Arduino Project

CSE1010 Project 9, Fall 2012

Jimmy Boivie

Liam Dickson

December 9, 2012

TA: Levon Nazaryan

Section: 9

Instructor: Jeffrey A. Meunier

1. Introduction

We discussed already in lecture how a computer stores all its information in binary. All that a computer knows is made of binary values, 1s and 0s, or more correctly, electrical signals that are on or off. To send data from one computer to another, it is simplest to send the binary data between the computers, since it's already in that form. There are many ways to do this: as black and white bar codes printed on a paper (but those are not actually pure binary), as electrical signals sent on a wire, as magnetic spots oriented on a disk, as radio signals sent through the air, or as flashes of light.  In this assignment you will use your Arduino board as the driving circuit for a communications channel that uses light to communicate with another computer.

2. Inputs & Outputs

First the system must be set up, by calling the arduinoOpen function, the globals function and the autoCalibrate function. The first function allows Arduinos to be used. The second defines the global variables. The third function defines what is “on light” verses “off light”

In the script file, one could replace the current ‘abc’ with any arbitrary message to be sent. First the program converts each character in the string to an 8-bit character. Then a parity bit is added to each 8-bit set. The program then attempts to send the 9-bits. A zero corresponds to a LO\_DURATION (.05 seconds) of time, and a one corresponds to a HI\_DURATION (.15 seconds) of time. The program sends a zero by lighting the led for the .05 seconds and sends a zero by lighting the led for .15 seconds. In between send characters, the sending Arduino pauses for a SPACE\_DURATION of .05 seconds works well. In other words, the sending Arduino waits .05 seconds before sending its next character.

For the receiving end the light is determined by on if and only if the light reading is greater than the average of the “on light” and the “off light”. Otherwise, the light is deemed to be off. For the receiving end to determine whether it received a zero or one it looks at the time. If the observed time is greater than the average of the HI\_DURATION and LO\_DURATION than the receiving Arduino perceives a 1. Otherwise it perceives a 0.

The program then checks to see if the received bits makes sense as discussed in the next section. Eventually it will or it may make a mistake. Regardless it will repeat the process but now the sender is now the receiver and the receiver is now the sender. Once the character string is sent both ways, it is displayed as well as time for the round trip. The time of the round trip is time the program took to send the character string back and forth.

3. Program Modifications

To increase the reliability of the system, a parity bit was added to the program. So for each 8 bit character, the sending Arduino would send 9 bits, where the 9th bit was parity. The receiving Arduino would expect to receive 9 bits. If the receiving Arduino were to receive less than 9 bits it would time out and send NAK which would imply to send the bits again. If the receiving Arduino were to receive the correct number of bits, it would calculate a new parity for the first 8 bits and compare it to the receive parity. If they are different, the receiving Arduino would send NAK back to the sending Arduino. If the calculated parity and the received parity are the same, the receiving Arduino sends an ACK which would imply that it received the 8-bit character correctly and is ready for the next character. On the sender’s end, it will time out if it does not receive an ACK or NAK. If this were to happen, the sending Arduino would just resend the character it just sent.

4. Experimental Data

We made quite a few test runs. In order to induce flaws in the program we tried reducing the space time. The space time is the time between the light turning off and back on again. The lower the space time, the greater chance an error occurs. Usually when the space time is approximately .02 seconds, the receiving Arduino has trouble receiving the characters it is supposed to be receiving. However, after sometime, it usually receives the correct string, although it has made mistakes, which occur when the more than 1 bit is flipped so the parity is technically correct although the corresponding character is incorrect. We also experiment with bring both durations of light down. Similar results were noticed. We also changed the distance between the Arduinos. Generally the issue with this was that Arduinos were too far to really get close to receiving the correct data from either. We also tried blocking the path of light with a notecard. This was done to see that the receiving Arduino would keep asking for the transmission again even though it received nothing. Simultaneously, the sending Arduino would still attempt to send the same character if it got no ACK or NAK. And finally when the note card was removed, the data transmission worked and received the correct string.

While trying to optimize our times without parity we recorded this data:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Test 1 | Test 2 | Test 3 | Test 4 | Test 5 | Test 6 | Test 7 |
| HI\_DURATION (s) | 0.667 | 0.667 | 0.2 | 0.1 | 0.15 | 0.15 | 0.15 |
| LO\_DURATION (s) | 0.25 | 0.25 | 0.1 | 0.05 | 0.08 | 0.05 | 0.05 |
| SPACE\_DURATION (s) | 0.2 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.05 |
| Pass/Fail | P | P | P | F | P/F | P | P |

5. Source Code:

5.1 append Parity

function b = appendParity(b,parity\_bit)

% Given a vector of numbers and a parity value (0 or 1) this function will

% add a value to the end of the vector to make the vector match the parity

% Use: appendParity(b,parity\_bit)

b(end+1) = parityOf(b,parity\_bit);

end

5.2 arduinoOpen

function serialPort = arduinoOpen(portName)

% FUNCTION: open a serial port connection to an Arduino board

% USE: serialPort = arduinoOpen(portName)

% PARAMS: portName = the name of the port to which the Arduino board is connected

% RETURNS: an open serial port object

% EXAMPLES: serialPort = arduinoOpen('COM4');

% serialPort = arduinoOpen('/dev/tty.usbmodem411');

serialPort = serial(portName, 'BaudRate', 115200);

fopen(serialPort);

% it takes a few seconds before the port can be used, but in the mean

% time the board can be reset if this is a necessary step on your system

disp('reset the board if necessary')

for i=10:-1:1

fprintf('%d ', i);

pause(0.3);

end

fprintf('\n')

end

5.3 autoCalibrate

function [lo hi] = autoCalibrate(arduino)

global LO\_LEVEL

global HI\_LEVEL

lo = readLightLevel(arduino);

hi = readLightLevel(arduino);

% state 1: look for low and high levels

iters = 0;

% Flash the LED at random intervals and wait for the adjacent board

% to do the same thing. This will allow us to synchronize the flashing.

while true

diff = hi - lo;

if diff > 30

break

end

if iters > 20

error('Adjacent LED not detected')

end

iters = iters + 1;

led5(arduino, 9)

pause(0.1)

led5(arduino, 0)

n = 5 + floor(10\*rand);

for i=1:n

level = readLightLevel(arduino);

if level > hi

hi = level;

elseif level < lo

lo = level;

end

pause(0.01)

end % for i

end % while

% Adjacent board detected. Commence synchronized flashing, and determine

% the mean light levels for on and off.

threshold = lo + (hi - lo)/2;

loSum = 0;

loCount = 0;

hiSum = 0;

hiCount = 0;

state = false;

for i = 1:10

led5(arduino, 9)

pause(0.1)

led5(arduino, 0)

tic

while true

level = readLightLevel(arduino);

if state == false && level > threshold

hiSum = hiSum + level;

hiCount = hiCount + 1;

state = ~state;

break

elseif state == true && level < threshold

loSum = loSum + level;

loCount = loCount + 1;

state = ~state;

break

end % if

if toc > 1.0

lo = loSum / loCount;

hi = hiSum / hiCount;

return

end % if

end % while

end % for i

LO\_LEVEL = lo;

HI\_LEVEL = hi;

end % function

5.4 bin2char

function c = bin2char(b)

% Given a vector of binary numbers this will convert the vector to the

% corresponding ASCII character defined by them.

% Use: bin2char(b)

b\_str = char(b+48);

c = char(bin2dec(b\_str));

end

5.5 char2bin

function b = char2bin(c)

% Given a character this will convert the vector to the

% corresponding ASCII number and then the binary number that it corresponds to.

% Use: char2bin(c)

bin\_str = dec2bin(c);

bin\_str\_long = sprintf('%08s',bin\_str);

b\_vec = bin\_str\_long - 48;

b = logical(b\_vec);

end

5.6 checkParity

function result = checkParity(bits, parity)

% This function takes a vector of bits and if the actual parity of the bits

% is equal to desired parity of the bits then a 1 is returned. If the

% vector is not 9 bits in length or the parities are not equal then a 0 is

% returned

% Use: checkParity(bits, parity)

if length(bits) ~= 9

result = 0;

return

end

if bits(9) == parityOf(bits(1:8), parity)

result = 1;

else

result = 0;

end

5.7 globals

global HI\_LEVEL

global LO\_LEVEL

global HI\_DURATION

global LO\_DURATION

global SPACE\_DURATION

HI\_LEVEL = 0;

LO\_LEVEL = 0;

HI\_DURATION = 0.150;

LO\_DURATION = 0.050;

SPACE\_DURATION = 0.050;

5.8 led5

function led5(a,value)

value = num2str(value);

fwrite(a, value);

end

5.9 led13

function led13(a,logical\_value)

if logical(logical\_value)

fwrite(a, 't');

else

fwrite(a, 'f');

end

end

5.10 parityOf

function parity\_bit = parityOf(b,desired\_par)

% This function determines the parity of a vector and if it is equal to the

% desired parity then it sets the 'parity\_bit' equal to 0, otherwise it

% sets it eqaul to 1.

% Use: parityOf(b,desired\_par)

if rem(length(find(b)),2) == desired\_par

parity\_bit = 0;

else

parity\_bit = 1;

end

end

5.11 readLightLevel

function result = readLightLevel(a)

% This function determines reads the light level of the photoresister and

% outputs it as a single value.

% Use: readLightLevel(a)

fwrite(a, 'r');

var = fread(a, 2);

result = var(1)\*256 + var(2);

end

5.12 receiveBit

function bit = receiveBit(a)

% This function will wait until a bit is received and then output the bit

% Use: receiveBit(a)

global LO\_DURATION

global HI\_DURATION

global LO\_LEVEL

global HI\_LEVEL

avgLevel = (HI\_LEVEL + LO\_LEVEL)/2;

avgDuration = (HI\_DURATION + LO\_DURATION)/2;

while true

lightLevel = readLightLevel(a);

if lightLevel > avgLevel

tic

break

end

end

while true

lightLevel = readLightLevel(a);

if lightLevel < avgLevel

times = toc;

break

end

end

if times > avgDuration

bit = 1;

else

bit = 0;

end

end

5.13 receiveBitParity

function [bit status] = receiveBitParity(a)

% This function will wait until a bit is received and then output the bit

% if the program times out then return a status of 1 which indicates a

% timeout

% Use: receiveBit(a)

global LO\_DURATION

global HI\_DURATION

global LO\_LEVEL

global HI\_LEVEL

avgLevel = (HI\_LEVEL + LO\_LEVEL)/2;

avgDuration = (HI\_DURATION + LO\_DURATION)/2;

timeout = tic; % Starts timer for timeout

status = 0;

while true

lightLevel = readLightLevel(a);

if lightLevel > avgLevel

t = tic;

break

end

if toc(timeout) > 2 % If the timer reaches x seconds then set the status to 1 and bit equal to 2

status = 1;

bit = 0; % Set bit equal to 0 (so that there is a return)

return

end

end

while true

lightLevel = readLightLevel(a);

if lightLevel < avgLevel

times = toc(t);

break

end

end

if times > avgDuration

bit = 1;

else

bit = 0;

end

end

5.14 recvChar

function c = recvChar(a)

% This waits for a string of 8 bits to be recieved using receiveBit and

% outputs a character by converting the binary values using bin2char.

% Use: recvChar(a)

for i = 1:8

binVector(i) = receiveBit(a);

fprintf('%g',binVector(i))

end

fprintf('\n')

c = bin2char(binVector);

fprintf('recvChar got char %c \n',c)

end

5.15 recvCharParity

function c = recvCharParity(a)

% This waits for a string of 8 bits to be recieved using receiveBit and

% outputs a character by converting the binary values using bin2char. It

% uses receiveBitParity to determine whether or not there was a timeout

% error. If there was a timeout then it changes the binaryVector to

% 'timeout' which causes the function to repeat itself after sending a false code to the other

% arduino. If the parity check is

% Use: recvCharParity(a)

for i = 1:9

[binVector(i) status] = receiveBitParity(a);

if status == 1

binVector = ['timeout'];

break

end

fprintf('%g',binVector(i))

end

fprintf('\n')

if ischar(binVector) == 1

fprintf('recvChar timed out: ACK \n')

sendBit(a,0)

c = recvCharParity(a);

else

c = bin2char(binVector(1:8));

fprintf('recvChar got char %c \n',c)

check = checkParity(binVector,0);

if check == 1

sendBitACK(a)

else

sendBit(a,0)

c = recvCharParity(a);

end

end

end

5.16 recvString

function charString = recvString(a)

% This function waits to receive a string by using the first character

% received to determine the length of the rest of the string.

% Use: recvString(a)

char1 = recvChar(a);

for i = 1:double(char1)

charString(i) = recvChar(a);

end

fprintf('recvString got string ')

disp(charString)

end

5.17 recvStringParity

function charString = recvStringParity(a)

% This function works exactly like recvString except that it uses

% recvCharParity instead of recvChar.

% Use: recvStringParity(a)

char1 = recvCharParity(a);

for i = 1:double(char1)

charString(i) = recvCharParity(a);

end

fprintf('recvString got string ')

disp(charString)

end

5.18 sendBit

function sendBit(a,bit)

% This function sends a bit 0 or 1 by leaving the led on for a length of

% LO\_DURATION and HI\_DURATION respectively.

% Use: sendBit(a,bit)

global LO\_DURATION

global HI\_DURATION

global SPACE\_DURATION

led5(a, 9);

switch bit

case 0

pause(LO\_DURATION);

case 1

pause(HI\_DURATION);

end

led5(a, 0);

pause(SPACE\_DURATION);

end

5.19 sendBitACK

function sendBitACK(a)

% This function sends a bit length 2\*HI\_DURATION to ensure that the

% acknowledgement gets across (The other end is looking for a length of at

% least (HI\_DURATION+LO\_DURATION)/2

% Use: sendBitACK(a)

global HI\_DURATION

global SPACE\_DURATION

led5(a, 9);

pause(2\*HI\_DURATION);

led5(a, 0);

pause(SPACE\_DURATION);

end

5.20 sendChar

function c = sendChar(a,c)

% This function sends a character one bit at a time using sendBit.

% Use: sendChar(a,c)

fprintf('sendChar sending char %c \n',c)

binVector = char2bin(c);

for i = 1:length(binVector)

sendBit(a,binVector(i));

fprintf('%g',binVector(i))

end

fprintf('\n')

end

5.21 sendCharParity

function c = sendCharParity(a,c)

% This function sends a character one bit at a time with an appended parity

% bit. It also stops after the send to check for an ACK. If it receives a

% NAK or times out then it resends the character.

fprintf('sendChar sending char %c \n',c)

binVector = char2bin(c);

binVector = appendParity(binVector,0);

for i = 1:length(binVector)

sendBit(a,binVector(i));

fprintf('%g',binVector(i))

end

fprintf('\n')

[check status] = receiveBitParity(a);

if check == 0 || status == 1

c = sendCharParity(a,c);

end

end

5.22 sendString

function s = sendString(a,s)

% This function sends a string one character at a time using the sendChar

% function.

% Use: sendString(a,s)

fprintf('sendString sending string ')

disp(s)

s = [char(length(s)) s];

for i = 1:length(s)

sendChar(a,s(i));

end

fprintf('sent \n')

end

5.23 sendStringParity

function s = sendStringParity(a,s)

% This function sends a string one character at a time using the

% sendCharParity function.

% Use: sendStringParity(a,s)

fprintf('sendString sending string ')

disp(s)

s = [char(length(s)) s];

for i = 1:length(s)

sendCharParity(a,s(i));

end

fprintf('sent \n')

end

5.24 testArduino

function testArduino(portName)

a = arduinoOpen(portName);

tic

while toc < 10 % 10 seconds

inLevel = readLightLevel(a);

inPercent = inLevel / 512;

outPercent = 1-inPercent;

outLevel = outPercent \* 10;

led5(a, outLevel);

end

led5(a, 0)

fclose(a);

end

5.25 timer1

t = tic;

sendString(a,'abc')

recvString(a);

roundTripTime = toc(t)

5.26 timer1Parity

t = tic;

sendStringParity(a,'abc');

recvStringParity(a);

roundTripTime = toc(t)

5.27 timer2

s = recvString(a);

sendString(a,s)

5.28 timer2Parity

s = recvStringParity(a);

sendStringParity(a,s);