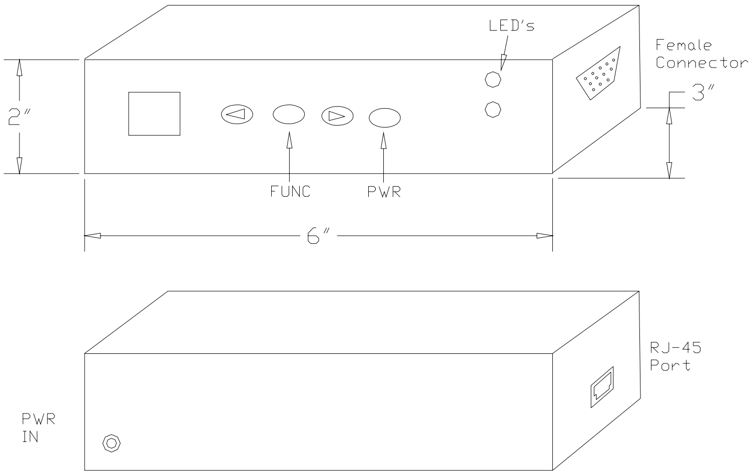
Homework 10
Software Design Considerations
Group Number: 12
Phillip Boone

Introduction

The “Digital Picture Box” functionally is used to display to a VGA monitor a database of pictures stored remotely on a personal computer via the Internet. The status LED is used to indicate the device has power, and also indicates when a picture is being transferred. The push buttons are used to cycle through the pictures stored on a remote PC. The remote control will have the same functionality as the push buttons. The picture box is designed to connect directly to any standard VGA monitor or LCD screen. The box must have a custom TCP/IP server running with it’s own custom image transfer protocol in order for the data exchange with the PC to work.

The “Digital Picture Box” design basically consists of a small cubicle casing that has push buttons and a status LED on the front. On the right side of the casing there is a VGA connector, on the left side there is a RJ-45 network connector, and on the rear there is a power connector.



The “Digital Picture Box” requires two major software projects: the Rabbit 3000 TCP/IP server and the PC picture client. From a network perspective, the Rabbit is a server because the Rabbit has no way of inputting an IP address with its limited push buttons and IR input. However, the PC client acts as a file server because all of the picture files are stored on the PC’s hard drive. I plan to use a custom TCP/IP protocol for communication that is described later.

The Rabbit 3000 micro controller must be able to listen on the RJ-45 port for any incoming connections. Once the client PC program connects, the Rabbit will control what file to transfer as well as when to transfer it. It will have to forward the data to the Epson graphics controller in a timely fashion. I plan to setup a simple multitasking environment that I explain in the “Software Design Narrative” section, which consists of several modules. The first module “incoming_tcp” will monitor the TCP/IP connection, and if active dump all data to a large circular buffer. The second module "decodePCdata" will decode the instruction stored in the circular buffer, and then execute the corresponding data-transfer command on the Epson graphics controller. The third module "monitor_buttons" debounces the input buttons. It sends the corresponding command via TCP/IP to the PC if a different image needs to be fetched, or updates text overlay if in text-overlay mode. The fourth module "monitor_IR" performs the same function as "monitor_buttons," but receives the command from the IR remote instead of push buttons. Finally, the last module "overlay_mode" will execute a series of overlay setup menus stored on the Rabbit.

The PC client software is responsible for converting the picture data to a useable format for the Epson graphics controller, sending the picture data in order, compiling a picture library, and keeping track of the current displayed picture. It will constantly monitor any incoming rabbit control statements, and perform the corresponding command. It either needs to execute commands that transfer a picture or update the pointer to the current picture.

Software Design Considerations

The Rabbit 3000 micro controller has several general design considerations. Since the controller comes with a Dynamic C compiler, it automatically handles the memory mapping and management at compile time. The compiler also sets up the necessary startup code so the downloaded program will automatically execute on power up. There is also support for a simple multitasking environment included with the compiler. The current proposed implementation for sending address information to the Epson controller is somewhat inefficient. Transferring a single picture could take a fair amount of time. It was decided that in order to create a responsive software interface, we should take advantage of the built in multitasking functions to easily interrupt a picture transfer in progress. There are three basic C functions that implement a cooperative multitasking environment. First, a "costatement" designates a piece of code to a single running process. Within that process, it has the option to call the "yield" function. This gives up control to the next process. When the process that called yield obtains control again, it returns to the line of code after the yield function. The "abort" function is very similar to yield; however, when the process regains control it starts execution at the beginning of the costatement block.

Software Design Narrative

In order for the PC program module and Rabbit module to successfully communicate together, they must obey some type of protocol. It was decided to implement a custom TCP/IP Picture Frame Protocol. The packet data that is sent from the Rabbit to the PC file server is just plain text ending in a null terminating character. The Rabbit only needs to send four commands to the PC: "nextpic," "prevpic," "getpicname," and "recv." "Nextpic" tells the PC to change it's current file pointer to the next available picture file. "Prevpic" tells the PC to change it's current file pointer to the previous picture file. "Getpicname" requests the name of the current picture file. "Recv" tells the file server to begin sending the current picture file. The packet data that is sent from the PC to the Rabbit requires the use of two length fields. The Rabbit dumps all of the incoming data into a temporary buffer. A total length field that is placed at the beginning of each packet is necessary to distinguish between consecutive packets dumped into the buffer. Once a single packet is extracted from the buffer, it is necessary to separate the included command from binary data. The second part of each packet is a command length integer, which is the number of bytes the included command occupies. There are also four commands sent to the Rabbit: "sendpicname," "sendnewpic," "sendnextpixel," and "sendpixeladdress." "Sendpicname" includes a text string that is the filename, as it is stored on the PC. "Sendnewpic" resets the current address pointer to the frame buffer on the Epson graphics controller to the first address, and sends the first pixel's color data. "Sendnextpixel" increments the frame buffer address pointer, and sends the corresponding color data for that pixel. "Sendpixeladdress" sets the frame buffer address pointer to an absolute address, and then sends the color data for that pixel.

Summary of TCP/IP Communication Protocol

Incoming Rabbit packets	**Incoming PC packets**
* Packets can include Commands and Data	* Null terminated commands
* Total length used to distinguish between consecutive packets stored in the Rabbit's temporary buffer	* Commands Include: getpicname, nextpic, prevpic, and recv
* Command length used to separate the Command from Data	
* Commands include: sendpicname, sendnewpic, sendnextpixel, and sendpixeladdress	

The Rabbit program consists of several programming modules, which are actually costatements that implement the cooperative multitasking environment described earlier.

The first costatement constantly runs the tcp_tick command. This is done to perform all the necessary low-level TCP/IP bookkeeping.

The second costatement “incoming_tcp” starts listening on port 39 for any TCP/IP connections. Once it detects a connection, it takes all incoming data, and drops it into a temporary circular buffer called “indata.” If “indata” is full, it yields control until there is enough space to put the data into the circular buffer. This is repeated while the PC is connected. Once the PC disconnects, the Rabbit goes back to listening on the network port.

The third costatement “monitorIR” polls the IR input pins two samples at a time. When an IR remote button is pressed, the logic value on the output of the IR pin changes. This can be detected when two consecutive samples are opposite logic values, indicating a recent logic change. Whenever a button press is detected, the Rabbit sends the corresponding command to the PC if a network connection is present.

The fourth costatement “monitor_buttons” is the same as “monitorIR,” except it debounces the switch inputs and sends the corresponding command to the PC if there is a network connection. The switches are debounced by taking three samples of each input pin. There is a delay of 50 ms inserted in between each sample via the “DelayMs” function that automatically configures a timer interrupt. Once three samples agree, the logic value of the samples is compared with the previously obtained agreeing samples. If a logic value zero was followed by a logic value one, indicating a rising edge, then a button press was successfully detected.

The fifth costatement “decodePCdata” consumes data contained in the circular buffer one command at a time. Each decoded command is stored in a temporary location. The command length is read in, and then the entire command is read in if possible. If it is not possible to decode an entire command, control is yielded to the next process. Once the command is decoded, the proper address is sent to the address bus of the PLD’s that interface with the Epson graphics controller. The address is clocked in 8 bits at a time for three clock cycles until the entire address is present. The 15 bit color data is then sent out on the data bus and the Rabbit generates the necessary signals to load the current pixel into the frame buffer of the Epson graphics controller.

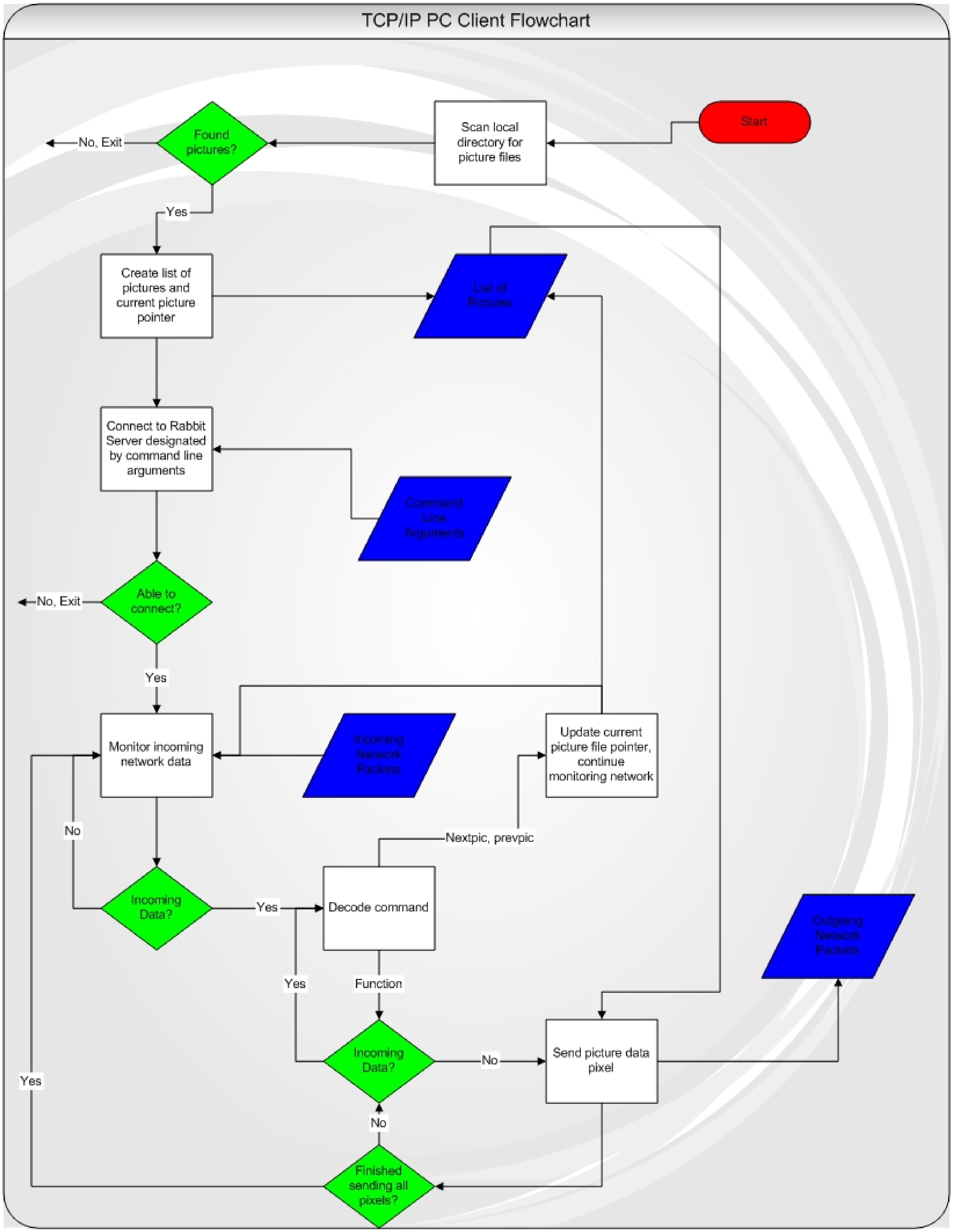
The last costatement “overlay_mode” is only called when the “Function” button is pressed. This will bring up a menu system that is overlaid on the VGA display. A set of fonts is stored into the flash memory of the Rabbit. A basic function "print_character" will output the character stored in the font data at a specific point on the screen. The function will require an address argument to specify the point location and a character argument. This basic function is used to display the entire character menu. The remote controller and push buttons now are polled in the same fashion as described in the earlier functions, but they now are used to navigate through the menu. The next and previous buttons now cycle through different menu options. The function button selects the highlighted menu option. The Power button exits out of the overlay menu system.

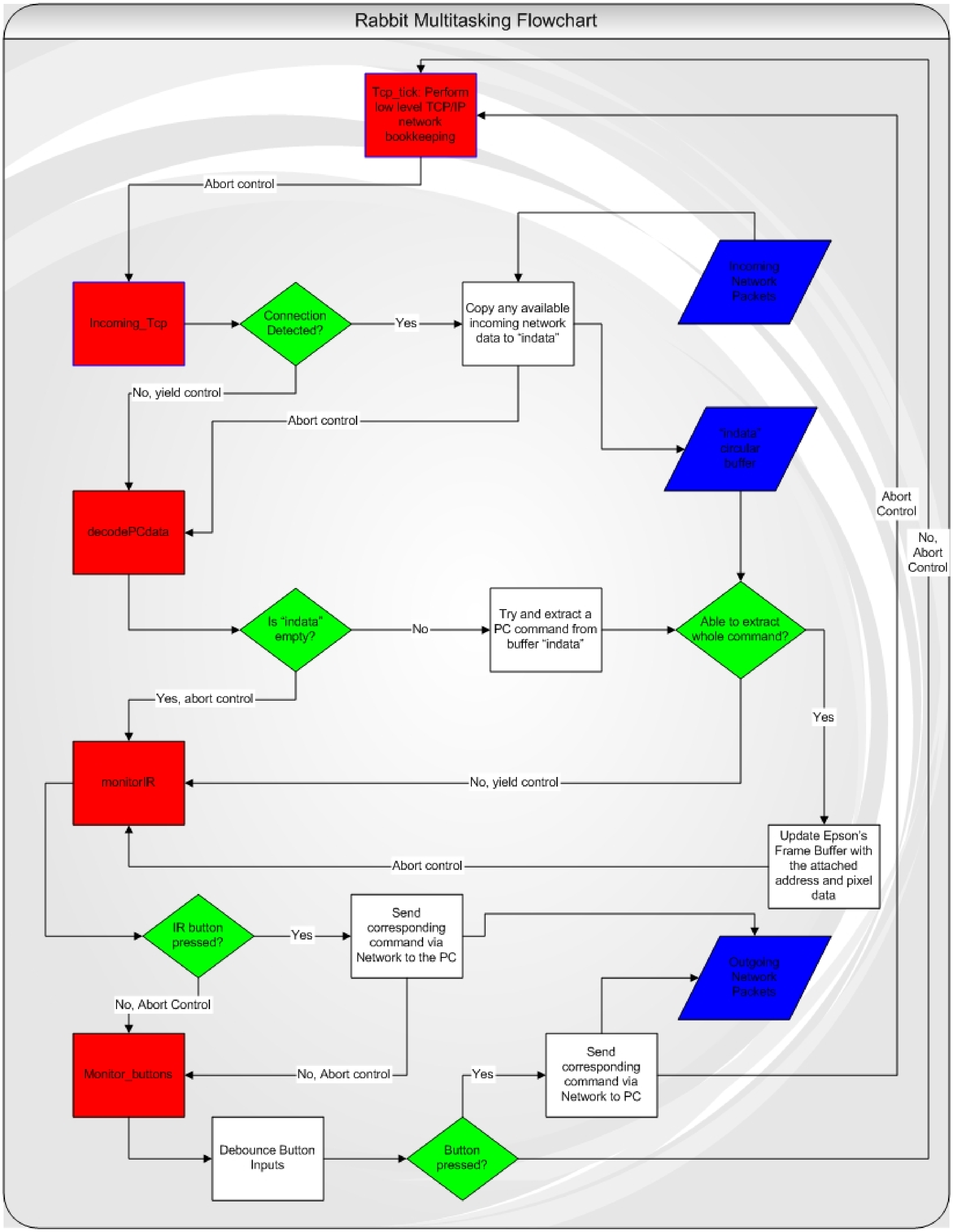
Summary of Rabbit Programming Modules

Programming Module	**Description**
tcp_tick	Constantly called to perform all necessary low-level network bookkeeping.
incoming_tcp	Listens to the network port. When connected, it drops all incoming data to a circular buffer if possible.
monitorIR	Monitors the infrared port. If a command is detected, the corresponding command is sent via TCP/IP to the PC.
monitor_buttons	Monitors the button input. If a command is detected, the corresponding command is sent via TCP/IP to the PC.
decodePCdata	Decodes the data stored in the circular buffer. The pixel data at the corresponding address is sent to the Epson controller.
overlay_mode	A basic menu system is displayed. All input buttons are now directed to this function while in the menu system.

The final major programming module is the TCP/IP client that can run on any standard Linux PC. It is a command line program that takes two arguments: IP address of the Rabbit, and port number of the Rabbit server. The IP address argument can actually be a domain name that will automatically be resolved on execution. The program begins by scanning the local directory for picture files in the .ppm format. It has a built in picture pointer that starts off pointing to the first picture in the directory. It then tries to connect to the IP address and port combination passed via the command line. When connected, it constantly tries to read in command data from the Rabbit. Once the Rabbit server sends it a command, it executes the corresponding command and then returns to reading commands. The "nextpic" command simply changes the picture pointer to the next available picture file. The "prevpic" command changes the picture pointer to the previous available picture file. The "getpicname" command from the Rabbit causes the PC to respond by sending a "sendpicname" command back to the Rabbit with the name of the file pointed to by the picture pointer as the data field. When the PC receives the "recv" command, it must proceed to send the entire picture in order to the Rabbit. As it sends picture data, it checks to see if another command has been received. Other commands can interrupt this command and take over execution. This is done so the user does not have to wait for an entire picture to display to cycle through to a different picture. In order to send a picture, the current picture file is broken down into pixels. Each pixel is converted from 24 bits per pixel to 15 bits per pixel because the Epson can only support a maximum of 15 bits per pixel without using a color table. First, the "sendnewpic" command is sent to the Rabbit that resets the Epson's address to the first address in the frame buffer and loads the first pixel color data. Every picture thereafter uses the "sendnextpixel" command to implicitly increment the address pointer as well as load in the corresponding pixel color data.

TCP/IP PC Client Flowchart


Start
Scan local directory for picture files
Found pictures?
No, Exit
Yes
Create list of pictures and current picture pointer
List of Pictures
Connect to Rabbit Server designated by command line arguments
Command Line Arguments
Able to connect?
No, Exit
Yes
Monitor incoming network data
Incoming Network Packets
Update current picture file pointer, continue monitoring network
No
Incoming Data?
Yes
Decode command
Nextpic, prevpic
Yes
Function
Outgoing Network Packets
Incoming Data?
No
Send picture data pixel
Yes
No
Finished sending all pixels?


Rabbit Multitasking Flowchart
Tcp_tick: Perform low level TCP/IP network bookkeeping
Abort control
Incoming_Tcp
Connection Detected?
Yes
Copy any available incoming network data to "indata"
Incoming Network Packets
No, yield control
Abort control
"indata" circular buffer
Abort Control
No, Abort Control
decodePCdata
Is "indata" empty?
No
Try and extract a PC command from buffer "indata"
Able to extract whole command?
Yes, abort control
Yes
monitorIR
No, yield control
Update Epson's Frame Buffer with the attached address and pixel data
Abort control
IR button pressed?
Yes
Send corresponding command via Network to the PC
Outgoing Network Packets
No, Abort Control
Monitor_buttons
No, Abort control
Yes
Send corresponding command via Network to PC
Debounce Button Inputs
Button pressed?

References

1) Richard Stevens. Unix Network Programming – Networking APIs: Sockets and XTI. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

2) Dynamic C – TCP/IP User's Manual. Davis, CA: Z-World, Inc., 2002.