Homework 10: Software Design Considerations, Narrative, and Documentation

Due: Thursday, April 8, at Classtime

Team Code Name: Chateau de Nemo Group No. 1

Team Member Completing This Homework: Jason LitJeh Lim

1. Introduction

A microcontroller is used to monitor a household aquarium and maintain its living conditions. This paper will discuss the software needs and considerations taken note of in the operation of such a device.

Our smart aquarium device, otherwise known as “Chateau de Nemo”, features internet accessible capabilities for the frequent traveler and thus would require software support for user communication and interaction with the device over the internet. The software would have to support a user interface for modification of aquarium settings and reporting of current aquarium conditions. At the same time our software must also support local setting modifications via a LCD display/keypad manufactured and marketed by Rabbit Semiconductors. This LCD display/keypad module will also serve as the software’s “window to the world”, where the software can communicate messages to users.

One of our team objectives is to maintain the aquarium so that the fishes don’t die, and thus it is pertinent that the user will be informed of any drastic changes in aquarium conditions that could be fatal to its inhabitants. Up to date information allows for the user to take prompt action (i.e. calling upon a neighbor to do something if he/she isn’t able to at the moment) and prevent fatal casualties. This responsibility of notifying the user of any irregular conditions will be held by the software for this device. Locally the software will have to light up an LED on the LCD display/keypad module to alert local users and when a user establishes connection via the internet, he/she will also be informed by way of messages or notifications. As this information is time sensitive material, a time stamp should come with any messages or notifications.

2. Software Design Considerations

2.1 Memory Usage

Our main memory usage is limited to three different blocks. The main program code, current conditions, and user defined desired conditions.

FLASH: Flash memory on the Rabbit Core Module (RCM) 3000 will hold our main program code and this is where the software program will run off. This is a good place to hold the main program code as flash memory can only be programmed a specific number of times before its reliability to retain data drops; we only program the flash a few times during the manufacturing process and during the prototyping process a few dozen times.

SRAM: SRAM onboard the RCM 3000 will store more long term variables like current conditions and user defined desired conditions. As these variables are often read and written, it is a good idea to put them in SRAM to prevent loss of data from excessive reading/writing.

In the main program code, current conditions and user defined desired conditions are declared as global variables so that they are accessible throughout the entire program as they need to be modified often.

Making use of Rabbit Semiconductor's Dynamic C program, we will allow for exact memory mappings to be handled by this program. As the Dynamic C program was design specifically for Rabbit microcontrollers, it will be able to do a much better job of memory mapping than if we were to attempt it by integrating assembly code into our C code.

2.2 Startup Code/Initialization Routine

This is an important section of our software program. Chateau de Nemo is heavily dependent on its I/O pins to communicate with its many peripherals to

keep track of aquarium conditions and modify them as necessary. It is of great import that the I/O pins are setup as input or output correctly on reset and if they are output pins, output the right signal. For example, it would be disastrous to have the pin controlling the heating element reset asserted, effectively cooking the fishes slowly.

On reset, our I/O pins have been setup to be at the conditions as shown in Table 2.1 on the following page.

Port	I/O	I/O State	Chateau de Nemo Use
PB0	Output	High	LCD Module LED 1 (Power)
PB2	Output	Low	LCD Module LED 2
PB3	Output	Low	LCD Module LED 3
PB4	Output	Low	LCD Module LED 4
PB5	Output	Low	LCD Module LED 5
PB6	Output	Low	LCD Module LED 6
PB7	Output	Low	LCD Module LED 7
PC0	Output	Low	Light
PC1	Input	N/A	Water Level Low
PC2	Output	Low	Heater
PC3	Input	N/A	Water Level High
PC4	Output	Low	Auto Feeder
PC5	Input	N/A	Water Level Critical
PC7	Input	N/A	Temp Sensor
PE0	Output	Low	LCD Module DB0B
PE1	Output	Low	LCD Module DB1B
PE2	Output	Low	LCD Module DB2B
PE3	Output	Low	LCD Module DB3B
PE4	Output	Low	LCD Module DB4B
PE5	Output	Low	LCD Module DB5B
PE6	Output	Low	LCD Module DB6B
PE7	Output	Low	LCD Module DB7B
PF0	Output	Low	Water Pump In
PF1	Output	Low	Water Pump Out
PF4	Input	N/A	LCD Module A0B
PF5	Input	N/A	LCD Module A1B
PF6	Input	N/A	LCD Module A2B
PF7	Input	N/A	LCD Module A3B
PG2	Output	Low	A2D SCLK
PG3	Input	N/A	A2D Dout
PG5	Output	High	LCD Module Reset
PG6	Output	Low	LCD Module PE7

Table 2.1 I/O Pin reset use/status

Using the initialization routine found in the RCM3000.lib library provided with Rabbit Semiconductor's Dynamic C program, the routine has been modified to cater to our own needs; setting up our microcontroller to the status as shown in Table 2.1.

During startup and initialization, the RTC is updated with the current time as well. This is done via a query to the user for the current local time.

2.3 Organization of Embedded Code

Our software program is mostly polled loop driven, where it runs in an infinite loop after setup. Conditions are constantly being tested, some every cycle, some every second. Upon meeting certain conditions/criteria, certain pins would be asserted to handle the event. At the same time, the TCP socket is listening during every loop for a client program to request a connection to the server on the microcontroller. Should a connection be established, program control would be handed to a function that interacts with the client to transfer information.

3. Software Design Narrative

Our software is designed to consist of 4 main modules, the main module, the server module, the client module, and the local setup module. The main module is generally where the program will always be running at most times. The server and client modules work hand in hand with each other in communicating via the internet; allowing the user to control his/her aquarium from a distance. The local setup module controls the LCD display/keypad module to setup aquarium conditions locally.

3.1 Main Module:

The main module of our software program runs in an infinite loop, and handles does different things as certain conditions are met. It can be envisioned as a state machine, passing to different states given different inputs. The program checks input signals on every iteration of the loop for changes and the need to take action. Conditions checked are as follows:

- pH Sensors – should aquarium pH exceed user defined desired levels by more than 1, both water pumps would be activated to cycle fresh water into the aquarium, effectively lowering ammonium levels and bringing the water in the aquarium back to comfortable conditions for the fishes.
- Temperature Sensors – if the surrounding temperature brings the water to be 5 degrees lower than user desired levels, the water heater should be activated to bring the water temperate back up to desired temperatures. The water heater should heat the water to no more than 5 degrees more than desired levels, and will be shut off when this level is reached.
- Water Level Sensors – the optimum level for water to be at is exactly at the middle sensor. However this is often impossible to maintain so our bounds are the top and bottom sensors. When water levels drop below the low sensor, the pump in is activated and run for 10 seconds or so. This is sufficient time to bring the water back to safe levels. It would be the same for the situation when water levels rise above the high critical sensor, except the pump out is activated instead. Water level sensor functions supercede the pH sensors such that these will be asserted first before pH stabilizing functions. They also have the capability of shutting off pH stabilizing functions for the duration of the water balancing.
- Light and Feeder Timings – the main driving force behind the infinite loop is the real time clock (RTC) and each loop iteration will check for matching times with light on, light off, or feeding times. As the names imply, when light on occurs, the lighting pin is asserted and when light off occurs, the lighting pin is driven low. Feeding times assert the pin which drives a motor; this motor shakes fish feed into the aquarium, not unlike a saltshaker, from above.

The main module also checks for a TCP/IP connection every loop and executes the server module when a connection from the client is detected. At the same time, it executes the local setup module if input is received from the LCD display/keypad module.

3.2 Server Module:

The server module is the program segment that listens for a client to try to establish a connection with our device. It initializes the microcontroller TCP/IP settings to listen for a connection and when such a connection is established, waits for commands from the client. The server has three major functions;

1. Receiving new user defined desired conditions from the client and modifying memory on the microcontroller.
2. Sending current aquarium conditions to client.
3. Sending current user defined desired conditions to the client.

The server, upon receiving a selection will execute accordingly. When receiving data from the client to modify memory on the microcontroller, it writes directly to the global variables where settings are defined.

When a request for information is received from the client, two strings are written, in a predefined format, to temp variables and one string is sent as a packet to the client module depending on which set of information is desired by the client. The first string contains current aquarium conditions and the second contains current user desired conditions.

3.3 Client Module:

The client module serves as a user interface program for the user to access the device. It works over telnet (port 23) and on connection presents a menu to the user. The user is given the option to:

1. View current aquarium settings.
2. Modify desired aquarium settings.
3. View desired aquarium settings.
4. Quit connection.

As the client module works hand in hand with the server module, it works almost exactly opposite of the server. It sends out commands to the server to

send or receive information. When modifying aquarium settings, the user is presented with another set of menus on which option to modify. As mentioned in the server module analysis, the team has predefined methods for identifying data sent to and fro the server; each data item starts with a character from ‘A’ through ‘H’, which identifies which condition it is. After selection and input of data from the user, this data is sent and parsed by the server to modify the correct settings.

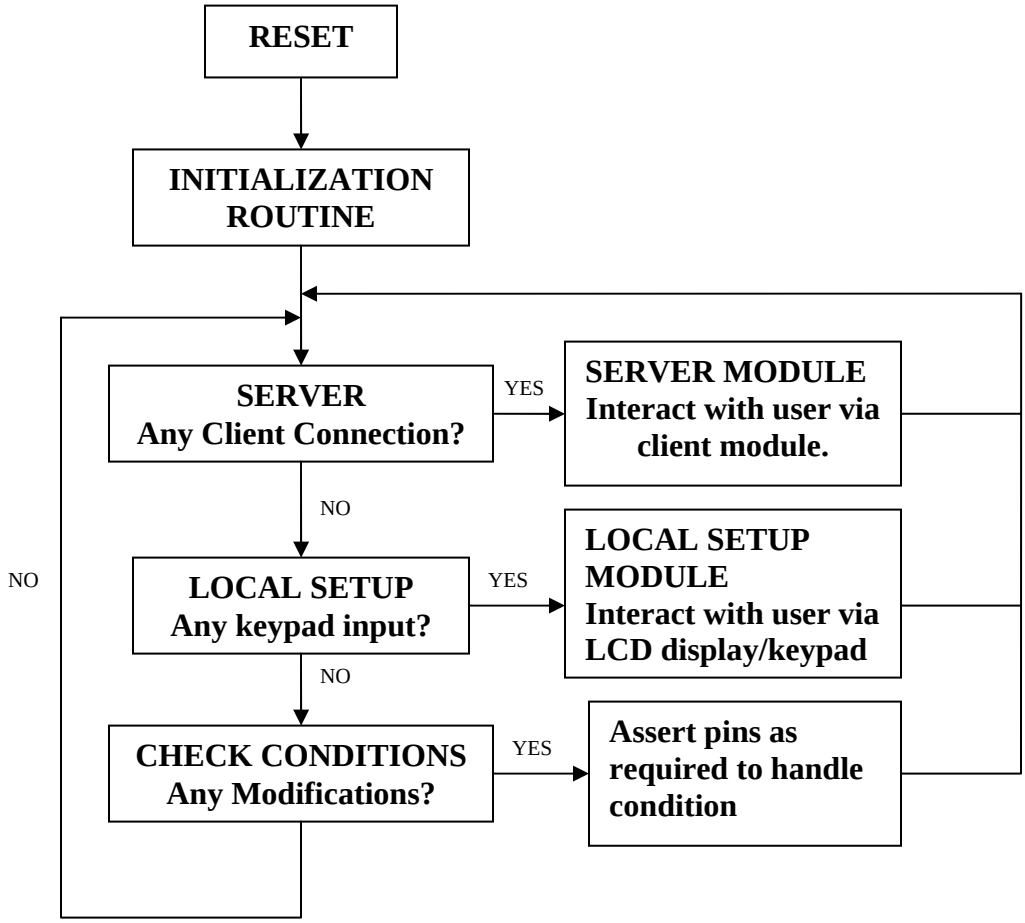
When sending a request for information to the server, the client readies itself to receive a packet of information. It then parses this packet of data received and presents the information to the user.

3.4 Local Setup Module:

This module executes when input is detected from the LCD display/keypad module. On execution, the module acts very much like the client module. It presents the user with a menu for modification of aquarium settings and allows the user to view current aquarium conditions or current desired aquarium conditions. It communicates with the LCD display/keypad module via the auxiliary I/O port and as this is the only time the LCD display/keypad module is used, it controls all inputs and outputs to and from the LCD display/keypad module.

4. Software Documentation

4.1 Program Flowcharts



4.2 Functions Listing

Main Module	
void print_time(unsigned long)	Prints current time given seconds

void level_check()	Water level check, water in aquarium is broken into a few regions from -2 to 4:
	`| 4`
	`critical sensor ----- 3`
	`| 2`
	`good sensor ----- 1`
	`| 0`
	`low sensor ----- -1`
	`| -2`

float read_pH()	Reads pH readings from the pH sensor LEDs. Data comes in serial.
float read_temp()	Reads the temperature from the temperature sensor. Data comes in serial.
void time_change()	Queries the user for present local time and updates RTC
void brdInit()	Board I/O pin initialization routine
void ledOut(int led, int value)	LED ON/OFF control on LCD display/keypad module
Server Module	
int receive_packet()	Receives information from client module, parses it and stores it in global variables
int send_packet()	Sends requested information to client module
void server()	Server initialization and port listener
Client Module	
int main(int argc, char **argv)	Main program. Sets up connection with server module and sends commands to server depending on selection
void Read_Port(int sockfd)	Receives current aquarium conditions from server and prints to screen
void Read_PortD(int sockfd)	Receives desired aquarium conditions from server and prints to screen
void Write_Port(int sockfd, char IPAddress[ ], int ServerPort)	UI for desired aquarium conditions modification. Sends selected condition to be modified and value to modify it to
Local Setup Module	
void receive_input()	Receives LCD display/keypad input and modifies global variables
void print_screen(char* string)	Prints input string to LCD display

4.3 References

A few of our functions were adapted off of code from Rabbit Semiconductor's sample demo code that is packaged with their Dynamic C program.

Main Module

- Uses modified version of brdInit() function found in RCM3000.lib

Server Module

- Echo_Server.c
  http://www.rabbitsemiconductor.com/documentation/SamplesRoadmap/roadmap.htm#1006382

Local Setup Module

- KEYPADTOLED.c
  http://www.rabbitsemiconductor.com/documentation/SamplesRoadmap/roadmap.htm#1006405