Some of my reports for Hardware and Software Projects

Sasank Chilamkurthy - 110070051

October 21, 2014

You may find reports of some of my projects appended to this document. Since appending code to this pdf is cumbersome, you may view my code on GitHub. This is my profile: `https://github.com/chsasank`. Please take a moment to go through it. These projects are in reverse chronological order:

1. Decoding of LDPC codes using sum-product/message-passing algorithm.

   Code available here: `https://github.com/chsasank/LDPC-Codes`
2. Lempel-Zev-Welch compression for files

   Code available here: `https://github.com/chsasank/Compression`
3. Implementation of pipelined ARM 7 processor in Verilog

   Code available here: `https://github.com/chsasank/ARM7`
4. Wireless communication between micro-controllers using Amplitude Shift Keying
5. Sensing of light using LEDs and displaying pattern drawn on LCD array on GLCD aka '*e*slate'. Done on CPLD (think of it as cheap FPGA) in Verilog.

Apart from these, I'm doing a project in image processing this semester : Traffic Sign Classification using Random Forests and LDA on Histogram of Oriented Gradients descriptors. I'm attaching proposal of this at the end.

LDPC codes: Sum Product Algorithm

Sasank Chilamkurthy, Tharun Kumar Reddy, Varun Bairaboina

25 April 2014

1 Introduction

Low Density Parity Check codes are capacity achieving codes proposed with very large parity check matrix. But we choose parity check matrix to be sparse, which helps in decoding. In general, decoding of a code amounts to finding hamming distance between received binary vector and all codevectors and return the codeword with least distance. Since we are doing block coding, we have 2^n codewords to search, where n is length of code-word/vector.

Our method of decoding, though not optimal, uses graph structure of parity check matrix and pass messages/beliefs on the graph. This algorithm is called sum-product algorithm on a factor graph or belief propogation.

In this report we'll not give proof why our algorithm works but we'll give semantics of the algorithm as there are many papers/books dealing with this. Our reference was D McKay, Information Theory, Inference, and Learning Algorithms

2 How is decoding done?

What we have: A large parity check matrix H. Received binary vector $\mathbf{x}$, which is $\mathbb{F}_2$ sum (same as bitwise xor) of a code word $\mathbf{c}$ and a binary noise vector $\mathbf{n}$. i.e. $\mathbf{x} = \mathbf{c} + \mathbf{n}$. All code words satisfy $H\mathbf{x} = 0$.

What we want to find: Most probable noise vector $\hat{\mathbf{n}}$ that explains received vector $\mathbf{x}$. Then we can recover original code word by subtracting (same as bitwise xor for $\mathbb{F}_2$) $\hat{\mathbf{n}}$ from $\mathbf{x}$. Define syndrome as $\mathbf{z} = H\mathbf{x}$

$$z = Hx = H(c+n) = Hc + Hn = Hn$$

Thus, our noise estimate $\mathbf{n}$ has to satisfy $\mathbf{z} = H\mathbf{n}$. So, our problem is to find most probable $\hat{\mathbf{n}}$ which satisfy $H\hat{\mathbf{n}} = \mathbf{z}$ where

$$\begin{aligned} P(\mathbf{n}, \mathbf{z}) &= P(\mathbf{n})\mathbf{1}[\mathbf{z} = H\mathbf{n}] \\ &= \prod_n P(n_n) \prod_m \mathbf{1}[z_m = \sum_{n \in \mathcal{N}(m)} n_n] \end{aligned}$$

where $\mathcal{N}(m)$ denotes set of indices with non-zero elements in mth row. We also

introduce notation $\mathcal{M}(n)$ which denotes set of indices with non-zero elements in nth coloumn.
Since we want to do bit wise decoding, we want to marginalise above function to find $P(n_i = 0)$ for each i, so as to declare ith bit to be 0 if this is greater than 0.5 and 1 otherwise.

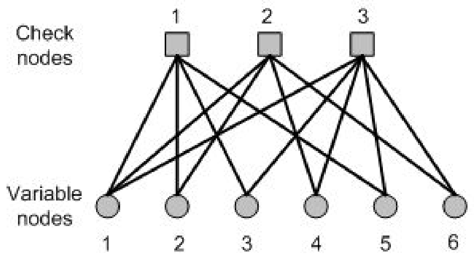
3 Graph Corresponding to a Code

Parity check matrix H defines a unique bi-partite graph as follows:
Let H be $m \times n$ matrix. This means code vector is of length n and there are m parity checks. mth parity check checks the parity of bits from set $\mathcal{N}(m)$. Similarly nth bit occurs in parity check equations indexed by set $\mathcal{M}(n)$. This allows to define a bipartite graph with bits and parity checks as nodes. There is a edge between *nth bit node* and *mth check node* bit n participates in mth parity check equation. This graph is called Tanner Graph.

Let's illustrate this with a (6,3) linear block code with the following parity check matrix:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Corresponding Tanner Graph is



4 Decoding using sum product algorithm

notation: $\mathbf{r_{mn}}$ is the message passed from check nodes to bit nodes and $\mathbf{q_{mn}}$ is the message passed from bit nodes to check nodes. These are meant to be beliefs held by check nodes and bit nodes respectively about noise vector $\hat{\mathbf{n}}$ and are vectors of length 2.

Initialization:

Initialize for every (n, m), $\mathbf{q_{nm}} = [1 - f, f]$ where f is probability of error. This is initial belief held by bit nodes about nth bit of noise vector.

Horizontal step:

In Horizontal step (horizontal from the point of view of H), we compute $\mathbf{r_{mn}}$ which is belief held by mth check node about nth bit node. It is calculated by considering beliefs of all bit nodes received by mth check node except nth bit node.

$$\mathbf{r_{mn}}[0] = \sum_{x'_n : n' \in \mathcal{N}(m) \backslash n} \mathbf{1}[z_m = \sum x'_n] \prod_{n' \in \mathcal{N}(m) \backslash n} q_{mn'}[x_{n'}]$$
$$\mathbf{r_{mn}}[1] = 1 - \mathbf{r_{mn}}[0]$$

Vertical step:

In this step, we compute $\mathbf{q_{mn}}$ which is the belief held by nth bit node about itself that will be transmitted to mth check node, considering its own belief and received beliefs from all check nodes except mth check node.

$$\mathbf{q}_{mn}[0] = \alpha * (1 - f) \prod_{m' \in \mathcal{M}(n) \backslash m} r_{m'n}[0]$$
$$\mathbf{q}_{mn}[1] = \alpha * f \prod_{m' \in \mathcal{M}(n) \backslash m} r_{m'n}[1]$$

α is chosen such that $\mathbf{q}_{mn}[0] + \mathbf{q}_{mn}[1] = 1$
In this step we also calculate net belief of nth node about itself. This is computed as

$$\mathbf{q}_n[0] = \beta * (1 - f) \prod_{m \in \mathcal{M}(n)} r_{mn}[0]$$
$$\mathbf{q}_n[1] = \beta * f \prod_{m \in \mathcal{M}(n)} r_{mn}[1]$$

β is chosen such that $\mathbf{q}_n[0] + \mathbf{q}_n[1] = 1$
We use these for estimating $\hat{\mathbf{n}}$ as follows: $\hat{\mathbf{n}}_n = 0$if $\mathbf{q}_n[0] > 0.5$ and $= 1$ otherwise

We now check if this estimate satisfies $H\hat{\mathbf{n}} = \mathbf{z}$. If it does, we have the required estimate and we stop here. If it is not satisfied, we go to Horizontal step. If iteration doesn't stop after fixed number of steps, we stop anyway. Once we have estimate of noise vector $\hat{\mathbf{n}}$, we return decoded codeword $c = \mathbf{x} + \hat{\mathbf{n}}$

It should be clear why this algorithm is also called Belief Propogation.

5 Implementation Details

We created and stored Tanner Graph as adjacency lists storing indices of nodes adjacent to a given node. We used two `arrayList<arrayList>` for this - one for check nodes and other for bit nodes. We stored messages to be passed in a data structure which we call as `Table`. It is basically Hashmap with tuples as keys. We implemented this using `hashMap<hashMap>`.

6 Analysis

Parity check matrices for LDPC codes are very sparse. Column and row weights are very small and are in the range of 2 - 10. Thus Horizontal step takes only linear time with n, where n is length of code vector. Vertical step also takes linear time. We can compute syndrome/matrix multiplication very efficiently using constructed graph structure in linear time. If parity check matrix was not sparse as we assumed, horizontal step takes exponential time! Summarily, complexity of decoding using sum product algorithm for LDPC codes is $O(n)$

7 References and Acknowlegements

Algorithms for lzw encoding and decoding are from Prof.B.K.Dey's slides on source coding. We've learnt sum product algorithm and LDPC codes from D McKay Information Theory, Inference, and Learning Algorithms which is available for free online viewing here: `link`. Complete Bibliography is available in this book.

Thanks to Prof.B.K.Dey for suggesting implemetation of sum-product algorithm.

Lempel-Zev-Welch Compression Algorithm

Sasank Chilamkurthy, Tharun Kumar Reddy, Varun Bairaboina

25 April 2014

1 Introduction

This is a source coding algorithm or in other words, lossless compression algorithm. This is a dictionary based algorithm. It stores previously appeared string patterns in a dictionary. If a pattern in dictionary appears again, we transmit the index of the entry instead of the entire pattern. This is how compression is achieved. We implemented a very basic version of this algorithm. Many variants of lz algorithm are available which incorporate various optimisations in dictionary management etc.

2 Encoding

2.1 High Level Description

1. Maintain a list of substrings appeared before in dictionary.
2. Store the output string from the source in a buffer.
3. Check if the present string at the buffer is there in the dictionary.
   - If yes, then wait for one more symbol to come into the buffer and then go back to step 2.
   - If no, then
     (a) Find the substring (buffer string excluding the last symbol) in the dictionary and transmit its index.
     (b) Transmit the last symbol.
     (c) Empty the buffer.

2.2 Implementation details

- Dictionary here requires searching by substrings. So we implemented dictionary as a `hashMap<String>` which has strings as its keys and integer indices as its values
- We will not transmit integer index in base 10 or 2 because that will be waste of character space as ascii characters can take values in range 0-127. So, we transmit index in base 128 system by converting it to corresponding character sequence.

3 Decoding

Decoding is simmilar to encoding.

1. Maintain a list of substrings appeared before in dictionary.
2. Store the input string in buffer. Fill buffer with incoming characters until it is of expected length. We calculate expected length from current dictionary size.
3. Convert buffer[0:length-2] to integer using base 128 system. This was the index transmitted.
4. Access string at this index from dictionary and append buffer[length-1] to it and return this.

4 Analysis

Everything (decoding and encoding) is done in constant time given performance of hashmap.

ARM7TDMI Processor Implementation in Verilog

Sasank Chilamkurthy, Varun Bairaboina, Bhanu Vikas, Nithin Nethipudi

May 2014

Abstract

We have implemented pipelined version of ARM v7 processor.

1 Pipeline stages design

We have 6 pipeline stages:

1. Instruction Fetch (F)
2. Register Read (R)
3. Multiplier (M)
4. ALU
5. Memory (MEM)
6. Writeback (W)

1.1 Instruction Fetch (F)

We fetching instructions from instruction cache and determine type of instruction. There is not much complex logic here.

1.2 Register Read (R)

We read all poosible registers that can be needed by a instruction.Also, forwarding decisions are made here.

1.3 Multiplier (M)

Multiplier block does any possible multiplications required.

1.4 ALU

This stage has ALU, Barrel shifter and condition field checker. Flag outputs from ALU is forwarded to the same stage i.e. fed back to itself.

1.5 Memory (MEM)

Accesses memory for Load/store instructions.

1.6 Writeback

We write back to registers and PC depending on instruction.

2 Implementation details

We made different modules for each blocks mentioned above and instantiated them in top level controller module. We maintained a `will_this_be_executed` signal at each pipeline stage to facilitate easier flushing and bubble insertion. Code is loaded by editing `initial` block of instruction cache module. We checked design with the standard fibonacci sequence generator program.

Wireless communication of microcontrollers using Amplitude Shift Keying (ASK)

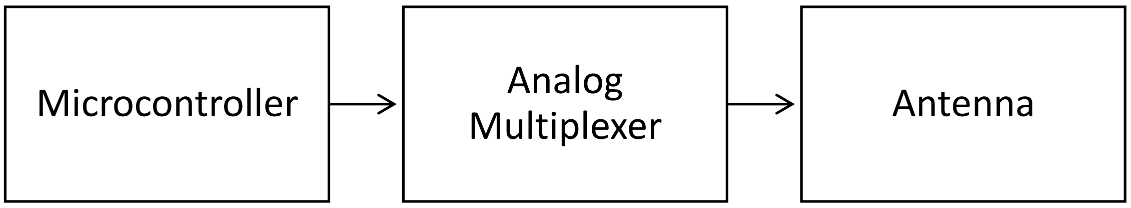
GROUP – 10: SASANK CHILAMKURTHY (110070051), ANURAAG REDDY (110070052), CHANDRAKANTH (110070053)

Aim:

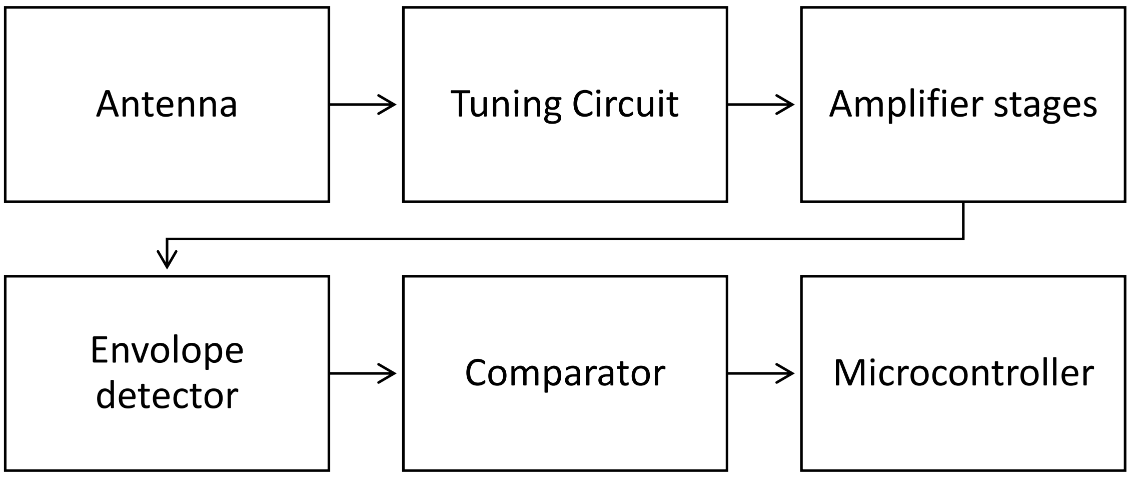
Establish a digital communication link wirelessly between two microcontrollers in medium wave band

Block diagram:

Transmitter Circuit:



Receiver Circuit:



Explanation and Problems faced

Analog Multiplexer multiplexes between ground and carrier signal as controlled by microcontroller. We tried to make our own carrier wave. But we couldn't make 1.3 MHz oscillator (our carrier frequency). We tried 555 timer. Maximum frequency that can be generated from it is about 700 KHz. We tried OPAMP oscillator. We even used 2 different high performance OPAMPs. They couldn't stand high slew rate and have very high output impedance. So high that buffering it using BiCMOSOPAMP buffer did not work. So, we had no other option but to use sine wave from signal generator for carrier signal.

Antennas are simple monopole antennas. Because of low frequency, we couldn't transmit much power. Antennas had to be close for reception.

Tuning circuit is simple LC – band pass filter centered at 1.3 MHz. Then there are two OPAMP amplifier stages. Problem here is limited gain-bandwidth product. We need high gain to amplify signal from antenna. But high frequency limited the operation and it took trial and error to decide OPAMPs to be used.

High frequency operation debugging took the most of our time. 1.3 MHz may be too limited a frequency for good bandwidth, but we couldn't go any higher.

Demodulator is simple envelope detector. We compared the output of envelope detector with a fixed DC voltage and output will be the input signal to microprocessor.

We used UART protocol to communicate between microcontrollers. We sent a text message from transmitting microcontroller and received and showed it on LCD on receiving microcontroller. We tested for various Baud Rates possible. Maximum baud rate we tested to be working is 31,250 – highest the microcontroller we used can operate.

Once high baud rate communication link is established, we could've transmitted almost anything. But in this project, we were seriously limited by processing power and hardware. We tried to transmit digital audio. There are codecs available which can compress voice to 2400 Baud! In the first place, we do not have 3.5 mm jack to get analog waveform of sound from computer/phone. Then we need to sample at reasonable frequency. We cannot transmit this directly without compression. We had neither time nor processing power to do that. We tried using PC's COM port. But it didn't work out. We were using 8051 based microcontroller. It has very low RAM. So, we had to stop at transmission of simple text messages.

Working circuits

Transmitting 'A' continuously using highest baud rate:

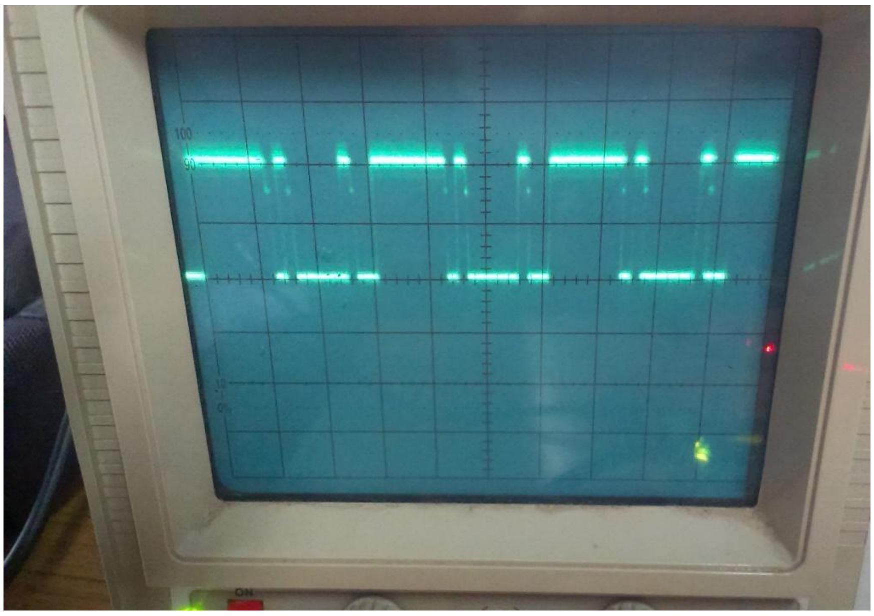


Figure 1Waveform at receiving pin of microcontroller. We've given delay between two transmissions so that frame can be seen clearly. ASCII code of A - 41h – 0010 0100 should be the bits in one frame. Observe that it matches the waveform.

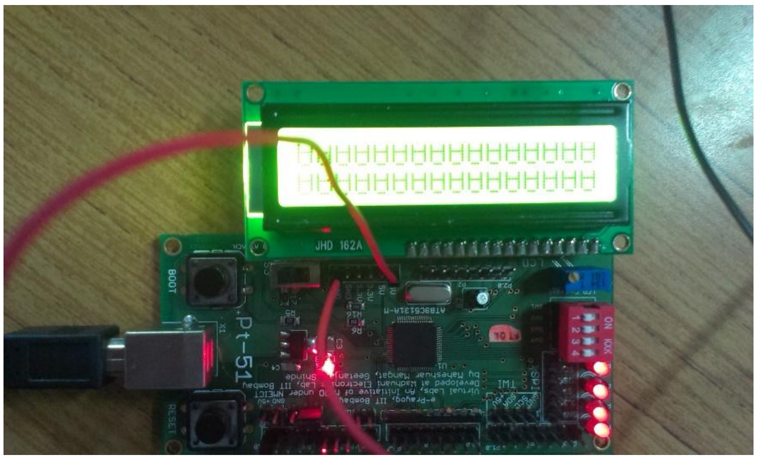


Figure 2 Receiving microcontroller

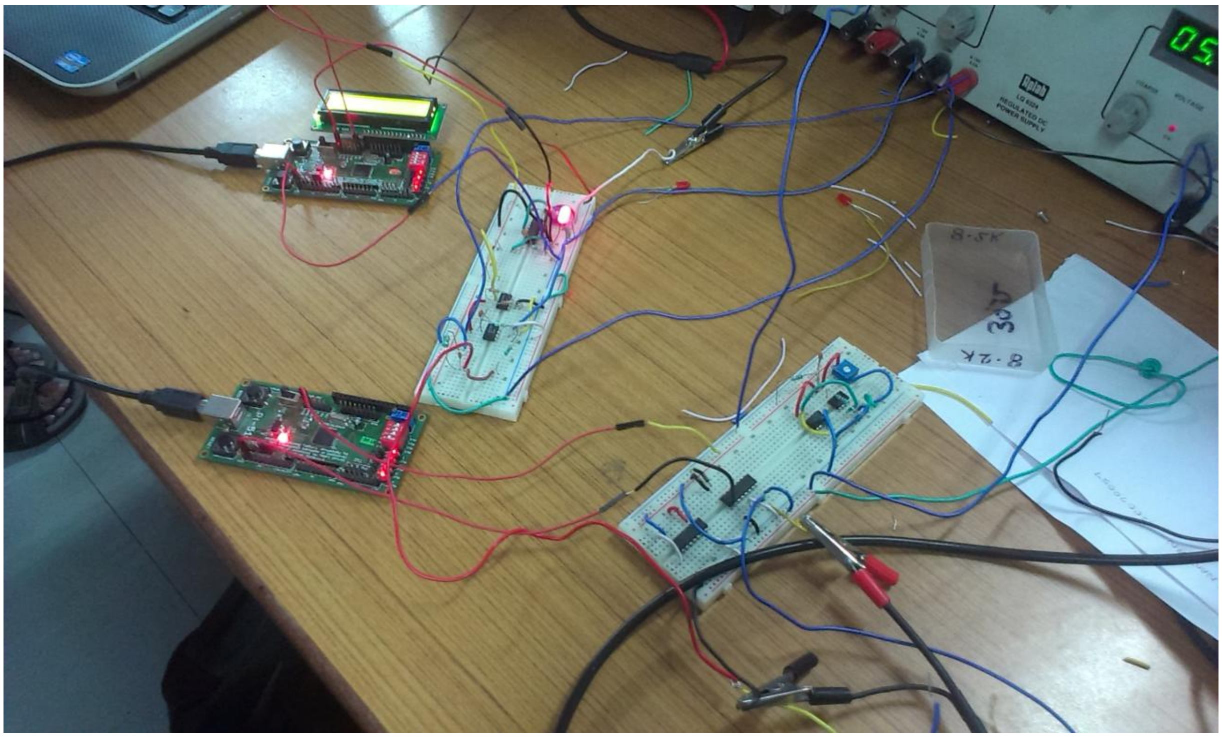


Figure 3 Microcontroller without LCD is transmitter. One with LCD is receiver.

Further improvements that can be done

1. Use higher frequency
2. Improve antennas to transmit for longer distances
3. Sample and compress audio (using DSP?) and transmit it
4. Reconstruct the audio from received bits

*e*SLATE

CONTENTS:

Problem Statement

To design an electronic slate, i.e. a board on which one can write

To be able to see the written pattern on a GLCD too.

Design Approach and Reasons for Choosing the Various Components

Basic idea for this experiment has been taken from the source:

http://www.fischtisch.de/uploads/Main/p287-hudson.pdf

We expanded on this concept and used a led as a light sensing device to create an electronic slate on which we can draw patterns. This drawing has been done using a laser. The shape in which the laser pointer is moved on the input device will then be on the shown GLCD.

During the initial stages of the project, we thought of using a photodiode matrix to create the input device. But photodiode matrix was not available in the lab. So we did some research through internet to find some alternatives for the photodiode matrix. So, ultimately we decided to go for another input device. Searching for that we came across the above mentioned paper which describes the usage of a led to sense the light falling on it.

This idea being new to us and innovative has been adopted for the future course of the project. Thus we obtained an input device, a led matrix, to serve this.

In this project, this led matrix served as the sensor matrix. Basically when a light ray is incident on a particular led the led will become on. When we shine the laser on the matrix in a particular shape the corresponding led's will be on. For safety reasons we used resistors in the circuit. The value of the resistances is 220 ohms

In the next step, we had to decide the number of leds that should be used in the input stage.

We finally settled for a 5X7 led matrix because going for any higher number of leds leads to increase in current that is needed to drive these leds. But the CPLD board limited us to use a 5X7 matrix. Thus the numbers of leds and output stages are determined.

So the basic hardware units were not that complicated. We kept them as simple as we can so that we don't face any problem later. The led matrix, resistors were assembled on a breadboard and the assembly is interfaced to a GLCD

As illustrated in following figure in its normal operation, light is emitted when current flows across the junction of an LED from its anode to its cathode. On the other hand, because it is a diode, it does not nominally conduct current in the opposite direction, when it is reverse biased by placing a positive charge on the cathode and a negative charge on the anode. However, small amounts of current do leak across the diode junction when it is reverse biased. The amount of such leakage is related to the incident light striking the LED, with higher light levels producing substantially larger leakage across the junction. It is possible to use the CPLD to exploit this property to measure incident light.

This is done as follows. First, both ends of the LED and current limiting resistor pair are wired to separate I/O pins. To sense light, the CPLD briefly reverse-biases the LED by setting pin A to logic 0 (ground or 0v) and pin B to logic 1 (typically +5v). This charges the small intrinsic capacitance found in the wire and diode. Pin B is then switched to high impedance input mode. At that point, the input value at the pin will read logic 1. Over a short period of time, the charge on the wire will leak past the reversed biased LED junction to the ground provided by pin A, with the input at pin B subsequently dropping far enough to

register as a logic 0. By measuring the time it takes for this charge to drop below the logic 1 level, we can determine the rate of reverse bias leakage, and hence the level of incident light (again with higher light levels causing more leakage, and hence shorter times). As we're using CPLD, not microcontroller, pins can't be assigned as inputs and outputs dynamically. So, we connected two wires to pin B shown in figure below and configured one as input and other as output. This wouldn't change functioning of the circuit because input pins of CPLD are always at high-impedance state.

LED mode of working for the purpose of sensing light:

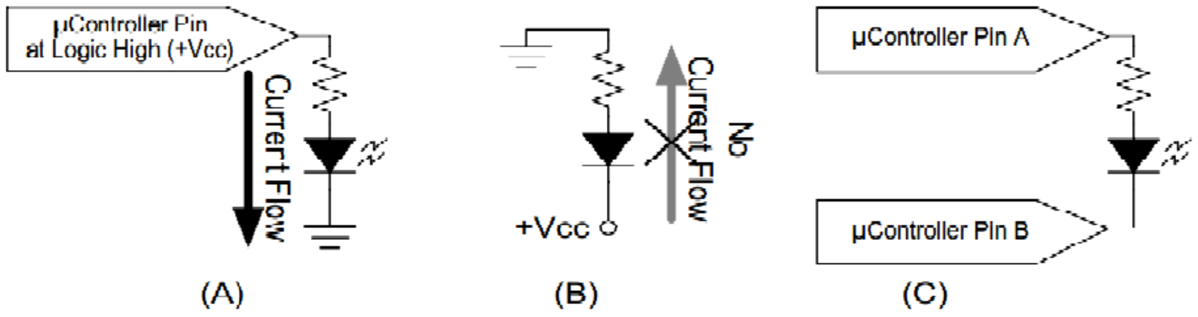
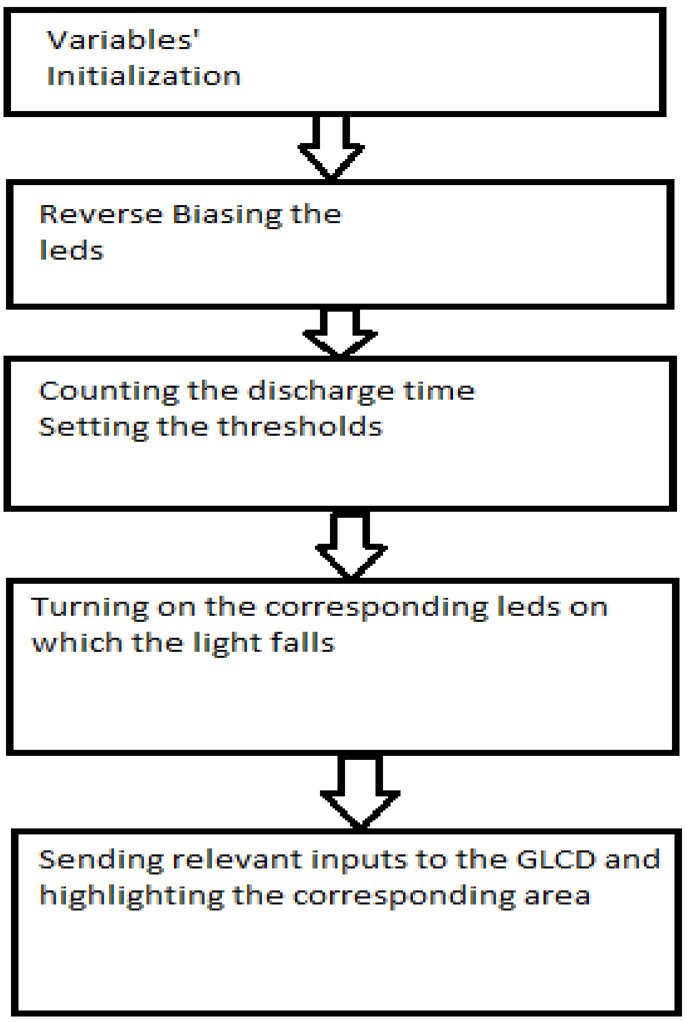


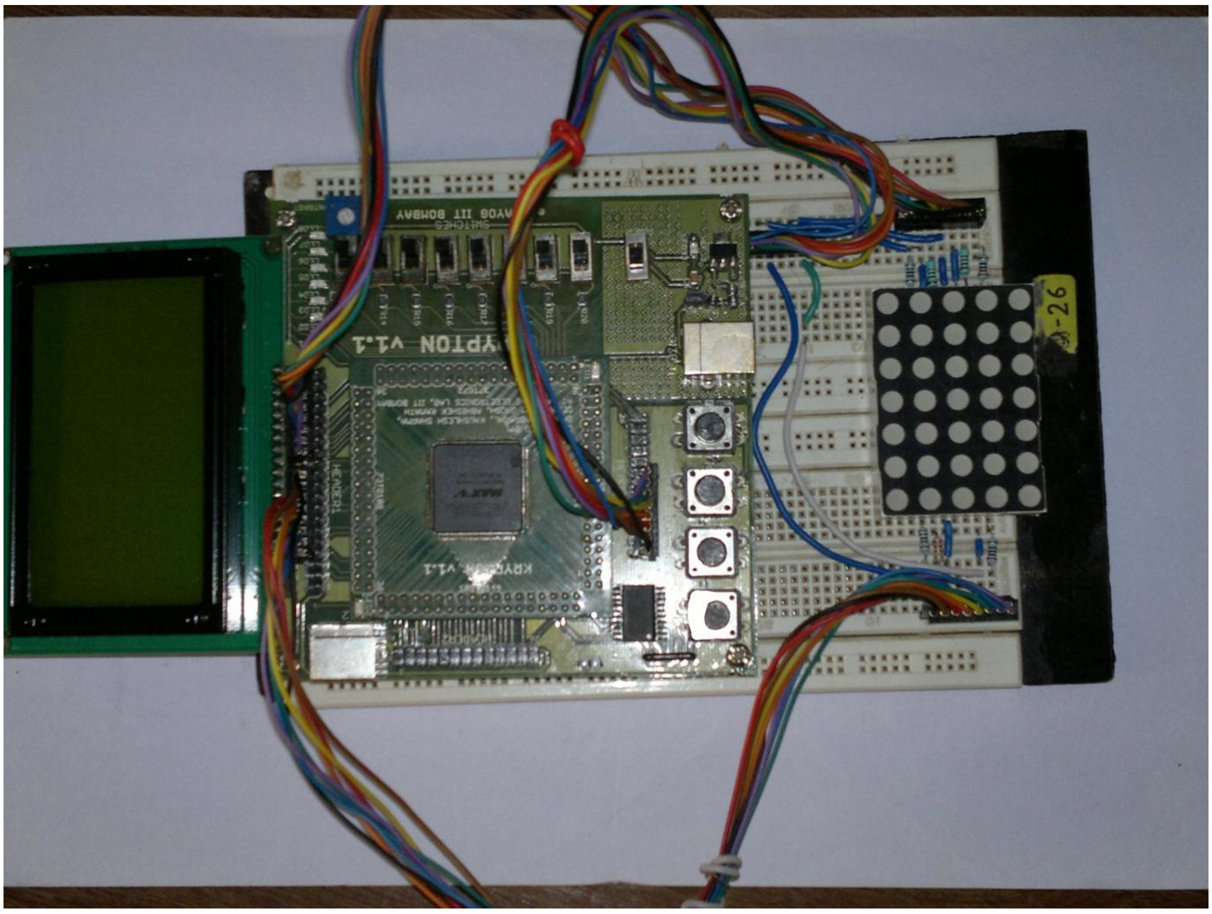
Figure 2. a) Normal use of an LED,
b) Reverse biased LED, c) Circuit for light sensing.

The algorithmic flow chart is as shown in the figure below


Variables' Initialization
Reverse Biasing the leds
Counting the discharge time Setting the thresholds
Turning on the corresponding leds on which the light falls
Sending relevant inputs to the GLCD and highlighting the corresponding area

The above algorithmic flow chart describes the way in which our code has been written.

Below shown is our assembled circuit:



Conclusion and suggestions for further improvement:

- In the onset of the conclusion, we satisfactorily present the ***e*****SLATE,** an electronic slate on which we managed to fairly produce the shapes drawn on it using a laser and output the shape on a GLCD too.

- As the laser is moved across the input pad, the leds on which the light falls turn on and also the corresponding part is highlighted in a partition of the GLCD.

- We did it on a miniature scale.   This can be further improved to create larger version that can be of a much better use.

  Evidently, they have the potential to replace chalks, pens and pencils☺.

- A major suggestion is to use higher sensitivity leds and smaller ones.

The leds we used a large and hence the shape you desire, for example 3, may not be produced as might be desired.

Hence, increasing the number of leds and decreasing their size and gap between them can help produce output, that resembles the input more.

List of references:

1. http://www.fischtisch.de/uploads/Main/p287-hudson.pdf
2. ePrayog website.

A DIGITAL PROJECT BY:

SASANK CHILAMKURTY

ANURAAG REDDY P

CHANDRA KANTH E

Traffic Sign Classification using Random Forests and LDA on Histogram of Oriented Gradients descriptors

Sasank Chilamkurthy - 110070051
Tharun Kumar Reddy - 110070048
Rajeev Puppala - 110070055

October 13, 2014

We evaluate the performance of Random Forests and Linear Discriminant analysis (LDA) for traffic sign recognition using Histogram of Oriented Gradients (HOG) [3] descriptors. We use German Traffic Sign Recognition Benchmark (GTSRB) dataset available `here`. Description of data set and overview of results of various learning algorithms used for this benchmark is presented in [1]. Random Forests was shown to one of the best classifiers just behind convolutional neural networks which are very computationally intensive. More details on this implementation is presented in [2].

Different HOG descriptors of test images are supplied along with dataset. We implement LDA on these descriptors as a baseline for performance. We chose Random Forests over others as Random Forests are easy to implement, understand and perform comparably to other state of the art machine learning algorithms.

If time permits, we will do similar classification for faces or human poses.

References

[1] Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, August 2012, Neural Networks (32), pp. 323-332

[2] Traffic sign classification using K-d trees and Random Forests, F. Zaklouta, B. Stanciulescu, O. Hamdoun, August 2011, International Joint Conference on Neural Networks (IJCNN) 2011

[3] N. Dalal, B.Triggs Histograms of oriented gradients for human detection, Proceedings of the IEEE conference on computer vision and pattern recognition, 2005 (pp. 886893).