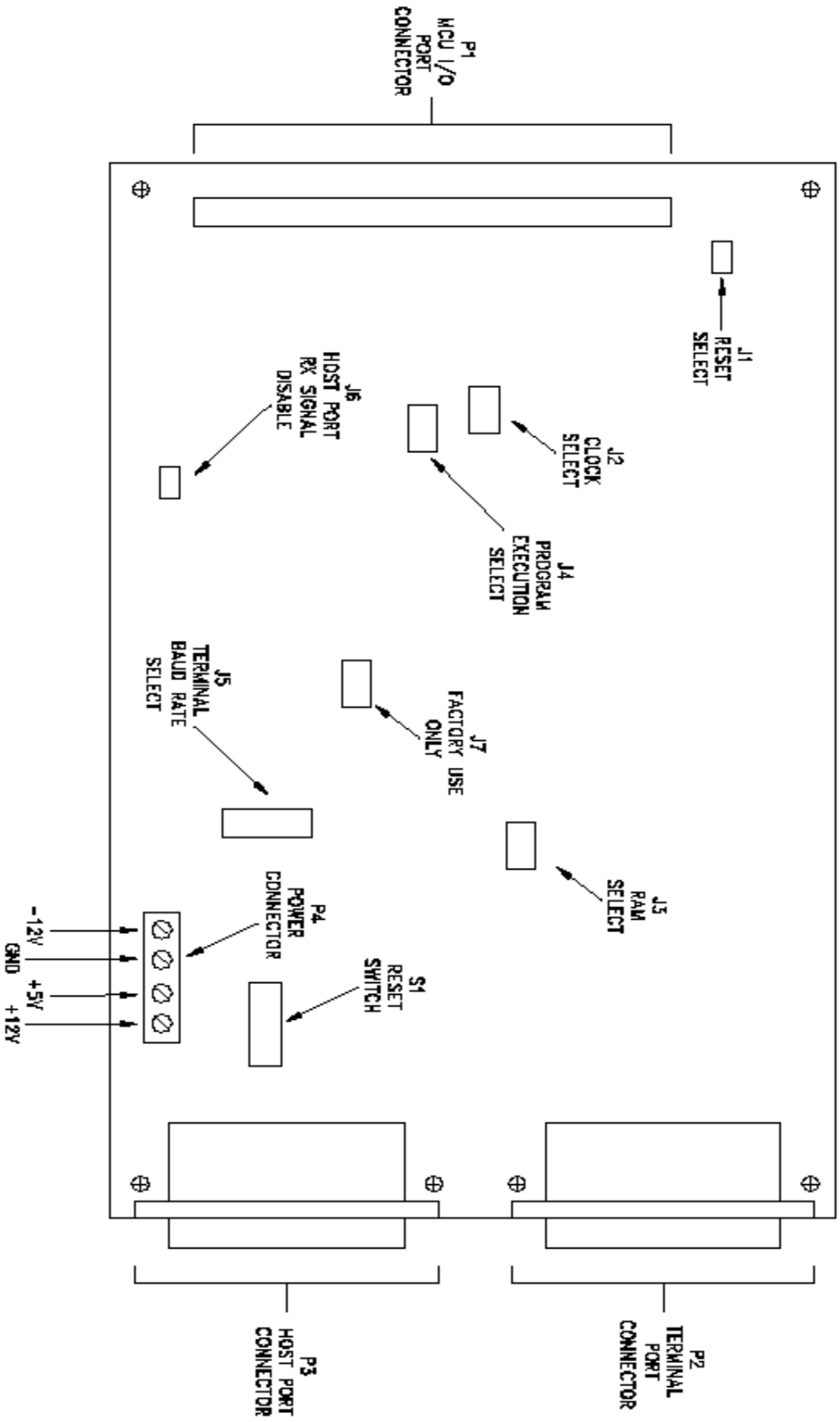


Figure 17: EVB connector location diagram

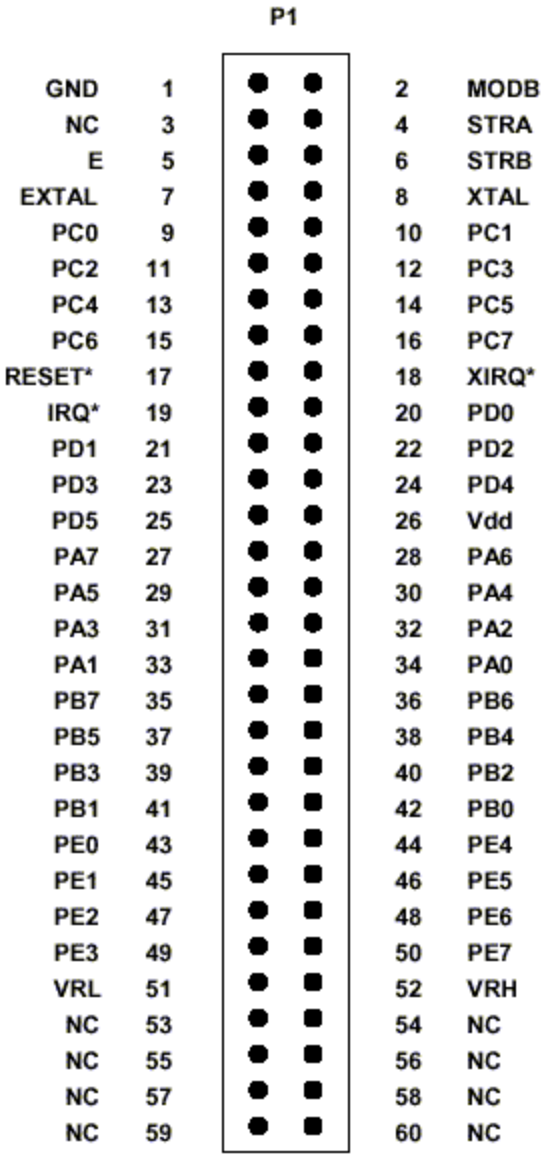
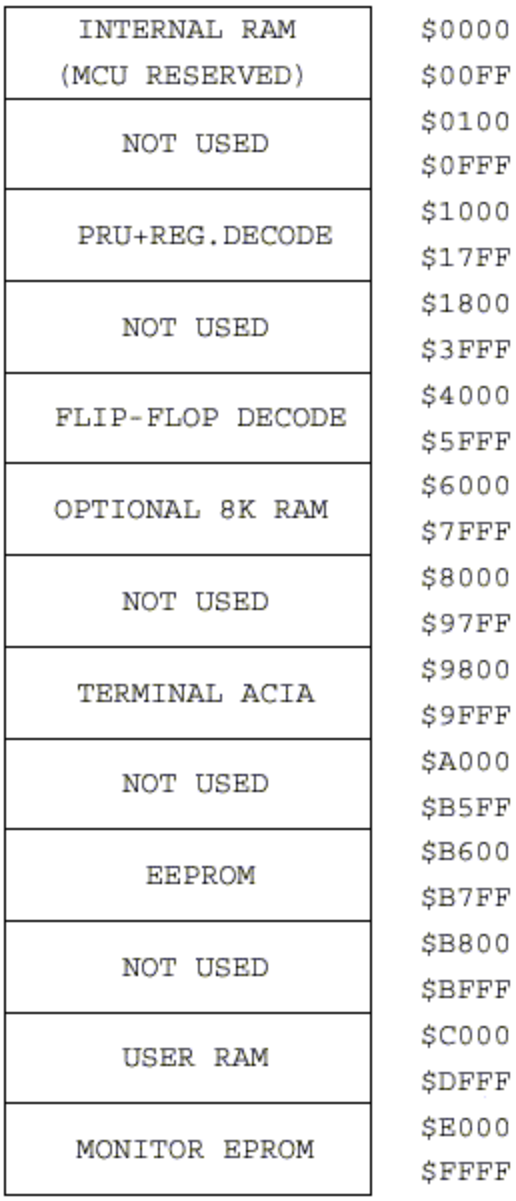


Figure 18: Pin assignment of connector P1



$0000-$0032 USER RAM
$0033-$0047 USER STACK POINTER
$0048-$00C3 MONITOR VARIABLES
$00C4-$00FF VECTOR JUMP TABLE

Figure 19: EVB memory map location

Interrupt Vector	Field
Serial Communications Interface (SCI)	$00C4 - $00C6
Serial Peripheral Interface (SPI)	$00C7 - $00C9
Pulse Accumulator Input Edge	$00CA - $00CC
Pulse Accumulator Overflow	$00CD - $00CF
Timer Overflow	$00D0 - $00D2
Timer Output Compare 5	$00D3 - $00D5
Timer Output Compare 4	$00D6 - $00D8
Timer Output Compare 3	$00DC - $00DE
Timer Output Compare 1	$00DF - $00E1
Timer Input Capture 3	$00E2 - $00E4
Timer Input Capture 2	$00E5 - $00E7
Timer Input Capture 1	$00E8 - $00EA
Real Time Interrupt	$00EB - $00ED
IRQ	$00EE - $00F0
XIRQ	$00F1 - $00F3
Software Interrupt (SWI)	$00F4 - $00F6
Illegal Opcode	$00F7 - $00F9
Computer Operating Properly (COP)	$00FA - $00FC
Clock Monitor	$00FD - $00FF

Table 13: Interrupt jump vector table

APPENDIX D: POP3 AND SMTP COMMANDS

POP3 Command Summary [13]

- USER <name> put your email username
- PASS <string> put your email password
- QUIT quit
- STAT tells number of messages inbox and total size
- LIST list message numbers and their size
- RETR <msg> read message
- DELE <msg> delete message
- NOOP no operation
- RSET

POP3 Replies:

- +OK command accepted
- -ERR command refused

SMTP commands summary [14]

The Internet protocol used in the transfer of e-mail is SMTP, or Simple Mail Transfer Protocol SMTP is usually accessed on port 25 of your Internet provider's SMTP server. If you manually telnet to port 25 of an SMTP server, an SMTP session is initialized. In such a session, the following commands can be used:

• HELO <hostname>	Introduce yourself.
• EHLO <hostname>	Introduce yourself and request extended SMTP mode.
• MAIL FROM: <sender>	Specifies the sender.
• RCPT TO: <recipient>	Specifies the recipient. Can be used any number of times.
• DATA	Following text is collected as the e-mail message. End message with a period on a line by itself.
• RSET	Resets the system. Once reset, a new sender can be specified.
• NOOP	Do nothing.
• QUIT	Exit sendmail (SMTP)
• HELP	Gives command info. If used alone, displays commands implemented on specific system. If used as HELP <command>, specific information on that command is given.
• VRFY <recipient>	Verify an address. To view aliases, use EXPN instead.
• EXPN <recipient>	Expands an address. Same as VRFY, but includes aliases and mailing lists.

APPENDIX E: S7600A iCHIP INFORMATION [5]

Name	I/O	Description	Type
VDD1,VDD2	-	Positive power supply	
VSS1,VSS2	-	GND potential	
RESETX	I	Reset input	A
TEST, TI1 to TI7	I	Test input (pull-down resistor is built in) When normal use, connect to V_{SS} or open	B
TO1 to TO7	O	Test output When normal use, connect to Vss or open	D
CLK	I	Clock input	C
CTSX	I	Clear to send input	C
DSRX	I	Data set ready input	C
RI	I	Ring indicator input	C
RXD	I	Serial received data input	C
DCD	I	Data carrier detect input	C
DTRX	O	Data terminal ready output	D
RTSX	O	Request to send output	D
TXD	O	Serial transmit data output	D
RS	I	Register selection input	C
CS	I	Chip selection input	C
C86	I	MPU interface mode selection input 68k mode : 1 x80 mode : 0	C
READX	I	x80 mode : read requirement input 68k mode : enable input	C
PSX	I	parallel/serial interface selection input	C
WRITEX	I	x80 mode : write requirement input 68k mode : read/write selection input	C
INTCTRL	I	INT1/INT2X drive type(CMOS/OD) selection input	C
INT1	*OT	Interrupt output(active High) from S-7600A chip to MPU	E
INT2X	*OT	Interrupt output(active Low) from S-7600A chip to MPU	E
BUSYX	O	busy indicator output	D
SD7	*B	x80/68k mode : data bus Serial mode : serial data input	F
SD6	*B	x80/68k mode : data bus Serial mode : serial clock input	F
SD5	*B	x80/68k mode : data bus Serial mode : serial data output	F
SD0 to SD4	*B	Data bus	F

*OT : Tri-state output
*B : bi-directional

Table 14: Pin description of the S7600A

R/W cycle for 68k family MPU mode

This mode can be selected by pulling the **C86** input pin “H” and the **PSX** input pin “H”. In this mode, the address and data are muxed into a single 8-bit bus. All cycles start by placing an address on the bus and setting the **RS** pin to “L”. In this mode **WRITEX** signal works as read/write(R/WX) signal and **READX** is the enable(E) signal for 68k Family MPU interface. After the address cycle, the MPU generates a read or writes strobe by setting the **READX** and **WRITEX** pins. The S-7600A MPU interface logic assert a **BUSYX** signal low during data write and read phases. The MPU samples the **BUSYX** signal before starting a new cycle. The CPU can initiate a new cycle if the bit is “H”.

Write Cycle Timing

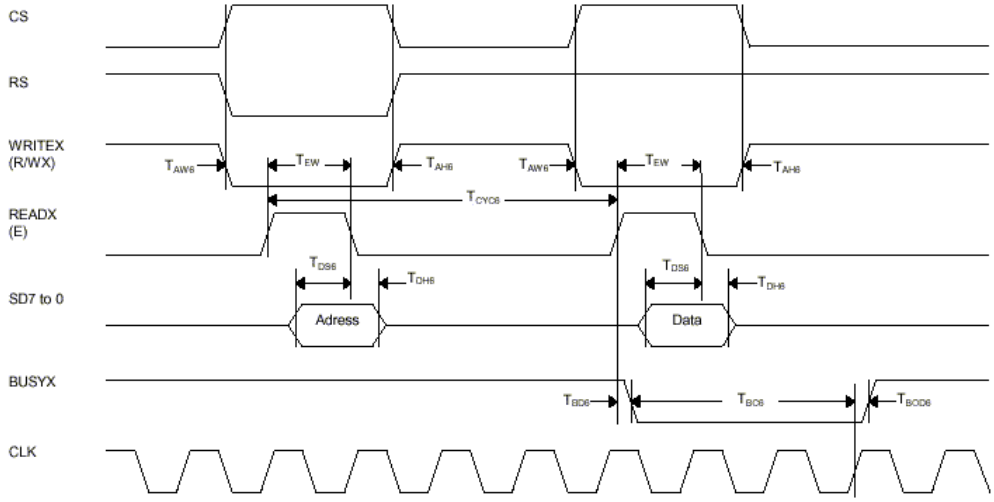


Figure 20: 68 family MPU write cycle timing

Read Cycle Timing

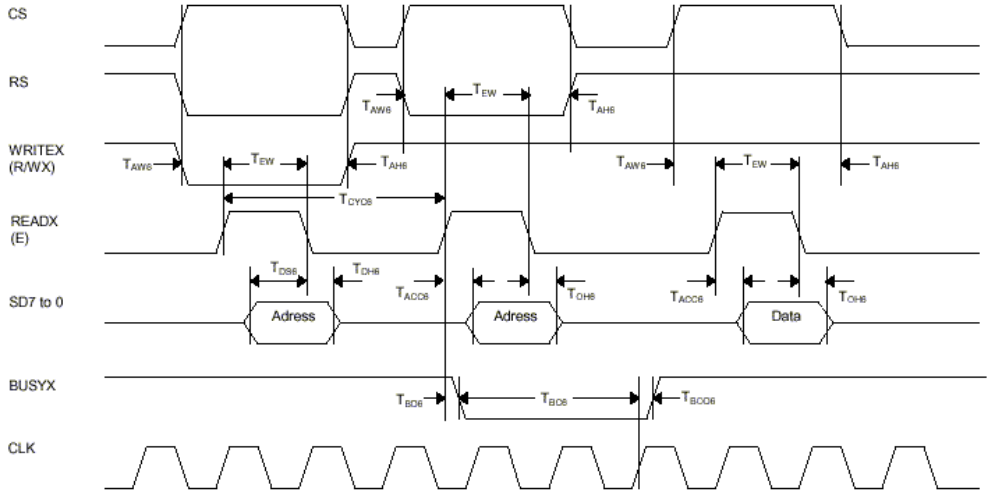


Figure 21: 68 family MPU read cycle timing

Add	Register	Bit Definitions							
0x00	Revision	Major Revision Number				Minor Revision Number			
0x01	General_Control	-	-	-	-	-	-	-	SW_ RST
0x02	General_Socket_ Location	0	0	0	0	0	0	S1	S0
0x04	Master_Interrupt	-	-	-	-	-	PT_ INT	LINK _INT	SOCK _INT
0x08	Serial_Port_Config	S_ DA V	DCD	DSR/ HWFC	CTS	RI	DTR	RTS	SCTL
0x09	Serial_Port_Int	PT_ INT	-	-	-	-	-	-	-
0x0A	Serial_Port_Int_ Mask	PINT _EN	DSINT_ EN	-	-	-	-	-	-
0x0B	Serial_Port_Data	Serial Data Register							
0x0C - 0x0D	BAUD_Rate_Div	BAUD Rate Divider Registers							
0x10 - 0x13	Our_IP_Address	Our IP Address							
0x1C	Clock_Div_Low	Low Byte for 1 kHz clock divider							
0x1D	Clock_Div_High	High Byte for 1 kHz clock divider							
0x20	Index	Socket index							
0x21	TOS*	Type of Service Field							
0x22	Socket_ Config_Status_Low*	TO	Buff_ Empty	Buff_ Full	Data_ Avail/ RST	-	Protocol_Type		
0x23	Socket_Status_Mid*	URG	RST	Term	ConU	TCP State			
0x24	Socekt_Activate	-	-	-	-	-	-	S1	S0
0x26	Socket_Interrupt	-	-	-	-	-	-	I1	I0
0x28	Socket_Data_Avail	-	-	-	-	-	-	DAV1	DAV0

NOTE: 1)Reserved bits are signified by a dash (-). All reserved bits should be written as "0".
2)Indexed registers are signified by an asterisk (*).

Table 15: Register map of the S7600A – Part 1

Add	Register	Bit Definitions							
0x2A	Socket_Interrupt_ Mask_Low*	TO_ En	Buff_ Emp_ En	Buff_F ull	Data_ Avail_ En	-	-	-	-
0x2B	Socket_Interrupt_ Mask_High*	URG_En	RST_ En	Term_ En	ConU En	-	-	-	-
0x2C	Socket_Interrupt_Low*	TO	Buff_ Empty	Buff_ Full	Data_ Avail	-	-	-	-
0x2D	Socket_Interrupt_High*	URG	RST	Term	ConU	-	-	-	-
0x2E	Socket_Data*	Socket 8-bit data							
0x30	TCP_Data_Send (WO)*	Any write causes data to be sent							
0x30 - 0x31	Buffer_Out (RO)*	Buffer Out Length							
0x32 - 0x33	Buffer_In (RO)*	Buffer In Length							
0x34 - 0x35	Urgent_Data_Pointer*	Urgent Data Offset Pointer, UDP Datagram Size							
0x36 - 0x37	Their_Port*	Target Port Address							
0x38 - 0x39	Our_Port*	Our Port Address							
0x3A	Socket_Status_High*	-	-	-	-	-	-	-	Snd _bsy
0x3C - 0x3F	Their_IP_Address*	Target IP Address							
0x60	PPP_Control_Status	PPP_Int	Con_ Val	Use_ PAP	To_ Dis	PPP_ Int_En	Kick	PPP_ En	PPP_ Up / SRset
0x61	PPP_Interrupt_Code	Interrupt Code							
0x62	PPP_Max_Retry	-				PPP Maximum retry			
0x64	PPP_String	Pap user name and password							

NOTE: 1)Reserved bits are signified by a dash (-). All reserved bits should be written as "0".
2)Indexed registers are signified by an asterisk (*).

Table 16: Register map of the S7600A – Part 2

APPENDIX F: S7600A SDK BOARD FOR ISA BUS [15]

The S-7600A SDK board interfaces to between the ISA bus and S-7600A with glue logic in a CPLD. The RS232C Driver/Receiver is used to connect the physical layer to the S-7600A. Figure 2-1 shows a functional block diagram of the S-7600A SDK board.

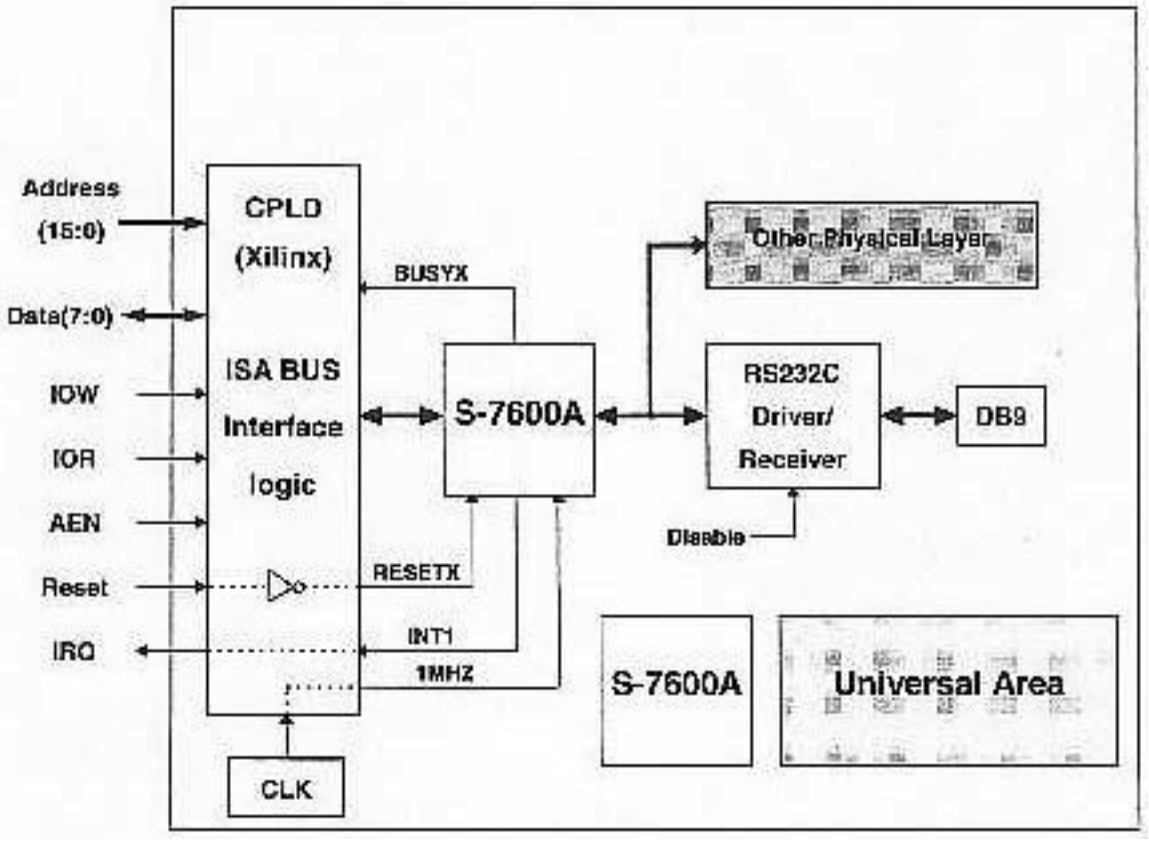


Figure 22: Block diagram of the Seiko Development Kit (SDK)

The S-7600A SDK Board is equipped with several connectors. The ISA bus interface connector (CN1) is provided to interface the board to a ISA bus for PC. The CN3 provided direct access to the second S-7600A signals for use user's appreciation. Also the CN5 can provided direct access to the UART of Main S-7600A for user's physical interface or for use to signal monitor when debugging.

Table 3-2 shows signal for CN1 (ISA bus).

Row A	Pin name	Row A	Pin name	Row B	Pin name	Row B	Pin name
1	-	17	A14	1	0V	17	-
2	D7	18	A13	2	RESETDRV	18	-
3	D6	19	A12	3	+5V	19	-
4	D5	20	A11	4	IRQ9	20	SYSCLK
5	D4	21	A10	5	-	21	IRQ7
6	D3	22	A9	6	-	22	IRQ6
7	D2	23	A8	7	-	23	IRQ5
8	D1	24	A7	8	-	24	IRQ4
9	D0	25	A6	9	-	25	IRQ3
10	IOCHRDY	26	A5	10	0V	26	-
11	AEN	27	A4	11	-	27	-
12	-	28	A3	12	-	28	-
13	-	29	A2	13	IOW	29	+5V
14	-	30	A1	14	IOR	30	-
15	-	31	A0	15	-	31	0V
16	A15			16	-		

Figure 23: Pin assignment of CN1

Pin No.	Pin name	Pin No.	Pin name	Pin No.	Pin name	Pin No.	Pin name
1	RESETX	14	-	27	CS	40	SD6
2	TEST	15	-	28	CS6	41	-
3	CLK	16	-	29	READX	42	VDD
4	VSS	17	-	30	VSS	43	SD5
5	CTSX	18	-	31	PSX	44	SD4
6	DSRX	19	-	32	WRITEX	45	SD3
7	RI	20	VDD	33	INTCTRL	46	SD2
8	RXD	21	-	34	INT1	47	SD1
9	DCD	22	-	35	INT2X	48	SD0
10	DTRX	23	-	36	BUSYX	49	VDD
11	RTSX	24	-	37	SD7	50	VSS
12	TXD	25	-	38	-		
13	-	26	RS	39	-		

Figure 24: Pin assignment of CN3

APPENDIX G: PROGRAM OF AN EMAIL REPORT ENGINE

File custom.h

```
#define WritePCPort _outp
#define ReadPCPort _inp

#define FALSE   0
#define TRUE    1

/* I/O Address of S-7600A SDK Board setting.*/
#define HW_PORT_INDEX   0x0430
#define HW_PORT_DATA    HW_PORT_INDEX + 1
#define HW_PORT_BUSYX   HW_PORT_INDEX + 2

/*Set up CLK frequency of S-7600A SDK Board for ISA BUS*/
/*CLK must match the clock frequency of SDK board*/
/*Default is 1MHz.*/
#define CF 1000000L

/*Buffer size which program required temperaryly.*/
/*Usually 1K byte is enough.*/
#define BUF_SIZE    1024

/*types definitions*/
typedef unsigned char   BYTE;

/*Global variables*/
unsigned long BAUDRATE = 57600;             /*choose between 300,1200,2400,4800,9600,19200,38400,57600,115200*/
char ISPTel[] =         "48900";            /*phone number to be dialed*/
char UserName[] =  "fred@etal";             /*username*/
char Passwd[] =       "1176";               /*password*/
char Sender[] =       "fred@etal.uri.edu";  /*email address of the sender*/
char Recipient[] =    "frederic.bahuaud@fnac.net";    /*email address of the recipient*/
char Subject[] = "Trial of Email report engine";    /*subject of the email*/
char Message[] =    "This email was sent by a s7600a chip programmed by Frederic Bahuaud,URI, 2000";

/* function declarations*/
void Telnet(char* name, char* password);            /*telnet connection to remote server*/
void SendString(char* st);                          /*Send string on serial port*/
void HwSocketInit(char * name, char * password);    /*PPP connection*/
void HwSocketClose(void);                           /*PPP disconnect*/
int  HwTcpOpen(void);                               /*Require TCP connection to the server as a client*/
int  HwTcpClose(void);                              /*Force TCP from CLOSE_WAIT status to CLOSE status*/
int  HwTcpState(int);                               /*Obtain TCP State*/
int    HwTcpSend(char* text);                       /*Send data to socket*/
int  HwTcpRcv(void);                                /*Display Data from Socket */
void DialUp(unsigned long BAUD , char * isp_num);   /*Dial to the service provider*/
void abort(void);                                   /*close PPP connection*/
int  nstep(void);                                   /*Get the instruction for the procedure followed */
void x_write( BYTE, BYTE);                          /*1 byte data write driver for S-7600A SDK Board for ISA BUS*/
BYTE x_read( BYTE );                                /*1 byte data read driver for S-7600A SDK Board for ISA BUS*/
void timer(int);                                    /*Timer*/
int SendMail(char* from,char* to,char* subject,char* message);   /*send email report*/
```

File S7600A.c

```
#include <stdio.h>        /*printf*/
#include <string.h>       /* strcat strcpy strlen*/
#include <stdlib.h>       /* toupper exit*/
#include <conio.h>        /*getch putch kbhit _inp _outp*/
#include <time.h>         /*time*/

#include <custom.h>     /*path must be added in c/c++ preprocessor settings*/

/*********************************************************/
/*                    Main                               */
/*********************************************************/

/*Start program*/
void main(void)
{
    int keyPressed = 0;
    BYTE stat,echo;

    printf("\nS-7600A email report engine Program Ver.1.0 with SDK Board ISA 1.0.\n");
    printf("URI, Frederic Bahuaud, July 2000.\n");
    printf("Compiled by Microsoft Visual C++ 6.0.\n");
    printf("baud %d, ISP %s, username %s, password %s\n", BAUDRATE, ISPTel, UserName, Passwd);

    printf("\nReset S-7600A.....");
    x_write( 0x01, 0x01 );              /*Software Reset.*/
    printf("Done.\n\n");

    while((x_read(0x08) & 0x80) !=0)/* discard invalid data in RS232 buffer*/
    {
        echo=x_read(0x0b);          /*read answer of modem, char by char*/
/*      putch(echo);                display answer of modem*/
    }

    DialUp(BAUDRATE , ISPTel);          /*Dial to the service provider*/
    Telnet(UserName, Passwd);           /*switch server from telnet to ppp protocol*/
    HwSocketInit(UserName, Passwd);  /*PPP connection*/

    while(1)
    {
        x_write(0x08, 0x01);            /*ASCTL = Hardware control Enable modem*/
        timer(1);                       /*Wait for the modem ready*/
        stat = x_read(0x08);
        stat = stat & 0x70;             /*Confirm the modem is ready*/
        if(stat != 0)                   /*Modem ready is not confirmed*/
        {
            printf("Modem Hardware Error.\n");
            while(1)
            {
                if(nstep() == FALSE)
                    abort();
            }
        }
        if(HwTcpOpen() == FALSE)        /*Require TCP connect to server*/
        {
            continue;
        }
        printf("\n");
        if(SendMail(Sender,Recipient,Subject,Message) == FALSE) /*send email*/
        {
            printf("Error occured. Message could not be sent\n");
            continue;
        }
        HwTcpClose();                   /*Change TCP from CLOSE_WAIT status to CLOSE status*/
        x_write( 0x08, 0x03 );          /*After RTS disenabled, instruct maintains connection and*/
                                        /*waiting status to the modem*/
        printf("\nPress a Key !!    S : Start again   Any other key : Exit\n");
        keyPressed = toupper(getch());
        if (keyPressed == 'S')          /*Request connection to server*/
        {
            continue;
        }
        else                            /*end*/
```

```
      {
            printf("Disconnect Sequence.\n");
            abort();                    /*Close PPP connection*/
         }
     }
}

/*************************************************************/
/*                    abort                    */
/*************************************************************/
/*Close PPP connection*/
void abort(void)
{
   HwSocketClose();     /*Disconnect PPP*/
   x_write( 0x08, 0x06 );/*disenable modem*/
   exit(1);                 /*end*/
}

/*************************************************************/
/*              DialUp                        */
/*************************************************************/

/*Dial to the service provider*/
void DialUp(unsigned long BAUD , char * isp_num)
{
   BYTE baudlo;
   BYTE baudhigh;
   BYTE clocklo;
   BYTE clockhigh;
     BYTE stat,echo;
   char atdt_isp[22] = "ATDT ";         /*ORI ATDT = 5 , isp = 11 long ditance */

   baudlo   = (BYTE)( (CF/BAUD)-1 );
   /*Set up baud rate and clock frequency of S-7600A SDK board for ISA BUS*/
     baudhigh = (BYTE)( ( (CF/BAUD)-1 ) >> 8 );
   /*Calculate register value     clocklo   = (BYTE) ( (CF/1000)-1 )*/
   clockhigh = (BYTE) ( ( (CF/1000)-1 ) >> 8 );
   clocklo = (BYTE) ( (CF/1000)-1 );

   x_write( 0x0C, baudlo );             /*BAUD_Rate_Div low*/
   x_write( 0x0D, baudhigh );           /*BAUD_Rate_Div high*/
   x_write( 0x1C, clocklo );            /*Clock_Div_Low*/
   x_write( 0x1D, clockhigh );          /*Clock_Div_High*/

   strcat(atdt_isp , isp_num);          /*concatenating ATDT with phone number*/
     SendString("AT");                  /*check data terminal ready DTR*/
     timer(1);                          /*Waiting because Status would be changed*/
     while((stat = x_read(0x08) & 0x80) !=0)  /* wait for an answer*/
     {
          echo=x_read(0x0b);            /*read answer of modem, char by char*/
          putch(echo);                  /*display answer of modem*/
     }
     SendString(atdt_isp);              /*send numbers to dial*/
     timer(1);                          /*Waiting because Status would be changed*/
     while((stat = x_read(0x08) & 0x80) !=0)  /* wait for an answer*/
     {
          echo=x_read(0x0b);            /*read answer of modem, char by char*/
          putch(echo);                  /*display answer of modem*/
     }
     while((x_read(0x08) & 0x80)==0);          /*wait for reply from ISP*/
     printf("Server contacted\n");
}



/*************************************************************/
/*              HwSocketInit                  */
/*************************************************************/
/*PPP connection*/
void HwSocketInit(char * name, char * password)
{
   BYTE addr;

   x_write( 0x08, 0x21 );               /*Make serial port to hardware control when the dial line connected
                                        SCTL = Hardware ccontrol, HWFC = active*/
```

```
   x_write( 0x62, 0x0a );                /*PPP_Max_Retry = 0x0a*/
/*    x_write( 0x60, 0x60 );              //PPP_Control_Status setting : pap enabled


//Write of PAP strings          Required if pap enabled
   x_write( 0x64, (BYTE) strlen(name) );  //username
   while( *name )
     x_write( 0x64, *name++ );

   timer(1);

   x_write( 0x64, (BYTE) strlen(password) ); //password
   while( *password )
     x_write( 0x64, *password++ );
   x_write( 0x64, 0  );                          //Null Termination
*/
   x_write( 0x60, 0x42 );            /*PPP Enable ?should be 62 if pap enabled?*/
   while ( ( x_read( 0x60 ) & 0x01 ) == 0x00 )
     {
          if(kbhit() != 0)
          {
               abort();                 /*If something is input before PPP\Up confirmed */
          }                             /*Disconnect PPP and dial line,End */
     }

     printf("Done. PPP Connection Established. \n");
     printf("ISP assigned our IP Address = ");
     for(addr = 0x13; addr >= 0x10; addr--)    /*Display IP address allocated by server*/
     {
      if(addr == 0x10){
         printf("%d\n", x_read(addr));
       }
       else{
         printf("%d.", x_read(addr));
       }
   }
}

/*************************************************************/
/*                      HwSocketClose                */
/*************************************************************/

/*Close PPP*/
void HwSocketClose(void)
{
     printf("PPP Disable.....");
     x_write( 0x60, 0x00 );                          /*PPP Disenable*/
     while ( ( x_read( 0x60 ) & 0x01 ) != 0x00 );   /*Confirming PPP_Down*/
     printf("Done. PPP Connection Down.\n");     /*PPP_Down confirmed*/

}

/*************************************************************/
/*                  HwTcpOpen                        */
/*************************************************************/

/*Require TCP connection as server*/
int HwTcpOpen(void)
{
   BYTE addr;

     printf("Tcp Connect.....");
     x_write( 0x20, 0x00 );          /*Select Socket 0*/
     x_write( 0x22, 0x10 );          /*Reset Socket 0  */

     x_write( 0x3c, 21);             /*Destination IP address*/
     x_write( 0x3d, 1);              /*Select etal.uri.edu server*/
     x_write( 0x3e, 128);
     x_write( 0x3f, 131);

     x_write( 0x36, 25);             /*Destination port is 25*/
     x_write( 0x37, 0);              /*SMTP mail server*/

     x_write( 0x22, 02);             /*TCP Client Mode*/
     x_write( 0x24, 01);             /*Activate Socket 0*/
```

```
    while ((x_read(0x23) & 0x10) != 0x10)    /*Confirming TCP connection*/
    {
        if(nstep() == FALSE)
        {
            return(FALSE);
         }
   }
    printf("Done. TCP Connection Established.\n\n");
    printf("Server IP address = ");

     for(addr = 0x3f; addr >= 0x3c; addr--)
     {
        if(addr == 0x3c)            /*Display IP address of server*/
        {
            printf("%d\n", x_read(addr));
        }
        else
        {
            printf("%d.", x_read(addr));
        }
     }
   return(TRUE);            /*TCP connection confirmed*/
}

/*************************************************************/
/*                    HwTcpClose                             */
/*************************************************************/

/*From CLOSE_WAIT status of TCP to CLOSE status*/
int HwTcpClose(void)
{
   printf("Tcp Close....");
   x_write( 0x20, 0x00 );           /*Select Socket 0*/
   x_write(0x24,0x00);              /*Deactivate Socket 0*/
   while (x_read(0x23) & 0x10)   /*Confirming TCP No connection*/
    {
     if(nstep() == FALSE)          /*If something input before TCP No connection confirmed*/
        {
            return(FALSE);
        }
   }
   x_write( 0x22, 0x10 );           /*Reset Socket 0*/
   printf("Done. TCP No Connection Established. \n");
   return(TRUE);
}

/*************************************************************/
/*                    HwTcpState                             */
/*************************************************************/

/*Obtain TCP State*/
/*disp : Displayed or not*/
int HwTcpState(int disp)
{
    BYTE stat;

   stat = x_read(0x23);
   if(disp == TRUE){
      printf("TCP State is %02X ", stat);
      stat = stat & 0x0f;
      switch(stat){
         case(0x0):printf("CLOSED.\n");      break;
         case(0x1):printf("SYN_SENT.\n");    break;
         case(0x2):printf("ESTABLISHED.\n"); break;
         case(0x3):printf("CLOSE_WAIT.\n");  break;
         case(0x4):printf("LAST_ACK.\n");    break;
         case(0x5):printf("FIN_WAIT1.\n");   break;
         case(0x6):printf("FIN_WAIT2.\n");   break;
         case(0x7):printf("CLOSING.\n");     break;
         case(0x8):printf("TIME_WAIT.\n");   break;
         case(0x9):printf("LISTEN.\n");      break;
         case(0xa):printf("SYN_RECVD.\n");   break;
         default:                           break;
      }
```

```
    }
    return(stat & 0x0f);
}


/*********************************************************/
/*                    nstep                              */
/*********************************************************/

/*Get instruction for procedure followed*/
int nstep(void)
{
    if(kbhit() != 0)/*If there's no input, return immediately*/
      {
          getch();
          printf("\n");
          return(FALSE);
      }
    return(TRUE);
}

/*********************************************************/
/*                    x_write(S7600A)                    */
/*********************************************************/

/*Driver of 1byte data write for S-7600A SDK Board for ISA BUS*/
void x_write( BYTE addr, BYTE data )
{
    BYTE busyx;

      do
      {
          busyx = ReadPCPort( HW_PORT_BUSYX ) & 0x80;/*Confirming busyx=1*/
      }while (busyx == 0);                           /*busyx = 1 is confirmed*/
      WritePCPort( HW_PORT_INDEX, addr );            /*Set index register address*/
      WritePCPort( HW_PORT_DATA, data );             /*Write data at S-7600A index register*/

}

/*********************************************************/
/*                    x_read(S7600A)                     */
/*********************************************************/

/*Driver of 1byte data read for S-7600A SDK Board for ISA BUS*/
BYTE x_read ( BYTE addr )
{
    BYTE busyx;

      do
      {
          busyx = ReadPCPort( HW_PORT_BUSYX ) & 0x80;/*Confirming busyx = 1*/
      }while (busyx == 0);                           /*busyx = 1 is confirmed*/
      WritePCPort( HW_PORT_INDEX, addr );            /*Set index register address*/
      ReadPCPort( HW_PORT_INDEX );                   /*Read index register*/
      do
      {
          busyx = ReadPCPort( HW_PORT_BUSYX ) & 0x80;/*Confirming busyx = 1*/
      }while (busyx == 0);                           /*busyx = 1 is confirmed*/
      return(ReadPCPort( HW_PORT_DATA ));            /*Read data at S-7600A index register*/
}

/*********************************************************/
/*                    timer                              */
/*********************************************************/

void timer(int sec)
{

/*sec : Wait time. Unit is Sec. And there's +1 Sec tolerance at max*/
    long    oldtime, newtime;
    int i = -1;

    time(&oldtime);
    while(1){
       time(&newtime);
```

```
      if(newtime != oldtime){
        oldtime = newtime;
        i++;
      }
      if(i >= sec){
        return;
      }
    }
}


/*************************************************************/
/*                    telnet connection                      */
/*************************************************************/
void Telnet(char* name, char* password)
{
     char echo;
     int cyata1=0;
     BYTE stat;

     while((stat = x_read(0x08) & 0x80) !=0)  /*receive the prompt from ISP*/
          echo=x_read(0x0b);
     printf("Telnet data is being sent...Wait...");
     SendString(name);                        /*send name*/
     while((stat = x_read(0x08) & 0x80) !=0)  /* wait for reply*/
          echo=x_read(0x0b);
     SendString(password);                    /*send password*/
     while((stat = x_read(0x08) & 0x80) !=0)  /* wait for reply*/
          echo=x_read(0x0b);
     SendString("ppp");                       /*ask server for ppp connection*/
     while((stat = x_read(0x08) & 0x80) !=0)  /* wait for ppp prompt*/
          echo=x_read(0x0b);
     printf("OK...\n");

}


/*************************************************************/
/*                    Send string to remote server           */
/*************************************************************/
void SendString(char* st)
{
     BYTE stat;
     char buf[BUF_SIZE + 1];
     int len, i;

     x_write(0x08, 0);                /*SCTL = MPU control*/
   timer(1);                          /*Waiting for safety reasons, because Status would be changed */
     stat = x_read(0x08) & 0x20; /*Confirm modem DSR*/
   if(stat != 0)                      /*Can not confirm DSR of the modem*/
     {
          printf("Modem Hardware Error.\n");
          exit(0);
   }
   strcpy(buf, st);
   strcat(buf, "\r");                 /*Add terminator CR:0X0d:\r to command character line*/
     len = strlen(buf);
   for(i = 0; i < len; i++)
     {                                /*Serial Port Data Register(0x0b)*/
      x_write(0x0b, (BYTE)(buf[i]));   /*Write 1 char by 1 char*/
   }
}


/*************************************************************/
/*                    HwTcpSend                              */
/*************************************************************/

/*Send Socket data*/
int HwTcpSend(char* text)
{
     int len,i;
     char buf[BUF_SIZE];

     strcpy(buf,text);
```

```
    strcat(buf, "\n");                          /*Add terminator CR:0X0d:\r to command character line*/
    len=strlen(buf);
    for(i = 0; i < len; i++)
    {
        if((x_read(0x22) & 0x20) == 0)          /*If socket not full*/
        {
            x_write(0x2e, (BYTE)(buf[i]));      /*Write 1 char by 1 char*/
            printf("%c",buf[i]);                /*Read 1 char from Socket data Resister*/
            continue;
        }
        else
        {
            printf("\nBuffer full.");
            return(FALSE);
        }
    }
    x_write(0x30, 0x00);                        /* start sending data*/
  return(TRUE);
}


/*************************************************************/
/*                  HwTcpRcv                                 */
/*************************************************************/

/*Display Socket data*/
int HwTcpRcv(void)
{
  while((x_read(0x28) & 0x01) == 0)             /*wait for data available*/
    {
        if(nstep() == FALSE)
            return(FALSE);
    }
    while(x_read(0x28) & 0x01)  /*while data available...*/
    {
        if(nstep() == FALSE)
        {
            return(FALSE);
        }
        putch(x_read(0x2e));                    /*display it */
    }
    return(TRUE);
}


/*************************************************************/
/*                  SendMail                                 */
/*************************************************************/

/*Send an email using smtp strings (mail from:, rcpt to:, data)*/
/*from: sender of the message*/
/*to: recipient of the message*/
/*subject: subject of the message*/
/*message: email message*/

int SendMail(char* from,char* to,char* title,char* data)
{
    char buf[BUF_SIZE];

    if(HwTcpRcv() == FALSE)         /*receive prompt*/
        return(FALSE);
    printf("\n");

    strcpy(buf,"mail from:");       /*copy smtp command to specify sender*/
    strcat(buf,from);
    if(HwTcpSend(buf) == FALSE)     /*send sender of the message*/
        return(FALSE);
    if(HwTcpRcv() == FALSE)         /*display result of operation*/
        return(FALSE);
    printf("\n");

    strcpy(buf,"rcpt to:");         /*copy smtp command to specify the recipient*/
    strcat(buf,to);
    if(HwTcpSend(buf) == FALSE)     /*send receiver of the message*/
        return(FALSE);
```

```
    if(HwTcpRcv() == FALSE)          /*display result of operation*/
        return(FALSE);
    printf("\n");

    strcpy(buf,"data\n");            /*copy smtp command to start writing message*/
    strcat(buf,"Subject:");
    strcat(buf,title);               /*add subject at the beginning of the message*/
    strcat(buf,"\n");
    if(HwTcpSend(buf) == FALSE)      /* begin email message*/
        return(FALSE);
    if(HwTcpRcv() == FALSE)          /*display result of operation*/
        return(FALSE);

    strcpy(buf,data);                /*add message*/
    strcat(buf, "\n.\n");            /*Add smtp message terminator*/
    if(HwTcpSend(buf) == FALSE)
        return(FALSE);
    if(HwTcpRcv() == FALSE)          /*display result of operation*/
        return(FALSE);

    return(TRUE);
}
```

APPENDIX H: PROGRAM OF THE S^2B IN C

Main.h

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/

#pragma language=extended /*enable use of extended keywords*/

#include<IO_s2b.h>
#include<io6811.h>

void init_system(void);
void problem(int);
extern void timer(int);
extern int communication(int);
extern int maintenance(void);
extern int operation(void);      /*performs an automatic routine of deflation/reinflation*/
```

operation.h

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/

#include<IO_s2b.h>

int WAIT_EMERGENCY=7;         /*lapse time in seconds between deflation and re-inflation*/

int operation(void);                    /*performs an automatic routine of deflation/reinflation*/
void flashing_light_on(void);
void flashing_light_off(void);
extern void timer(int);
```

maintenance.h

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/

#include<io6811.h>
#include<IO_s2b.h>

int maintenance(void);
extern   void init_system(void);
```

communication.h

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/

#include <string.h>                          /* strcat strcpy strlen*/
#include<io6811.h>                           /*include declaration of 68HC11 registers*/
#include<s7600_reg.h>                        /*include declaration of S7600A registers*/
#include<IO_s2b.h>                           /*include declaration of s2b I/O*/

#define INDEX (* (unsigned char *) (0x6000))  /* INDEX register of the S7600A */
#define DATA (* (unsigned char *) (0x6001))   /* DATA register of the S7600A */
#define BUSYX (PORTD & 0x20)                  /* BUSYX signal from S7600A*/
#define CF       1000000L                     /*Set up CLK frequency of S-7600A*/
#define BUF_SIZE  200                         /*Buffer size which program required temperaryly.*/


typedef unsigned char BYTE;                   /*define type BYTE used to R/W iChip registers*/


/*global variables declarations*/
unsigned long BAUDRATE = 57600;     /*choose between 300,1200,2400,4800,9600,19200,38400,57600,115200*/
char ISPTel[]=    "48900";                    /*phone number to be dialed*/
char UserName[]=  "fred@etal";                /*username*/
char Passwd[]=    "1176";                     /*password*/
char Sender[]=    "fred@etal.uri.edu";        /*email address of the sender*/
char Recipient[]= "frederic.bahuaud@fnac.net";/*email address of the recipient*/
char MESSAGE_0[]= "Message sent from speed bump control system, for maintenance purposes.";
char SUBJECT_0[]= "S2B maintenance";          /*subject of the email*/


/*function declarations*/
int communication(int type_of_email);         /*main program to send email reports*/
void x_write( BYTE addr, BYTE data );         /*write a byte in a S7600A register*/
BYTE x_read ( BYTE addr );                    /*read a byte in a S7600A register*/
void timer(int sec);                          /*timer*/
int SendString(char* st);                     /*Send string on serial port*/
int DialUp(unsigned long BAUDT , char* isp_num);/*Dial to the service provider*/
int Telnet(char* name, char* password);       /*telnet connection to remote server*/
int HwSocketInit(char * name, char * password);/*PPP connection*/
void HwSocketClose(void);                     /*PPP disconnect*/
int  HwTcpOpen(void);                         /*Require TCP connection to the server as a client*/
int  HwTcpClose(void);                        /*Force TCP from CLOSE_WAIT status to CLOSE status*/
int  HwTcpSend(char* text);                   /*Send data to socket*/
int  HwTcpRcv(void);                          /*Display Data from Socket */
void abort(void);                             /*close PPP connection*/
int  SendMail(char* from, char* to, char* subject, char* message);/*send email report*/


/*******************************************************/
/*                  x_write(S7600A)                    */
/*******************************************************/
/*data write in S-7600A registers*/
void x_write( BYTE addr, BYTE data )
{
  while (BUSYX == 0);     /*wait for busyx signal*/
  INDEX=addr;             /*Set index register address*/
  DATA=data;              /*Write data at S-7600A index register*/

}



/*******************************************************/
/*                  x_read(S7600A)                     */
/*******************************************************/
/*data read in S-7600A registers*/
```

```
BYTE x_read ( BYTE addr )
{
  BYTE trash;

  while (BUSYX == 0);     /*wait for busyx signal*/
  INDEX=addr;             /*Set index register address*/
  trash=INDEX;            /*Read index register*/
  while (BUSYX == 0);     /*wait for busyx signal*/
  return(DATA);           /*Read data at S-7600A index register*/
}
```

main.c

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/
/*This is the main program that runs the speed bump */
/*it does nothing but monitoring inputs*/
/*if an event is detected, appropriate functions are called to handle the event*/

#include<main.h>              /*include files, function definitions, typedef, constants defs...*/

void main (void)
{
  int id=0;                   /*identifies the type of problem*/
  init_system();              /*initialize system*/

  while(1)
  {
    if(BTN_MAINT != 0)        /*if manual mode selected (maintenance)*/
    {
      id = maintenance();     /*go in maintenance mode*/
      if( id != 0) problem(id);   /*error handling, if any*/
    }                         /* + problem handeling if any*/
    if(EM_REQUEST != 0)       /*if request from emergency vehicle*/
    {
      id = operation();       /*go in operation mode*/
      if( id != 0) problem(id);   /*error handling, if any*/
    }
  }
}

void init_system(void)
{
  SCCR2=0x0;                  /*disable serial communication mode (to use PORTD as input)*/
  DDRD=0x0;                   /*select PORT D as input*/
  PORTB=0x0;                  /*reset outputs*/
  S_FLASH;                    /*turn on flashing light*/
  while(HI_PRESSURE == 0)     /*if pressure in s2b is not high enough*/
    S_INFLATE;                /*inflate*/
  R_INFLATE;                  /*stop inflation*/
}

void problem(int problem_id)
{
  int err;
  switch (problem_id)                 /*take an action depending on problem*/
  {
    case 1: S_ERROR;                  /*fatal error, transmission impossible*/
          break;
    case 2: err = communication(problem_id);  /*report problem by email*/
          if(err != 0) problem(1);
          break;
    case 3: S_ERROR;                  /*hardware modem error*/
    case 4: S_ERROR;                  /*telnet connection failed*/
    case 5: S_ERROR;                  /*TCP open failed*/
    default:S_ERROR;
  }
  timer(3);                           /*for debug purposes only*/
  R_ERROR;                            /*reset error led after 3 seconds*/
}
```

operation.c

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/
/*performs the deflation/inflation cycle, in case of emergency request*/

#include<operation.h>

int operation(void)
{
  /*deflation cycle*/
  while(LO_PRESSURE != 0)     /*deflate s2b until low pressure*/
    S_DEFLATE;
  R_DEFLATE;                  /*stop deflating*/
  R_FLASH;                    /*stop flashing light*/
  timer(WAIT_EMERGENCY);      /*wait for a certain time*/

  /*inflation cycle*/
  S_FLASH;                    /*turn on flashing light*/
  while(HI_PRESSURE == 0)     /*inflate s2b until high pressure*/
    S_INFLATE;
  R_INFLATE;                  /*stop inflating*/
  return 2;                   /*for debug purposes, return an error so that email is sent*/
}

/*enable RTI for flashing light*/
/*not implemented yet*/
void flashing_light_on(void)
{}

/*disable RTI for flashing light*/
/*not implemented yet*/
void flashing_light_off(void)
{}
```

maintenance.c

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/
/*allows manual operation of the speed bump, for maintenance*/

#include<maintenance.h>

int maintenance(void)
{
  S_MAINT_LED;                  /*turn on maintenance LED*/
  while(BTN_MAINT != 0)         /*while in maintenance m ode*/
  {
    while(BTN_INF != 0)         /*manual inflation*/
    {
      if(HI_PRESSURE == 0)      /*if pressure in s2b not too high*/
        S_INFLATE;              /*inflate*/
      else
      {
        R_INFLATE;              /*stop inflation*/
        S_FLASH;                /*turn on flashing light*/
      }
    }
    R_INFLATE;                  /*stop inflation*/

    while(BTN_DEF != 0)         /*manual deflation*/
    {
      if(LO_PRESSURE != 0)      /*if pressure in s2b is not low enough*/
        S_DEFLATE;              /*deflate*/
      else
      {
        R_DEFLATE;              /*stop deflation*/
        R_FLASH;                /*turns off flashing light*/
      }
    }
    R_DEFLATE;                  /*stop deflation*/
  }
  R_MAINT_LED;                  /*turn off maintenance LED*/
  init_system();                /*initialize system for normal operation*/
  return 0;                     /*return no error*/
}
```

communication.c

```
/*File written by Frederic BAHUAUD, URI, ESPEO, 2000*/
/*send an email report in case of s2b problem*/
/*the email body depends on the type of problem that occured*/

#include<communication.h>

/*********************************************************/
/*              Communication                            */
/*********************************************************/
/*Send an email report with a message depending on type_of_email*/
int communication(int type_of_email)
{
  BYTE echo,stat;

  S_COMMU_LED;                                  /*turn on communication LED*/
  x_write( 0x01, 0x01 );                        /*Software Reset.*/

  while((x_read(0x08) & 0x80) !=0)              /* discard invalid data in RS232 buffer*/
    echo=x_read(0x0b);                          /*read answer of modem, char by char*/

  if( DialUp(BAUDRATE , ISPTel) != 0) return 3;     /*Dial to the service provider*/
  if( Telnet(UserName, Passwd) != 0) return 4;      /*switch server from telnet to ppp protocol*/
  if( HwSocketInit(UserName, Passwd) !=0) return 5;   /*PPP connection*/

  x_write(0x08, 0x01);                          /*ASCTL = Hardware control Enable modem*/
  timer(1);                                     /*Wait for the modem ready*/
  stat = x_read(0x08);
  stat = stat & 0x70;                           /*Confirm the modem is ready*/
  if(stat != 0)                                 /*Modem ready is not confirmed*/
    return (1);
  if(HwTcpOpen() == 1)                          /*Require TCP connect to server*/
    return (1);
  if(SendMail(Sender, Recipient, SUBJECT_0, MESSAGE_0) == 1) /*send email*/
    return 1;
  HwTcpClose();                                 /*Change TCP from CLOSE_WAIT status to CLOSE status*/
  HwSocketClose();                              /*close ppp connection*/
  if( SendString("ATH") != 0) return (3);
  x_write( 0x08, 0x06 );                        /*After RTS disenabled, instruct maintains connection*/

  R_COMMU_LED;                                  /*turn off communication LED*/
  return (0);                                   /*return no error*/
}

/*********************************************************/
/*              DialUp                           */
/*********************************************************/
/*Dial to the service provider*/
int DialUp(unsigned long BAUDT, char* isp_num)
{
  BYTE baudlo;
  BYTE baudhigh;
  BYTE clocklo;
  BYTE clockhigh;
  BYTE stat,echo;
  char atdt_isp[22] = "ATDT ";                  /*ORI ATDT = 5 , isp = 11 long ditance */

  baudlo   = (BYTE)( (CF/BAUDT)-1 );            /*Set up baud rate and clock frequency of S-7600A*/
  baudhigh = (BYTE)( ( (CF/BAUDT)-1 ) >> 8 ); /*clocklo   = (BYTE) ( (CF/1000)-1 )*/
  clockhigh = (BYTE) ( ( (CF/1000)-1 ) >> 8 );
  clocklo = (BYTE) ( (CF/1000)-1 );

  x_write( 0x0C, baudlo );              /*BAUD_Rate_Div low*/
  x_write( 0x0D, baudhigh );            /*BAUD_Rate_Div high*/
  x_write( 0x1C, clocklo );             /*Clock_Div_Low*/
```

```
  x_write( 0x1D, clockhigh );            /*Clock_Div_High*/

  strcat(atdt_isp , isp_num);            /*concatenating ATDT with phone number*/
  if( SendString("AT") !=0) return (3);   /*check data terminal ready DTR*/
  timer(1);                              /*Waiting because Status would be changed*/
  while((stat = x_read(0x08) & 0x80) !=0) /* wait for an answer*/
  {
    echo=x_read(0x0b);                   /*read answer of modem, char by char*/
//    putch(echo);                       /*display answer of modem*/
  }
  if( SendString(atdt_isp)!=0) return (3);/*send numbers to dial*/
  timer(1);                              /*Waiting because Status would be changed*/
  while((stat = x_read(0x08) & 0x80) !=0) /* wait for an answer*/
  {
    echo=x_read(0x0b);                   /*read answer of modem, char by char*/
//    putch(echo);                       /*display answer of modem*/
  }
  while((x_read(0x08) & 0x80)==0);  /*wait for reply from ISP*/
  return (0);
}

/*********************************************************/
/*                Timer                                  */
/*********************************************************/
/* timer of sec seconds*/
void timer(int sec)
{
  int i,j,k;
  for (i=0;i<sec;i++)                    /*do nothing*/
    for (j=0;j<255;j++)
      for (k=0;k<255;k++);
}


/*********************************************************/
/*                        abort                          */
/*********************************************************/
/*Close PPP connection*/
void abort(void)
{
  HwSocketClose();        /*Disconnect PPP*/
  x_write( 0x08, 0x06 );  /*disenable modem*/
}


/*********************************************************/
/*                HwSocketInit                           */
/*********************************************************/
/*PPP connection*/
int HwSocketInit(char * name, char * password)
{
  BYTE addr;

  x_write( 0x08, 0x21 );          /*Make serial port to hardware control when the dial line connected
                                  SCTL = Hardware ccontrol, HWFC = active*/
  x_write( 0x62, 0x0a );          /*PPP_Max_Retry = 0x0a*/
/*  x_write( 0x60, 0x60 );        //PPP_Control_Status setting : pap enabled


//Write of PAP strings       Required if pap enabled
  x_write( 0x64, (BYTE) strlen(name) ); //username
  while( *name )
    x_write( 0x64, *name++ );
  timer(1);
  x_write( 0x64, (BYTE) strlen(password) ); //password
  while( *password )
```

```
    x_write( 0x64, *password++ );
  x_write( 0x64, 0  );              //Null Termination
*/
  x_write( 0x60, 0x42 );            /*PPP Enable ?should be 62 if pap enabled?*/
  while ( ( x_read( 0x60 ) & 0x01 ) == 0x00 )
  {
  }
//  printf("Done. PPP Connection Established. \n");
/*  printf("ISP assigned our IP Address = ");
  for(addr = 0x13; addr >= 0x10; addr--) //Display IP address allocated by server
  {
    if(addr == 0x10){
      printf("%d\n", x_read(addr));
    }
    else{
    printf("%d.", x_read(addr));
    }
  }*/
  return (0);
}

/*************************************************************/
/*                        HwSocketClose                      */
/*************************************************************/
/*Close PPP*/
void HwSocketClose(void)
{
//  printf("PPP Disable.....");
  x_write( 0x60, 0x00 );                                /*PPP Disenable*/
  while ( ( x_read( 0x60 ) & 0x01 ) != 0x00 );      /*Confirming PPP_Down*/
//  printf("Done. PPP Connection Down.\n");      /*PPP_Down confirmed*/
}

/*************************************************************/
/*                    HwTcpOpen                              */
/*************************************************************/
/*Require TCP connection as server*/
int HwTcpOpen(void)
{
  BYTE addr;

//  printf("Tcp Connect.....");
  x_write( 0x20, 0x00 );    /*Select Socket 0*/
  x_write( 0x22, 0x10 );    /*Reset Socket 0 */

  x_write( 0x3c, 21);       /*Destination IP address*/
  x_write( 0x3d, 1);        /*Select etal.uri.edu server*/
  x_write( 0x3e, 128);
  x_write( 0x3f, 131);

  x_write( 0x36, 25);       /*Destination port is 25*/
  x_write( 0x37, 0);        /*SMTP mail server*/

  x_write( 0x22, 02);       /*TCP Client Mode*/
  x_write( 0x24, 01);                           /*Activate Socket 0*/

  while ((x_read(0x23) & 0x10) != 0x10) /*Confirming TCP connection*/
  {
  }
/*  printf("Done. TCP Connection Established.\n\n");
  printf("Server IP address = ");

  for(addr = 0x3f; addr >= 0x3c; addr--)
  {
    if(addr == 0x3c)       //Display IP address of server
    {
```

```
    printf("%d\n", x_read(addr));
   }
   else
   {
    printf("%d.", x_read(addr));
   }
  }*/
  S_TCP_LED;                          /*turn off TCP led*/
  return (0);
}


/*************************************************************/
/*                    HwTcpClose                    */
/*************************************************************/
/*From CLOSE_WAIT status of TCP to CLOSE status*/
int HwTcpClose(void)
{
//  printf("Tcp Close....");
  x_write( 0x20, 0x00 );              /*Select Socket 0*/
  x_write(0x24,0x00);                 /*Deactivate Socket 0*/
  while (x_read(0x23) & 0x10)         /*Confirming TCP No connection*/
  {
  }
  x_write( 0x22, 0x10 );              /*Reset Socket 0*/
//  printf("Done. TCP No Connection Established. \n");
  R_TCP_LED;                          /*end of TCP connection*/
  return(0);
}


/*************************************************************/
/*                  telnet connection                */
/*************************************************************/
/*send telnet data to switch in ppp mode*/
int Telnet(char* name, char* password)
{
  char echo;
  int cyata1=0;
  BYTE stat;

  while((stat = x_read(0x08) & 0x80) !=0)          /*receive the prompt from ISP*/
    echo=x_read(0x0b);
//  printf("Telnet data is being sent...Wait...");
  SendString(name);                                /*send name*/
  while((stat = x_read(0x08) & 0x80) !=0)          /* wait for reply*/
    echo=x_read(0x0b);
  SendString(password);                            /*send password*/
  while((stat = x_read(0x08) & 0x80) !=0)          /* wait for reply*/
    echo=x_read(0x0b);
  SendString("ppp");                               /*ask server for ppp connection*/
  while((s tat = x_read(0x08) & 0x80) !=0)         /* wait for ppp prompt*/
    echo=x_read(0x0b);
//  printf("OK...\n");
  return (0);
}

/*************************************************************/
/*                  Send string to remote server     */
/*************************************************************/
int SendString(char* st)
{
  BYTE stat;
  char buf[BUF_SIZE + 1];
  int len, i;
```

```
  x_write(0x08, 0);                 /*SCTL = MPU control*/
  timer(1);                         /*Waiting for safety reasons, because Status would be changed */
  stat = x_read(0x08) & 0x20;       /*Confirm modem DSR*/
  if(stat != 0)                     /*Can not confirm DSR of the modem*/
  {
//    printf("Modem Hardware Error.\n");
    return (3);
  }
  strcpy(buf, st);
  strcat(buf, "\r");                /*Add terminator CR:0X0d:\r to command character line*/
  len = strlen(buf);
  for(i = 0; i < len; i++)
  {                                 /*Serial Port Data Register(0x0b)*/
    x_write(0x0b, (BYTE)(buf[i]));  /*Write 1 char by 1 char*/
  }
  return (0);
}

/*********************************************************/
/*                  HwTcpSend                    */
/*********************************************************/
/*Send Socket data*/
int HwTcpSend(char* text)
{
  int len,i;
  char buf[BUF_SIZE];

  strcpy(buf,text);                 /*copy string to send in a buffer*/
  strcat(buf, "\n");                /*Add terminator CR:0X0d:\r to command character line*/
  len=strlen(buf);                  /*get length of the string*/
  for(i = 0; i < len; i++)
  {
    if((x_read(0x22) & 0x20) == 0)  /*If socket not full*/
    {
      x_write(0x2e, (BYTE)(buf[i]));  /*Write 1 char by 1 char*/
//      printf("%c",buf[i]);          /*Read 1 char from Socket data Resister*/
      continue;
    }
    else
    {
      x_write(0x30, 0x00);          /* start sending data*/
    }
  }
  x_write(0x30, 0x00);              /* start sending data*/
  return  (0);
}

/*********************************************************/
/*                  HwTcpRcv                     */
/*********************************************************/
/*Display Socket data*/
int HwTcpRcv(void)
{
  while((x_read(0x28) & 0x01) == 0)     /*wait for data available*/
  { }
  while(x_read(0x28) & 0x01)            /*while data available...*/
  {
    x_read(0x2e);                       /*store it (not implemented yet)*/
  }
  return  (0);
}

/*********************************************************/
/*                  SendMail                     */
/*********************************************************/
/*Send an email using smtp strings (mail from:, rcpt to:, data)*/
```

```
/*from: sender of the message*/
/*to: recipient of the message*/
/*subject: subject of the message*/
/*message: email message*/
int SendMail(char* from, char* to, char* title, char* data)
{
  char buf[BUF_SIZE];

  if(HwTcpRcv() == 1)          /*receive prompt*/
    return(1);
//  printf("\n");
  strcpy(buf,"mail from:");    /*copy smtp command to specify sender*/
  strcat(buf,from);
  if(HwTcpSend(buf) == 1)      /*send sender of the message*/
    return(1);
  if(HwTcpRcv() == 1)          /*display result of operation*/
    return(1);
//  printf("\n");

  strcpy(buf,"rcpt to:");      /*copy smtp command to specify the recipient*/
  strcat(buf,to);
  if(HwTcpSend(buf) == 1)      /*send receiver of the message*/
    return(1);
  if(HwTcpRcv() == 1)          /*display result of operation*/
    return(1);
//  printf("\n");
  strcpy(buf,"data\n");        /*copy smtp command to start writing message*/
  strcat(buf,"Subject:");
  strcat(buf,title);           /*add subject at the beginning of the message*/
  strcat(buf,"\n");
  if(HwTcpSend(buf) == 1)      /* begin email message*/
    return(1);
  if(HwTcpRcv() == 1)          /*display result of operation*/
    return(1);
  strcpy(buf,data);            /*add message*/
  strcat(buf, "\n.\n");        /*Add smtp message terminator*/
  if(HwTcpSend(buf) == 1)
    return(1);
  if(HwTcpRcv() == 1)          /*display result of operation*/
    return(1);
  return(0);
}
```

APPENDIX I: COMPARISON OF C CROSS COMPILERS

Company name	ARCHIMEDE	IAR	HI-TECH	HIWARE	COSMIC SOFT.
Contents of software package	• ANSI C compiler (C++ optional) • Assembler • Linker • ANSI librarian • Debugger • SimCase simulator • EPROM burner	• ANSI C compiler • Macro assembler • Linker • ANSI librarian • Debugger • CSpy simulator	• ANSI C compiler • Macro assembler • Linker • Librarian with source • Debugger + remote debugger	• ANSI C compiler (C++ optional) • Macro assembler • Linker • Librarian • HI-WAVE Debugger • HI-WAVE simulator • Decoder • EPROM burner	• ANSI + ISO C compiler • Macro assembler • Linker • Librarian • ZAP remote debugger • ZAP simulator • HEX file generator • Object format converter
Warranty	3 months	1 year		6 months	1 year
Support	$ 449/year	free		6 months free	1 year free
Normal unit Price	$ 2,895	$ 2,395	$ 850	$ 3,880	$ 2,310
Educational purpose prices	30% discount + special anniversary price: $ 1,956.50	6 licenses: $ 0	1 license: $ 500 5 licenses: $ 850 10 licenses: $ 1,700	50% discount $ 1,940	40% discount 1 license: $1,650 10 licenses: $ 4,125
Interface user friendliness	Versatile but not user friendly. Use Windows notepad as text editor.	Integrated environment. GREAT!!!	DOS interface ?	Versatile but not user friendly. Use notepad as text editor.	Integrated environment. User friendly.
Miscellaneous	HIWARE software!!!	Advertising for IAR on URI web site	No simulator		

Table 17: Comparison of C cross compilers