OULUN YLIOPISTO
UNIVERSITY of OULU

DEGREE PROGRAMME IN ELECTRICAL ENGINEERING

Embedded Linux Based Demonstration Device for Printed Electronics

Thesis author _____
Muhammad Saad Saud

Thesis supervisor _____
Prof. Juha Häkkinen

Approved _____

Grade _____

Saud M. S. (2014) Embedded Linux Based Demonstration Device for Printed Electronics. University of Oulu, Department of Electrical Engineering, pages 114 p.

ABSTRACT

As use of printed electronics is going towards expansion in near future, researchers are studying methods for development of printed components, and also procedures for driving them. Few low speed but low cost, and flexible stand alone systems e.g., some sensor systems and displays; have already made their way to market after competing conventional, high speed, high cost, and rigid silicon electronics. At present, there are different type of printed sensors and display matrices, which have been developed but not being used in stand alone systems.

Thus, there was a need for demonstrating 'developed printed electronic components' in such a way, that they are integrated with other core electronic circuits like embedded systems, to assure their ability to replace individual conventional component.

In the thesis, some experiments were conducted to drive printed electronic components while overlooking their manufacturing details; i.e., the circuit used to drive the printed components was not a part of printed technology (e.g., printed processing, memory units), rather a conventional system (e.g., microcontroller and microprocessor boards available in market) is used. Particularly for this project the printed components are meant to be capacitive sensor matrix, resistive touch screen, organic light emitting diodes (OLED) matrix, electrophoretic display matrix, and electrochromic display matrix. These components were interfaced using embedded system in such a way that they can be driven meaningfully e.g., resistive touch screen can control display option for OLED matrix.

Embedded Linux has significant advantages over previously used OS and non-OS based solutions as it is robust, scalable, and manage resources efficiently. Such a system with backbone of Linux processing was a part of design, which handled a separately designed interface board capable of interfacing 13 capacitive sensor (maximum 8 pF) inputs, a four wire resistive touch screen (with few hundred ohm resistance between terminals), a current driver with 50 mA segment current for 10 x 10 display matrix and voltage range from 0 to 5 V, voltage driver for two led segments capable of dissipate 40 mA at 5 V, finally a voltage driver for separate 10 led segments capable of dissipating 40 mA from 0 to 15 V or a 10 x 10 display matrix with power rating of maximum 16.9 mA at 0 to 15V.

So, in this way different displays or individual led segments can be interfaced with a system which can download their display configuration from HTTP client based user interface, and upload the capacitive sensor and resistive touch screen readings back to user. The device was capable of changing demonstration parameters e.g., blinking, animation for displays via user interface, and program structure was kept convertible so that minor changes can create completely new demonstration: interfacing different components. Design included a power board capable of providing stable 1A supply current at 5V. A Current-Voltage (I-V) characterizing board was part of design which can measure I-V curve of printed solar cells. Printed electronic components were interfaced with embedded systems without much problem, making one comfortable in conclusion that these individual components are ready to replace conventional components in non-standalone systems at-least.

Keywords: embedded systems, linux, printed electronics

Saud M. S. (2014) Linux-pohjainen, sulautettu demonstraatiolaite tulostetulle eletroniikalle. Oulun yliopisto, Sähkötekniikan osasto, sähkötekniikan koulutusohjelma, 114 s.

TIIVISTELMÄ

Tulostettavan elektroniikan käytön laajentuessa lähitulevaisuudessa tarvitaan keinoja tulostettujen komponenttien toiminnallisuuden demonstrointia varten. Esimerkikis eräät halvat ja joustavat anturit ja näytöt ovat jo löytäneet tiensä markkinoille korvaamaan kalliin ja joustamattoman silikonipohjaisen elektroniikan komponentteja.

Tämän työn taustalla on tarve demonstroida tulostettuja elektroniikkakomponentteja tilanteessa, jossa ne on integroitu osaksi perinteistä elektroniikkaa. Näin osoitetaan niiden kyky korvata yksittäisiä perinteisesti toteutettuja komponentteja. Tässä työssä toteutetaan elektroniikka seuraavien painettujen komponenttien demonstroimiseen: kapasitiivinen leheisyysanturimatriisi, resistiivinen kosketusnäyttö, orgaaninen valoemittoiva diodimatriisi (OLED), elektroforeesinen näyttömatriisi ja elektrokromininen näyttömatriisi. Nämä komponentit liitettiin kehitettyyn demonstrointilaitteesen siten, että niitä voidaan käyttää yhdessä toiminnallisten demonstraatioiden luomiseen niin, että esimerkiksi resistiivinen kosketusnäyttö voi ohjata OLED matriisisnäyttöä.

Kehitetty demonstraatiolaite, joka on rakennettu Linux-käyttöjärjestelmää hyödyntävän sulautetun tietokoneen ympärille, sisältää 13 kpl enintään 8 pF kapasitiivisen lähestymissanturin sisääntuloa, neljä tuloa resistiivistä kosketusnäyttöä varten (jossa muutamien satojen ohmien resistanssi terminaalien välillä), 50 mA/0 - 5 V lähdöt 10 x 10 näyttömatriisille, 5 V/40 mA jänniteajurilähdöt kahdelle LED-segmentille sekä kymmenen 1 - 15 V/40 mA yleiskäyttoistä jänniteajurilähtöä.

Laitteen toiimintoja demonstroitiin kehittämällä esimerkkisovellus, jossa käyttäjä voi ohjata LED-näyttöä ja seurata kapasitiivisen lähestymisanturin tai resistiivisen kosketusnäytön tilaa verkkoselaimella käytettävän käyttöliittymän kautta. Sovelluksen rakenne oli suunniteltu helposti muunneltavaksi niin, että sitä kehittmällä eri käyttäjät pystyivät luomaan kokonaan uusia demonstraatioita, joissa voidaan liittää yhteen erilaisia painettuja komponentteja.

Tulostetut elektroniikkakomponentit liitettiin deonstraatiolaitteeseen ilman suurempia ongelmia, mikä omalta osaltaan todistaa kysesiten painettujen komponenttien olevan valmiita korvaamaan perinteisesti toteutetut komponentit ainakin osana periteisellä elektroniikalla toetuttettuja järjestelmiä.

Avainsanat: sulautetut järjestelmät, Linux, tulostettava elektroniikka

PREFACE

The work presented in this thesis has been conducted at the Department of Electrical Engineering, Electronics laboratory, University of Oulu, Finland. The aim of the thesis was to design an embedded Linux based demonstration device which can interface different type of printed electronic components.

I am grateful to all personnel from electronics lab for providing such a cooperative working environment. I am thankful to Niina Piipo, Marja Pohjola-Effe, Jari Pakarinen, Anssi Rimpiläinen, Polojärvi Matti, Christian Schuss, Tore N. Roald Leikanger, Juho Pylvänäinen, Antti Mäntyniemi, Nouman Bashir, and Haseeb Mufti for helping me in resolving practical problems faced in my thesis. I thank my second examiner Prof. Kari Maatta for his valuable suggestions and comments.

I would like to show my gratitude to all and especially my supervisor Prof. Juha Häkkinen for all of his support and contribution in design and implementation. Finally, I would like to dedicate this thesis to all my family members for their love and support along the way.

Oulu 12.2.14

Muhammad Saad Saud

LIST OF ABBREVIATIONS AND SYMBOLS

AD	Analog Devices
ADC	analog to digital converter
ARM	advanced RISC machine
BB	beaglebone
BBB	beaglebone black
BIOS	basic input/output system
CAN	controller area network
CDC	capacitance to digital converter
CGI	comon gateway interface
CSS	cascading style sheets
DAC	digital to analog converter
EC	electrochromic
EOC	end of charging
EP	electrophoretic
ESR	equavalent series resistance
FR-4	flame resistant grade 4
GPIO	general purpose input/output
HDMI	high definition multimedia interface
HTML	hyper-text markup language
I2C	inter-integrated circuit
IDE	integrated development environment
IDE	integrated drive electronics
INIT	initialization
IO	input/output
IP	Internet protocol
I-V	current–voltage
IVR	interactive voice response
JFFS2	journalling flash file system 2
LAN	local area network
LCD	liquid crystal display
Li-Ion	lithium ion
McASP	multichannel audio serial port
MIPS	microprocessor without Interlocked pipeline stages
MMU	memory management units
MOS FET	metal oxide semiconductor field effect transistor
OLED	organic light emitting display
OpAmp	operational amplifier
OPV	organic photovoltaic cell
OS	operating system
OTP	open telecom platform
PCB	printed circuit board
PDA	personal digital assistant

PHP	hypertext preprocessor
POS	point of sale
PSTN	public switched telephone network
PWM	pulse width modulation
RFID	radio frequency identification
RISC	reduced instruction set computing
RT	real time
SAR	successive approximation register
SDRAM	synchronous dynamic random-access memory
SPARC	scalable processor architecture
SPI	serial peripheral interface
SPST	single pole single throw
SRAM	static random access memory
TI	Texas Instruments
UART	universal asynchronous receiver/transmitter
UI	user interface
USB	universal serial bus
VCC	plus collector supply line voltage
VOIP	voice over ip
$\sum-\Delta$	sigma delta
C_{timer}	timing capacitor
F_{OSC}	oscillating frequency
I_{CHG}	charging current
I_{max}	maximum current
I_{SEG}	segment current
R_{FB}	feedback resistance
R_{SENSE}	sense resistance
t_{EOC}	end of cycle time
$V_{BAT(FLT)}$	floating voltage of battery
ΔI	ripple current

1. INTRODUCTION

Printed electronics represents a technology in which regular printing techniques are used to create electrical circuits on wide range of flexible and low cost substrates e.g., plastic substrate; utilizing organic semiconductors, inorganic semiconductor, or metallic conductors etc. This term is mostly used in contrast to conventional silicon based fabrication which is a rigid and expensive technology. Printed electronics has a huge potential as an upcoming technology, with the prediction of a multi-billion dollar market size due to flexibility and low cost fabrication it provides. It has the potential to replace inflexible and costly silicon electronics, revolutionizing the spread of electronics applications, if the researchers are able to develop inexpensive fast fabrication process. [1, 2, 3]

Since an embedded system based solution was required for the development of convertible demonstration system of printed electronic components; i.e., user can configure the resulting device to change the interfacing components (e.g., from electrochromic display to Organic LED display) and their effects (e.g., blinking, animation) respectively. Printed components included Solar Cell, Capacitive Switch Matrix, Resistive Switch Matrix, Electro-chromic Display Matrix, OLED Matrix, Electrophoretic Display Matrix, and other Led Display Matrices (having operating voltage/current in defined range). The initial task was to find an embedded solution which can interface all of the above mentioned printed components or some of those and to make a potential application for 'Point of Sale' (POS).

Embedded Linux system for interfacing printed component is a better option, though not compulsory. By embedded Linux, one actually means the operating system "Linux" being used for embedded device. Currently many developers these days are using Embedded Linux because of its qualities i.e., reliability, feature packed, robust and quick to market. [4]

Less design and implementation time (without optimal end results), cost and space limitation for circuit boards are key system requirements. Packaging (space consideration) is thought to be as one of the main restrictions of system; as a potential application for POS it should have least possible height, and should also have minimal length, and width. Cost is one of the restricting factors, as it is obvious in almost every practical system design. System's powering solution was also required, so that its operation can be extended from POS to remote applications, if desired. So, very good battery backup with input power from a solar cell can be a design parameter.

Potential applications with respect to POS can be printed feedback through capacitive and resistive sensing matrices or interactive (Responsive) lighting through any of printed display. Application scenario was added in the form of current–voltage characteristic (I-V curve) measurement of organic solar cells. The system was designed such that it allows configuration through a web server running on embedded Linux device. Since, embedded Linux machines come with enormous processing resources, not only we can implement a multi-process system realizing server and hardware control on same time, but also take advantage of its processing power to make complex system like IP exchange. Embedded Linux design was not still compulsory, but addition of configuration application through web server running on device posed need to use Linux, one of the most comfortable OS in the field of networking. Figure 1, below will give an introductory understanding about the system.

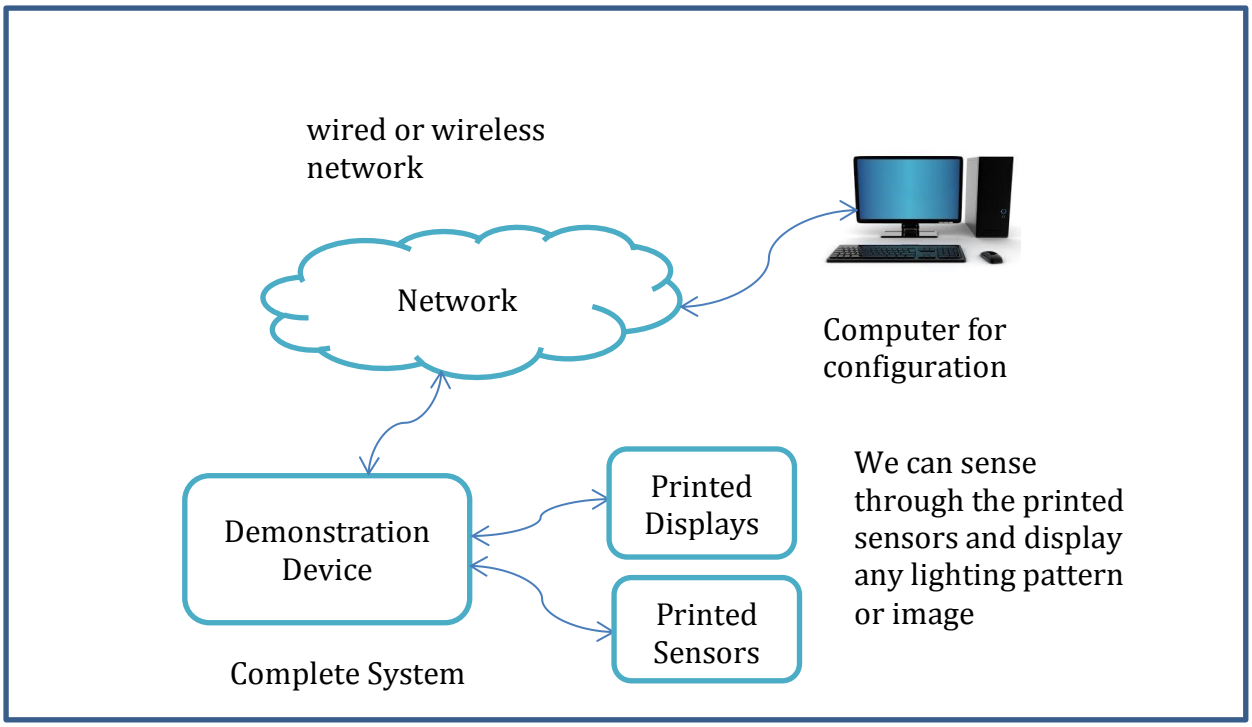


Figure 1. Big Picture of Demonstration Device.

In later chapters, details regarding types of printed components, possible solutions with embedded system designs and complete structure of device will be discussed. So starting from introductory chapter for printed electronics which was necessary to understand because that is why the reader would be able to comprehend actual motivation of this project: to be a part of development in completely new domain for electronic systems. So, light discussion regarding its definition, procedures, examples, and applications is presented in its introductory chapter. Embedded systems and typically Linux for those systems needed a preliminary discussion as reader should be aware of some basic facts about it. In chapter related to embedded Linux, some of the initial topics are touched, leaving behind the hard topics for anyone interested to be the part of development of such systems; i.e., reader will not learn how to actually develop an embedded Linux system entirely, but rather how to use those systems for one's own requirement. After introductory chapters, one can find actual details of system developed for the demonstration in two parts: its hardware structure firstly, and its software part secondly. Then, there is a chapter for demonstration details; i.e., how final behavior of the system should look like, and what other possibilities can be there to develop same kind of product. There is a chapter for discussion having fractional information which might be required for future work and lastly it has a summary as a standard format.

2. INTRODUCTION TO PRINTED ELECTRONIC COMPONENTS

Printed electronics is referred to flexible electronics developed by using conventional printing technology to create electrical circuits on different types of substrates (which can be low cost, flexible, and translucent etc.) as mentioned earlier. Common low-cost printing equipment is used for the purpose, e.g. inkjet printing. It has following characteristics: [1]

- Wide Spread
- Low-Cost
- Low-Performance electronics

Printing has caught significant interest for realization technique for low cost, large area electronic system. Process complexity is lower because it is purely an additive processing. We can use low cost substrates like plastic, metal foils, etc. It is believed that it will provide low cost realization for easily deployable electronic systems such as displays, sensors, etc. It offers combination of synthetically derived inorganic nanoparticles and organic materials, a wide range of inks can be used to have printed passive components, multilayer interconnections, transistors, memories, batteries, etc. Diverse functionalities and materials can be on same substrate proves to be very advantageous. [5]

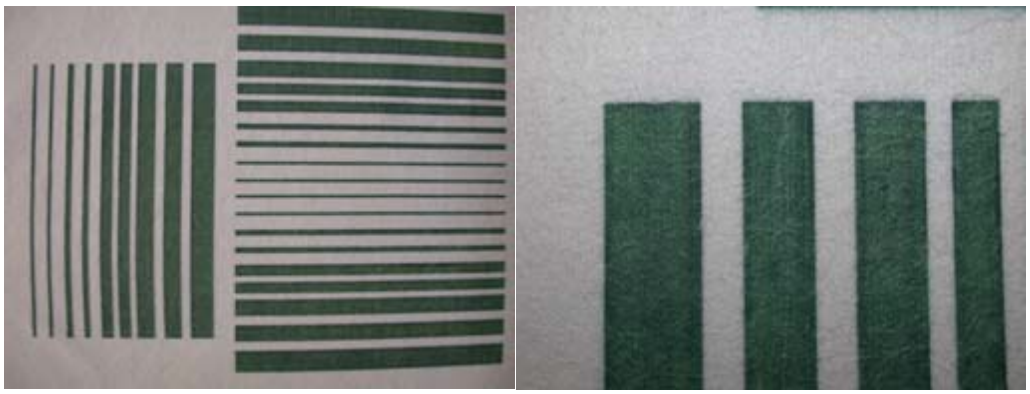


Figure 2. Printed patterns of conductive inks on stretchable non-woven substrates.

Figure 2 shows example printed patterns of conductive inks, was taken from [6].

2.1. PRINTING TECHNOLOGY FOR ELECTRONICS

Printing techniques available for electronics are screen printing, inkjet printing, stamping (nanoimprinting) and gravure printing. There are some other printing techniques which also exist but may not have been applied for electronics. [5]

2.1.1. SCREEN PRINTING

This can be seen as most mature technique for printed electronics and may be in use for electronics for decades. In this technique a patterned screen which may be patterned through photosensitive screen coating is used and viscous ink is spread on it using squeegee. Different materials are used to pattern different kind of components e.g. carbon films for resistors and silver paste for conductors etc. Resolution is not good even it is stated to be worse than 50 μm, though in theory it can be improved for range of < 10 μm. High viscosity is required to prevent spread of ink or bleed-out so

addition of polymer binders is done for viscosity. So it limits use of screen printing to application where binders can be added with acceptable loss of performance. [5]

2.1.2. INKJET PRINTING

Inkjet printing (propelling droplets of ink on substrate etc.) is famous now days as it allows non viscous inks to be used. So the active materials and solvents are used for ink purpose (without the need of binders which degrade the performance as it was described above). There are several issues with this kind of printing for example, drop-by-drop printing leads to pixilation related issues. Secondly it is slow, so low throughput is expected. Then there is uncertainty with ejection angle of drop from cone. So, ±3σ variance bound can be up to 10µm. Then in drying the drop one can see problems as well. Thickness of material is not constant after drying known as coffee ring effect shown in Figure 3. [5]

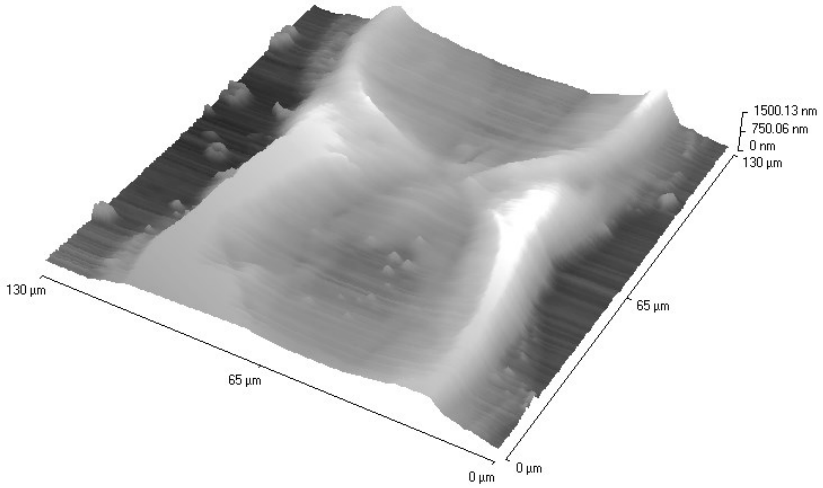


Figure 3. Atomic Force micrograph of inkjet pattern, formation of coffee ring.

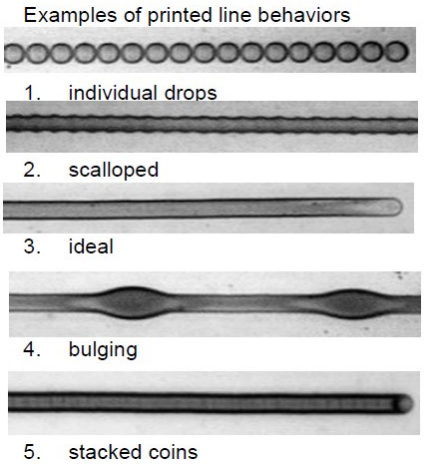


Figure 4. Effect of droplet spacing on printed line.

In above Figure 4, droplet spacing is reduced from 1 to 4; resultantly different types of patterns are seen on drying of ink. It can be seen by reducing spacing of droplets connection becomes smoother but after a certain threshold bulging problem starts. In last solution named as 4th i.e., stacked coins:

flash drying is applied where ink dry as it hit the substrate, and overlapping droplets are applied. In this kind of flash drying line suffers from poor thickness uniformity, limiting film and feature scalability. [5]

2.1.3. GRAVURE PRINTING

It is a rotary printing technique where image is engraved on cylinder [5].

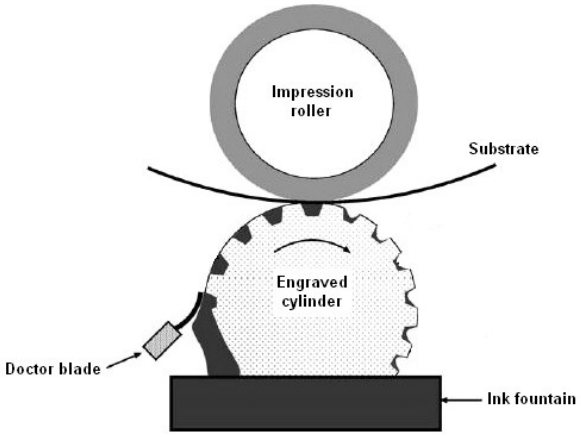


Figure 5. Gravure Printing.

Above Figure 5, shows Gravure Printing: Engraved Cylinder is having or a drum with features is etched into ink and wiped with doctor blade to remove ink from field regions is rolled over substrate to have printed patterns. It is very high speed printing technique and can produce patterns below 20µm with excellent pattern fidelity and low edge roughness. This technique requires higher viscosity inks and has defectivity challenges. This technique has not been much studied for electronic printing. [5]

2.2. APPLICATIONS OF PRINTED ELECTRONICS

By the time when feasibility of printed electronics was under discussion in research topics, some applications already made their way to market. In following, three main applications i.e., RFID, Sensors, and Displays are taken from [5], but these are not the final ones, as we can expect more standalone applications of printed electronics. The point can be made clearer by following examples of printed memory, sensor system, flexible OLED screen for mobiles and passive antenna on paper substrate. A company named THINFILM made printed memory [7] and they built first Stand-Alone Sensor System in printed electronics as well [8] shown in Figures 6 & 7 respectively.

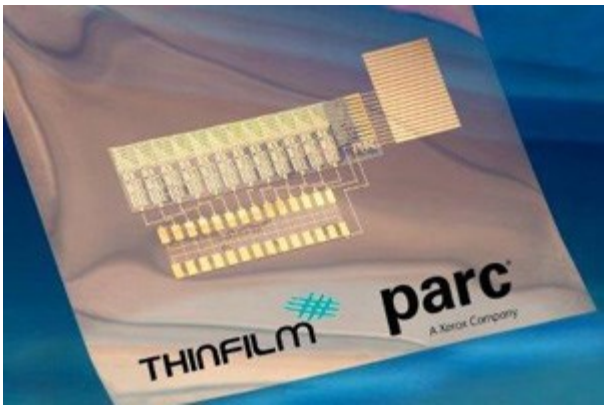


Figure 6. Thinfilm Unveils First Scalable Printed CMOS Memory.

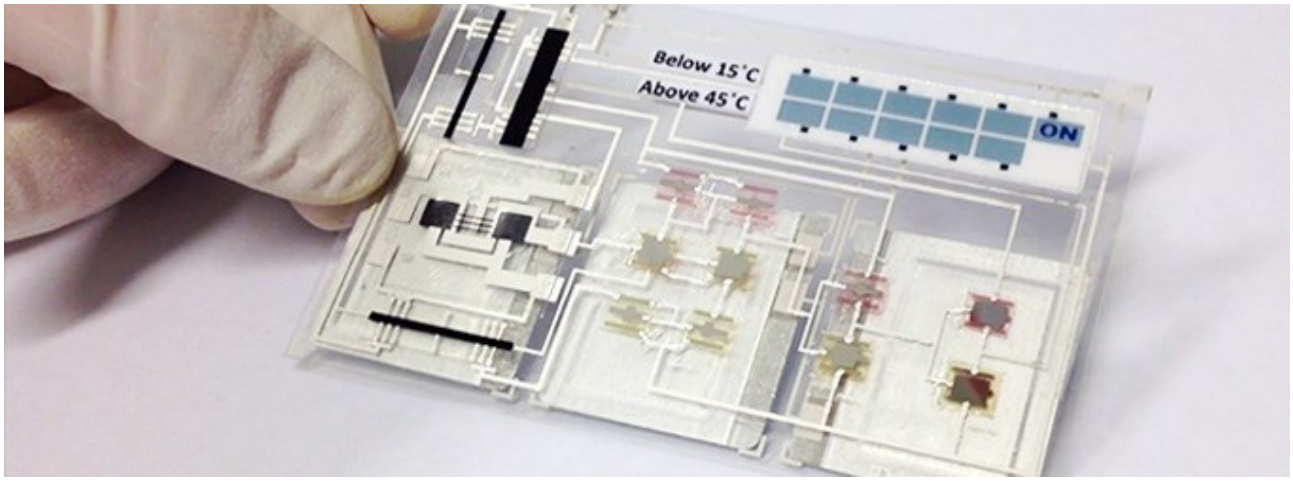


Figure 7. Thinfilm Builds First Stand-Alone Sensor System in Printed Electronics

Figure 8. LG latest to announce curved smartphone: LG Z to sport flexible display and already in production.

Figure 8 taken from news where it was claimed that LG has already started mass production of such mobile with rigid curved backplane and OLED screen. [9]

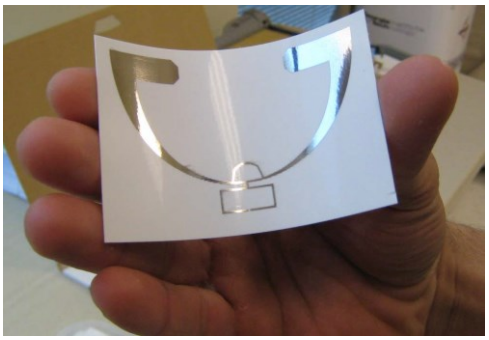


Figure 9. Photograph of the inkjet printed U-shaped RFID passive antenna along with the integrated matching stubs.

Figure 9 was taken from research publication by a group which is conducting research on printed transmission lines etc., and it concluded: "*Paper can be an excellent substrate for ultra-low-cost single-layer/multilayer wireless nodes for telecom, sensing and security applications*". Concluding remarks show success at least in implementation of passive antenna on paper substrate. [10]

2.2.1. RFID TAGS

In recent years there has been much interest in Printed RFID Tags, mainly because it's a low cost alternative to silicon based RFID.

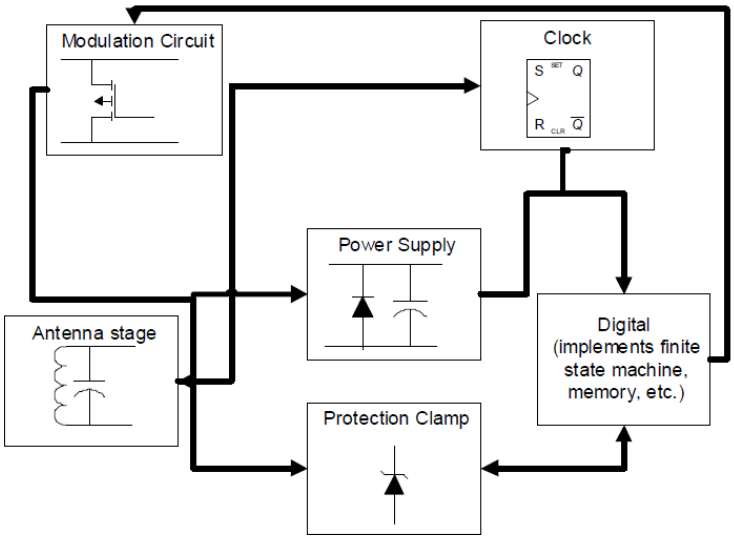


Figure 10. Example 13.56 MHz RFID tag.

In above Figure 10, RFID transistors are implemented in chip <.5 mm on side and large inductor and tuning capacitor are implemented off chip on plastic substrate. It can be seen as low cost device but simultaneously performance degrades, chip area minimizing is restricted since transistor size is large and also the switching frequency is limited. [5]

Above stated problem is not disappointing because the difficulty is arrived due the incapability of conventional printing procedures to go beyond certain widths. As soon as researchers develop procedure for fast reliable and low-cost printing performance would be improved, this point was indicated in introduction.

2.2.2. PRINTED ARRAYED SENSORS

It was already discussed that printing increase performance loss but it does offer certain advantage through addition of different materials on same substrate. For example inkjet printing allows to add different semiconductors ink on same substrate therefore realization of arrays is efficient and economical way. Without going into details of materials it can be concluded fabrication of organic transistors with exposed channels are useable as sensors. These sensors typically have poor specificity. Electronic noses (array of non-specific sensing elements and pattern matching is performed) are implemented to improve specificity. Moreover mostly printed devices show drift in electrical performance under bias. This problem is resolved by using differential architecture i.e. protected side and exposed side are joined using a bridge. Figure 11 shows example differential architecture to counter drift. [5]

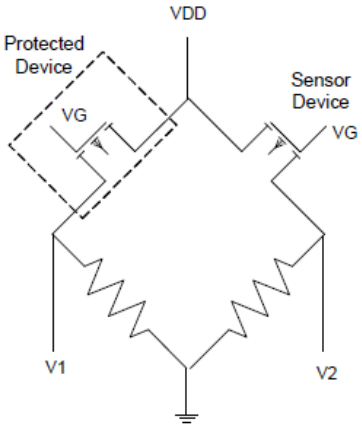


Figure 11. Differential architecture to counter drift.

2.2.3. DISPLAYS

There is an economic advantage in cost per area domain, so obviously display is the most pursued application of printed electronics. OLEDs, LCDs and various other types of displays are under research. Due to defects (which are more often seen in printed electronics than conventional technology) could be caught by eye very easily; therefore, requirements for manufacturing can be significantly important, for this reason many manufacturers have went for low resolution displays. Moreover transistor on-off ratio is relatively bad here; therefore, refresh rate is very important. Short channel lengths are required for good refresh rate, which is difficult in achieving. But the good news is, still some printed displays are being used in electronic devices. [5]

Figure 12. Flexible Printed Display.

Figure 12 shows an example for flexible display. [11]

2.3. Electrical Specification of Printed Components used in Project

Some printed electronic components which have a tendency to be good replacement for silicon electronics are displays, and sensor matrices. Here, the reasons to call them a good replacement were understandable: (a) flexibility, which gives user a perfect permission of inattentive easy handling; (b) invisibility (or translucence more precisely), which sometimes make displays and sensors more pleasant for environment; (c) Low Cost, which is sometimes most important constraint.

Thus, some type of printed LED matrices (e.g., OLED Display, Electrophoretic Displays, EC Display, and printed LEDs) and sensor matrices (e.g., printed capacitive sensor matrix, printed resistive touch screen) were considered suitable, experiments were conducted to make a system having driving capability with respect to their electrical specifications.

Table 1 describes dimensions or configurations of printed electronic components used in the project, and also electrical specifications. All interface board design considerations were dependent on following specs.

Table 1. Electrical Specification of Printed Components used in Project

Printed Component	Dimension/Configuration	Electrical Specs
LED Matrix	10 x 10	Voltage: ~5-9 V, Current: 10 mA per segment
Electrophoretic Display	10 x 10	15 V typical, 5 V optional, 12 micro-watts power per segment
EC Display	10 x 10	Voltage: 0.8 – 3 V, Current: few milli amps, per segment
Capacitive Sensor	6 x 6	8 pF max, per sensor
OLED Matrix	10 x 10	Turn on Voltage 2.5 to 3.5 V, 250 mA / matrix
Resistive Touch Screen	4 wire	few hundred ohms, between terminals

3. EMBEDDED LINUX INTRODUCTION

Though there are many definitions of embedded systems, simple definition of embedded systems would be related to any system with microprocessor or microcontroller designed to perform specific task. We can add more in it by saying that, it has input/output (I/O) with control logic, stored in system firmware. Another definition of embedded system can be combination of hardware, software and additional parts which can be either electronic or mechanical or combination of both, designed to perform a dedicated task e.g. microwave oven. Does our desktop computers with mouse, keyboards and capable of running software also fall in this definition? A satisfactory reply would be, no it is general purpose computing device and on other hand embedded system is specific purpose or at least designed primarily for some specific purpose. Recent developments have blurred the lines separating general purpose computers and embedded systems. Now hardware performance of embedded systems is comparable with general purpose computing devices but that wasn't the case some years back. [4, 12]

Arrival of microprocessors in 1970 revolutionized control. Complex structures with simple devices were made possible. Automobile emissions have decreased significantly due to use of microprocessors in its control. Luxury class automobile can have up to 70 microprocessors controlling specific tasks from engine spark to opening window when door is being closed to avoid pressure burst. The F-16 is unstable aircraft which can't be flown without on-board surface adjustment control. A jetliner can have more than 200 on-board dedicated microprocessors. [13]
Below Figure 13 shows BeagleBone an embedded Linux computer used in this project.

Figure 13. BeagleBone Linux Computer.

3.1. EMBEDDED LINUX

Using Linux for embedded system is very advantageous. More and more developers are interested in using Linux in their embedded design. In following, detailed discussion has been presented how Linux is advantageous over other possibilities.

Previously we had four or eight bit microprocessors in embedded designs. Now days, we have 32-bit microprocessor with several megabyte of memory. Desktop software requires more resources and provides less reliability. In Real-Time systems resources are limited and reliability is the key requirement. The gap of reliability with less resource is filled by Linux. It has been successfully ported to several microprocessors like x86, SPARC, ARM and MIPS. [4, 14]

Strong growth in adoption of Linux for embedded devices has been seen and it is being used in PSTN, global data networks, wireless cellular handsets, radio node controllers, back haul infrastructures that operate these networks, automobile application, games, PDAs, printers, enterprise switches routers and many more products. [14]

Reasons for growth in use are Linux's support for wide range of hardware, network protocols and applications. It is scalable from small devices to large systems like switches and routers. Its Deployment doesn't need royalties. Also increase of its use is directly increasing online support available for it. [14]

According to a survey, Linux has emerged as dominant operating system in embedded designs. Nearly half of the respondents used Linux as operating system leaving far behind the second OS which was used only by one in eight respondents. Hence, Linux is one of the most important OS used in embedded systems today. [14]

3.1.1. Typical Setup to Formation of Execution Contexts

First thing we need for development is setup. Following Figure 14 is a common arrangement for embedded Linux systems. [14]

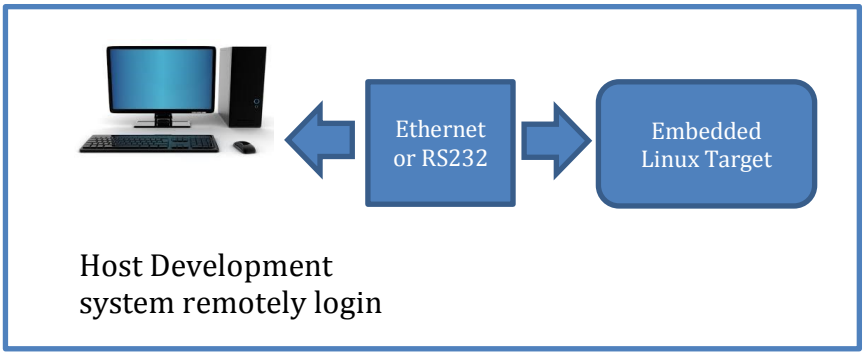


Figure 14. Embedded Linux Development Setup.

The development host normally contains tools for development which comes from embedded Linux distribution. So, one can login to the embedded Linux target from our host using Ethernet connection, serial or parallel communication busses. Sometimes USB cables are provided with boards for development as it was the case with 'BeagleBone' as well. [14]

It is noteworthy here BeagleBone comes with very easy and user friendly development environment, but it is up to user's expertise level to switch to any of difficult but advantageous in some other sense, possible development technique. As starting reference, user must know Cloud9 (Integrated Development Environment) IDE system is preloaded in BB machine. It is an online development environment for applications based on Node.js server side JavaScript as well as HTML, CSS, PHP, Java, Ruby and many more languages [15]. Cloud 9 IDE is very easy development system for embedded devices, all one have to do is to connect target Linux machine with personal computer through a USB Cable (miniUSB type B female connector is available at BB and type B male connector is needed on cable) and write down given IP address on web browser, applications can be made (code for applications can be written) and stored on BB through HTTP web browser. In this project this technique wasn't adopted because it has certain drawbacks, as one cannot go beyond the supported languages and taking full advantage of Linux wasn't possible. Rather remote login to root (with access to every resource of OS) was fairly a good choice.

Bootloader

On starting the target board (powering it on), bootloader takes control of processor immediately. This is in contrast to our conventional PCs where BIOS takes control of processor. Bootloader initializes critical hardware components such as SDRAM controller, I/O controllers, and graphics controller etc. It also initializes system memory and allocates system resources; e.g., memory, and interrupts circuits to peripheral controller. It also provides procedure for locating and loading OS image. So it loads OS, passes control and necessary information to it. All bootloaders have command to load and execute Linux kernel. When Linux kernel takes control, bootloader vanishes. Development of bootloader can be a part of embedded system development. U-Boot is an example of bootloader which is used in Freescale Semiconductor target boards as preloaded bootloader. [14]

Linux Kernel initialization and user space processes

High level view of Linux kernel can be seen as: it displays number of status from time it takes on from bootloader up to login in comprehensive boot process; shortly before login it mounts root file system. Root file system contains application programs, libraries, and utilities making Linux system. Linux kernel can mount root file system from other devices but regular practice is to mount a hard disk partition. [14]

Initialization steps performed by kernel in a context are known as kernel context in which it owns all of the resources till then. Then the first program of user space context called INIT starts. In this state, user space processes has restricted access to system and use kernel services to access devices and file I/O, it will be clearer to understand via Figure 16. [14]

Storage Considerations

Embedded systems have limited physical resources (that is what an embedded system best known for) especially memory. Hard drives are not available for these systems so mostly nonvolatile storage devices serve the purpose. Instead, flash memories i.e., solid state hard drives capable of storing many megabytes, are available. SD cards used in camera are example of Flash memory. Typical storage requirement for embedded Linux is from 16 MB to 256 MB. [14]

Flash memory is relatively slower than hard drives. It has relatively large erase blocks. If a byte has to be changed in an erase block or even a bit in a NOR Flash, whole block has to be erased and rewritten. But software control provides methods to change a bit or byte etc. These erase blocks are of larger size than hard drive sector size. Another problem is that these devices have limited write cycles e.g. 100,000 cycles per block. After that limit it gives write error. So, poorly implemented algorithms can destroy Flash memory easily. [14]

There is a relatively newer technology called NAND Flash, which differs in internal cell architecture from NOR Flash and available with lower block size making it faster. It is suitable for bulk storage for file system. [14]

In Flash memories as we discussed earlier there is finite lifetime. In addition to this we need to erase a block of 128 KB or 64 KB even for writing small file up to 8 KB. These problems motivate designers to use file system like JFFS2 or Journaling Flash File System 2 designed keeping in mind

all problem with Flash drives. JFFS2 have improved wear leveling, compression, decompression and hard link support. [14]

In traditional embedded operating systems, memory space is considered as a single large file. Normally the space starts from DRAM and goes up to flash memory. Following Figure 15 shows the typical embedded system memory map. [14]

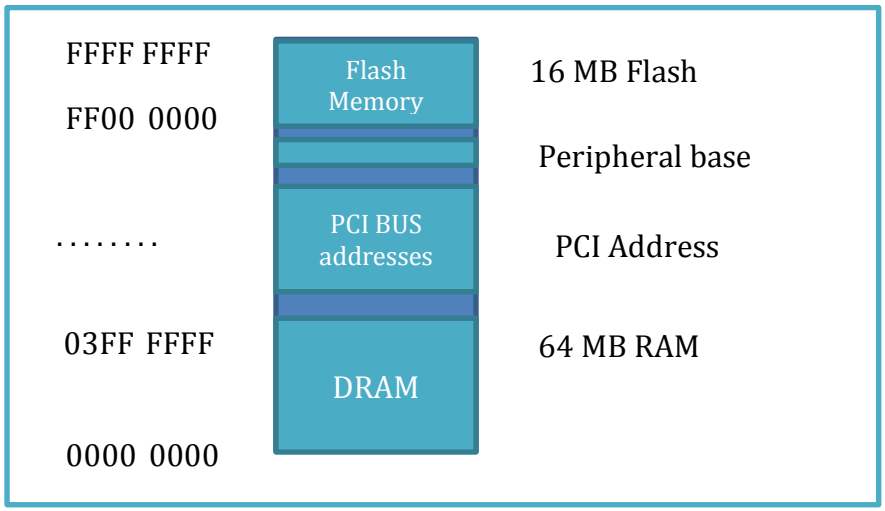


Figure 15. Typical embedded system memory map.

OS and all the tasks had equal access rights to resources, traditionally. Bug in one process could wipe out all memory contents. High performance microprocessors have Memory Management Units (MMU) for high degree of control over its address space and allocation to processes. It has two main forms i.e. access rights and virtualization of memory. [14]

Linux kernel takes advantage of MMU virtualization. Virtual memory has been beneficial as it allows more efficient use of physical memory by presenting more memory even though it wasn't present physically. Other benefit is that kernel can enforce access rights. [14]

Execution Contexts

First thing Linux perform is configuration of hardware MMU on processor and enables address translation. Kernel then runs in its own virtual memory space. In Linux there are two contexts i.e., kernel context and user space context. A thread can run either on kernel space or user space as application program. So, here right of access for any resource becomes important. User space program can access its own memory and also for device I/O and files or other privileged resources it must call a kernel system call. For example if application program request a data file, it needs to be context switch. For application program's read() command, C library request kernel. Inside kernel it results in hard drive access. Hard drive read request is issued asynchronously to hardware. Application program waits in queue and blocked unless data is available and when kernel receives interrupt upon data ready event it stops all processes and take the data. Application program then receive that data and exits from waiting queue. Following Figure 16 will elaborate the concept more. [14]

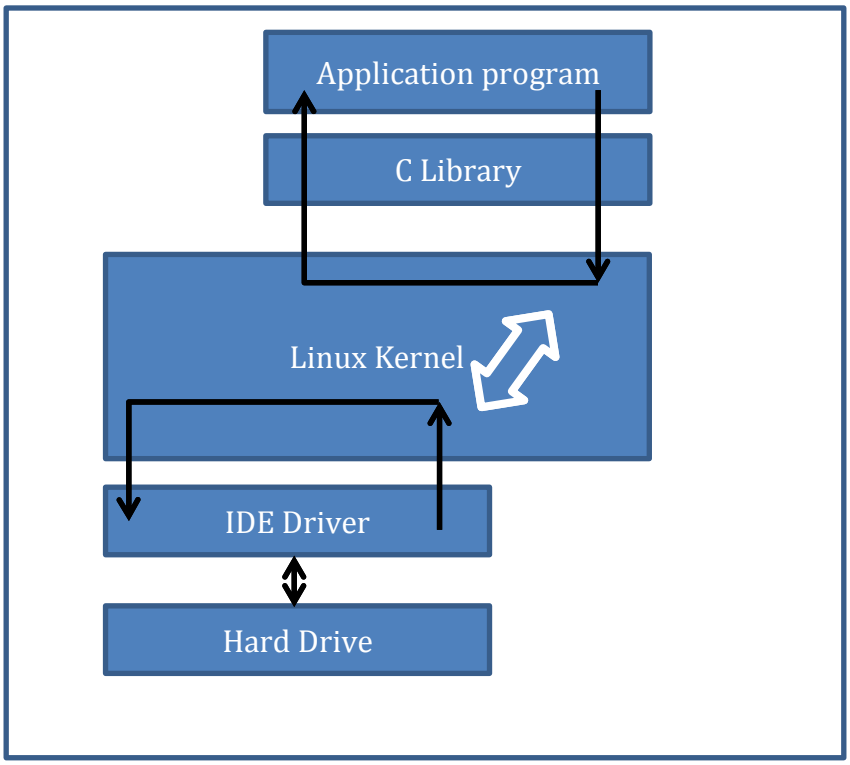


Figure 16. Simple File Read Request.

3.2. SOME LINUX DISTRIBUTIONS AVAILABLE FOR BB

In following, some of the Linux distributions which were considered by beagleboard.org for BeagleBone are summarized. It is noteworthy that the list of Linux distributions which are tested for BeagleBone is very large, the distributions which were very unique in some respect are described below, other are left due to time constraint.

3.2.1. ÅNGSTRÖM DISTRIBUTION

Angstrom Distribution is the one which is highly tested (for machine under discussion of the project i.e. BB), stable version and shipped with BeagleBone. It was a result of efforts made by a small group of people who worked for OpenEmbedded (software framework used for creating Linux distributions), OpenZaurus (Embedded System OS variant for BootStrap) and OpenSimpad (portable computer) to make stable distribution for embedded systems. Bone usually shipped with distribution having cloud9 IDE. For ease and lack of experimental time this distribution was chosen in Project. [16]

3.2.2. ARCH LINUX ARM PORT

A distribution for ARM computers, it claims to be simple, optimized and always up-to-date. Simplicity and user-centrism is philosophy carried out by Arch Linux and users are given complete instructions, control, and responsibility over system. Optimized packages for soft-float (software floating) ARMv5te, and hard-float (on chip floating point unit) ARMv6 and ARMv7 instruction sets are provided to take full advantage of each system. Software versions are up to date as daily update is available. [17]

3.2.3. NERVES – ERLANG/OTP

One can write one's own firmware using Erlang/OTP in cross compiled environment for small/medium size embedded devices. This is a very different concept from above mentioned distributions.

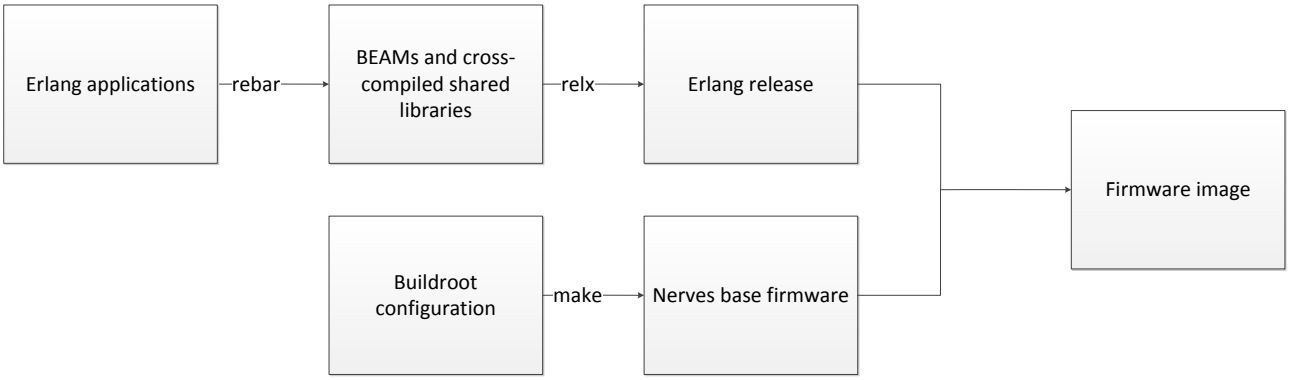


Figure 17. Erlang-Nerves Development.

In above Figure 17, the procedure has been described as 'Erlang Release' came as result of Erlang build tools Rebar and Relx is packaged into bootable all in one firmware image. [18]

4. SYSTEM STRUCTURE FOR DEMONSTRATION DEVICE

In this chapter, hardware design considerations for a device capable of demonstrating printed electronic components which were described in Table 1 of Chapter 2. Figure 18 shows the block diagram for system.

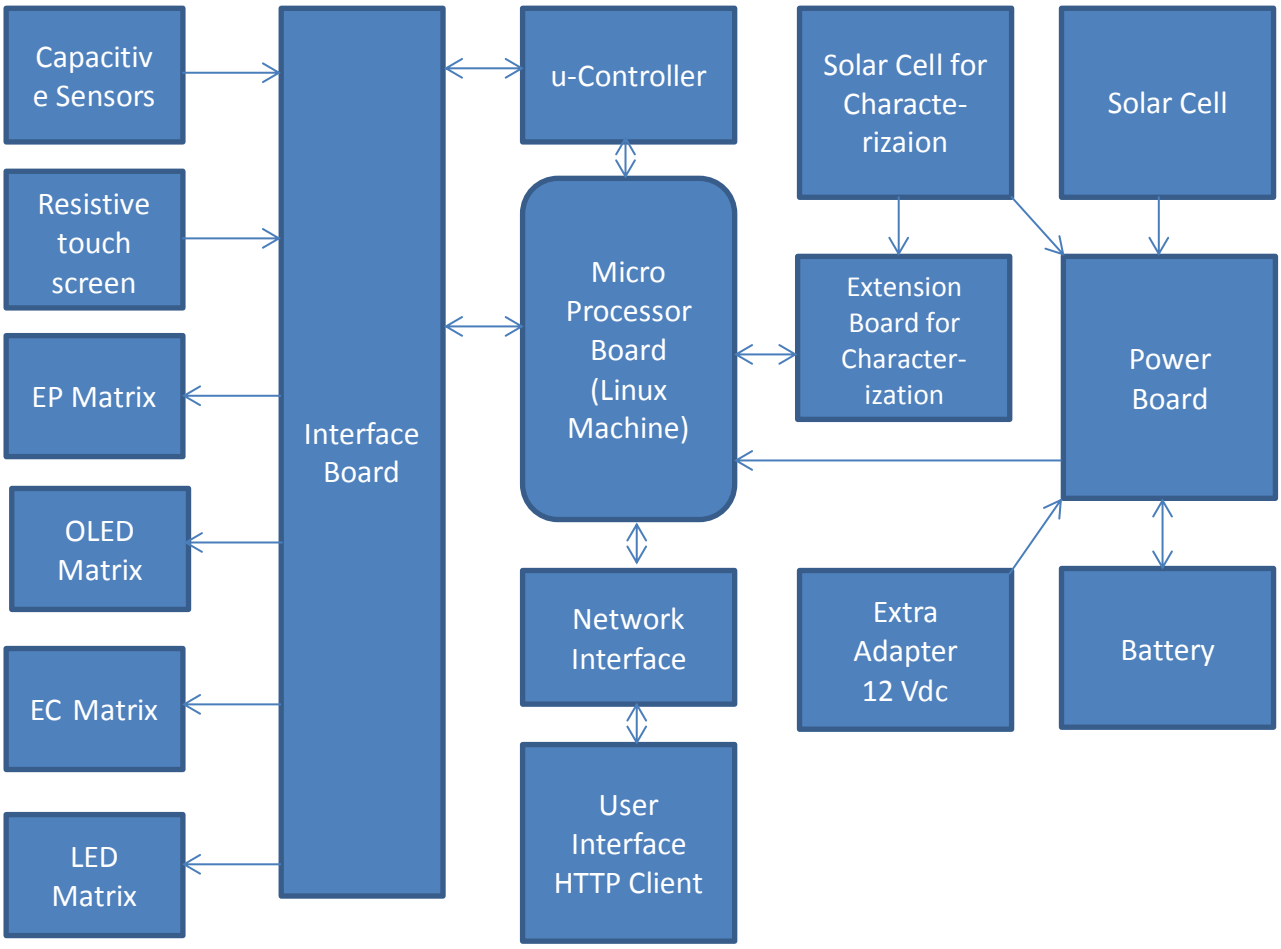


Figure 18. Block level diagram of Complete System.

Microprocessor board or embedded Linux machine is the primary processing unit in the system controlling almost all functions implemented. Interface board (IB) is has sub-controller to interface each printed component e.g., AD7147 controller for capacitive sensor matrix, and TSC2003 controller for resistive touch screen etc. These sub controllers will be detailed later. All of those sub-controllers on IB have capability to communicate with master device (microprocessor or microcontroller) using common I2C or SPI bus present on IB. Microcontroller was added in design as secondary processing unit. It was thought to be advantageous in power saving in later stage of the system development. Both microcontroller and microprocessor board have capability to communicate with IB; therefore, after initial configuration if microprocessor board is turned off, it can save allot of power as most power consuming unit in above system is embedded Linux machine or BeagleBone. User interface is provided through HTTP client. BeagleBone has been programmed to run an HTTP server and communicate with microcontroller to control IB through it. An I-V curve tracing board was also connected with BeagleBone to provide I-V curves of printed solar cells. It was thought, that it might route power output from measured solar cell to power board in

suitable conditions. Finally the power board is there to take input power from solar cell or a dc adapter, and provide stable 1A @ 5V to system.

A Li-Ion 3.78 V 6.8AH battery was used for backup, which can provide more than 10 hours standby @ 500 mA 5 V systems. Hardware boards are discussed in detail in following sections of the chapter.

Complete electrical characteristics of input and output pins are given in appendix 7. Appendix 8 contains complete hardware design (schematics). Appendix 4 includes connection details for all of the boards. Final snap shot of device is provided in Appendix 9.

4.1. BEAGLEBONE (MICROPROCESSOR BOARD)

[19] describes BeagleBone ARM Cortex A8 based processor board, designed for open source community. Using add on boards called Capes it is highly extendable. As it is addressed to open source community, design information is freely available which is very good for extensive development of systems. Following Table 2 allows us to look in some of specs for subjected machine and its close competitor i.e. Raspberry Pi. BeagleBone black was latest review of BeagleBone board and both were used in project.

Table 2. Specifications of BB, BBB & Raspberry Pi

	Feature (BeagleBone) [19]	Feature (BeagleBone Black)	RaspBerry Pi (close competitor) [22]
Processor	AM3359 500 MHZ-USB Powered 720 MHZ-DC Powered	Sitara XAM3359AZCZ100 Cortex A8 ARM 1 GHZ [20]	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)
Memory	256 MB DDR2 400 MHZ	512 MB DDR3L 400 MHZ	Model A: 256MB , M B: 512 MB
IO	GPIO x66 1.8 V 100 S/s A/D Converter x8 PWM outputs x8 UART x6 I2C x3 SPI x2 McASP x2 CAN x2 USB 2.0 HS OTG+PHY x2	GPIO (69), McASP, SPI, I2C, ADC 7, Timers (4), Serial Ports (4), CAN [20]	GPIO x8, UART, I2C bus, SPI, I2S HDMI, 3.5 mm jack (audio)
Power Consumption	502 mA@5V at UBoot for USB 305 mA@5V at Kernel Idling for USB	210 to 460 mA@5V [20]	300 mA, 700 mA @5V
Power	5V rail, 3.3V rail 1.8 V rail for ADC	Same	+3.3 V, +5 V, ground
OS	Originally shipped with Linux Angstrom	Soon following distributions will be	Linux (Raspbian, Debian

	Distribution, Ubuntu Android or third party solutions are also compatible	supported: • Texas Instruments releases: Android, Linux, StarterWare (no OS) • Linux: Angstrom Distribution, Ubuntu, Debian, ArchLinux, Gentoo, Sabayon, Buildroot, Erlang, Fedora • Other: QNX, FreeBSD [21]	GNU/Linux, OpenELEC ,Fedora, Arch Linux ARM, Gentoo) RISC OS, FreeBSD, NetBSD, Plan 9, Openwrt
Price	89 $	45 $	25 $ / 35 $

From the above table it can be explained why BB was preferred on RaspBerry Pi a close competitor of board. There were some other options too but these two boards are unique in processing speed, power consumption, board size and community etc. So here it will be presented why BB was thought to be superior over Pi for this project. First point is that BB has a lot of General Purpose IOs (GPIO), which is really a great point for extendable system. Not only GPIOs communication ports/busses were available in greater number then Pi. Thirdly Pi is attractive package to those who are looking for more like a general purpose solution with video, audio jacks and usb ports for mouse and keyboard interface but lesser GPIOs rather than typical embedded system without any displays and audio jacks but with a lot of IO options; it is capable of being part of embedded systems though. Online community wise and price wise Pi had advantage which was gracefully catered by no. of GPIOs provided plus ease of hardware integration in system.

4.2. POWER BOARD

Power Board consists of two parts: Solar Battery Charger and DC Boost Converter.

4.2.1. CHARGER

Linear Technology's LT3652 was used for the purpose. [23] describes LT3652 as "Power Tracking 2A Battery Charger for Solar Power". It gives very powerful description for design considerations of charge voltage, Charge Current and Ripple Current (ΔI). LT3653 is monolithic step down battery charger with input range from 4.95 V to 32 V. Maximum charge current can be 2A while it provide Continuous Current and Continuous Voltage charge characteristic. It can be configured to terminate charging if current falls below 1/10 of programmed maximum called C/10. Auto recharge starts if charge voltage drops 2.5% of programmed value. Hence many more system protections are provided in this small package making it very smooth choice for remote applications.
Following Design equations can be used to select sense resistance R4, inductance L1 etc. as shown in Figure 19 for desired voltage level, ripple current (achieved within limits by selecting suitable R4 and L1) shown in Figure 20. Equations (1) to (6) represent design solution for LTC3652.

End of Cycle timer Programming:

$$t_{EOC} = C_{Timer} \bullet 4.4 \bullet 10^6 \quad (1)$$

Typically .68 µF capacitor is used to generate EOC of three hours. We used end of charging (EOC) to be two hours for specific battery with 2A maximum charge current leading to 1µF as C_{Timer} (Figure 20).

$$R_{FB1} = (V_{BAT(FLT)} \bullet 2.5 \bullet 10^5)/3.3 \quad (2)$$

$$R_{FB2} = (R1 \bullet 2.5 \bullet 10^5)/(R1 - (2.5 \bullet 10^5)) \quad (3)$$

R_{FB1} (R5) & R_{FB2} (R6) are enough to produce desired $V_{BAT(FLT)}$ (VBAT) but for required equivalent resistance of 250 another resistor R_{FB3} (R7) is introduced, depicted in Figure 19.

$$R_{FB3} = 250k - R_{FB1} \| R_{FB2} \quad (4)$$

$$R_{sense} = \frac{0.1}{I_{CHG(MAX)}} (\Omega) \quad (5)$$

$$L = \left(\frac{10R_{sense}}{\Delta I_{MAX}}\right) \bullet V_{BAT(FLT)} \bullet \left[1 - \left(\frac{V_{BAT(FLT)}}{V_{IN(MAX}}\right)\right] (\mu H) \quad (6)$$

Where it is considered: $.25 < \Delta I < .35$. Solving the above equations for Vin 12 to 15 Vdc and Vout 3.78 Vdc, following are the design values

C_{Timer} = 1µF ,R_{FB1} =30 kΩ, R_{FB2}=330 kΩ, R_{FB3}=223 kΩ, R_{sense}=50 mΩ, L = 5.6 µH as depicted in Figure 19.

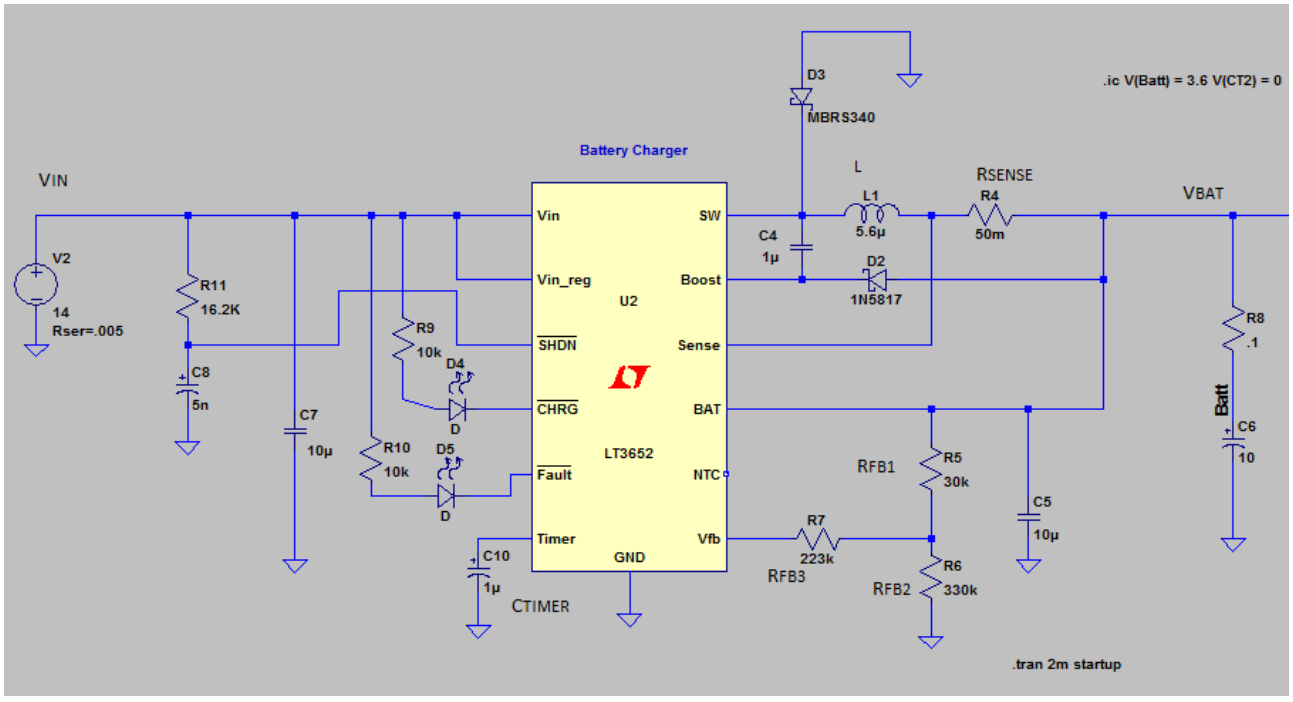


Figure 19. Schematic of solar charger part (LT SPICE Environment).

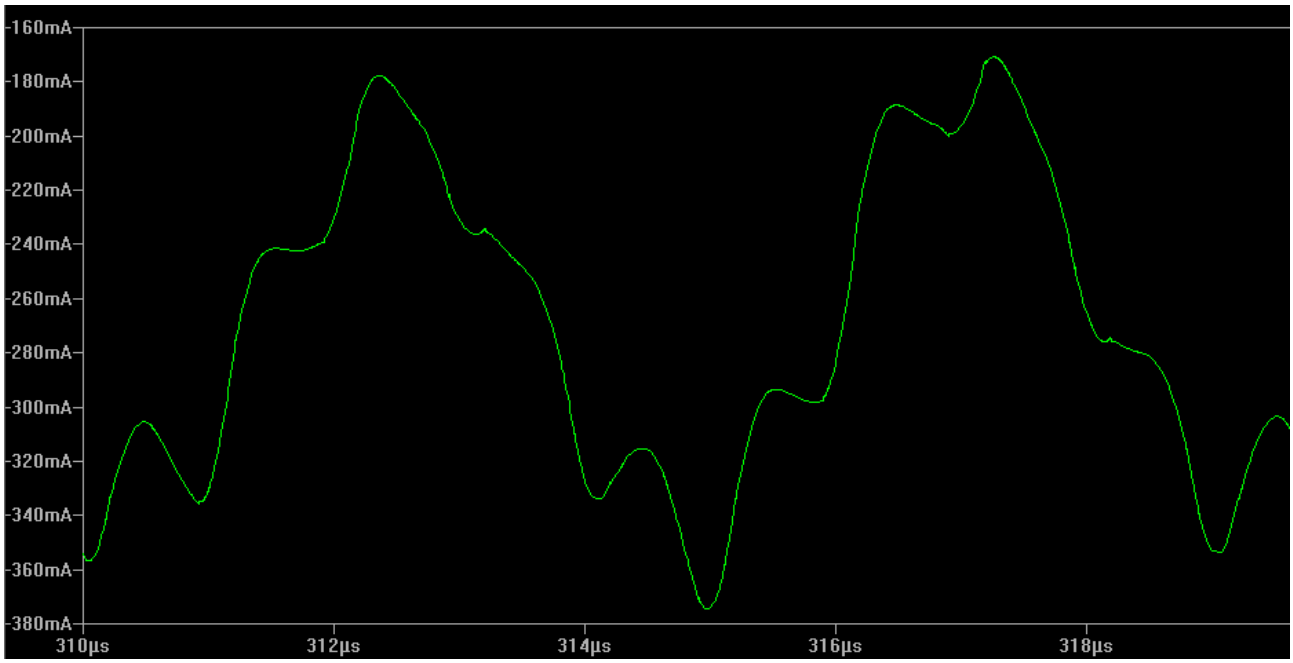


Figure 20. Ripples of Charge Current within limit of 35% [-378 mA] with V_{IN} = 14 V (LT SPICE simulation).

Alternative of LT3652 can be LT3650-4.1/LT3650-4.2 which are described as Monolithic 2A Switch Mode 1-Cell Li-Ion battery Chargers, with typical float voltages 4.1 and 4.2 respectively which was not suitable for battery with float voltage 3.78 Vdc used in project whereas float voltage can be programmed up to 14.4V in LT3652. LT3650-8.2/LT3650-8.4 Monolithic 2A Switch Mode 2-Cell Li-Ion Battery Charger with input voltage ≥ 9V and typical float voltages 8.2 or 8.4 Vdc can also be used but flexibility for battery choice and input voltage range would be sacrificed.

4.2.3. BOOST CONVERTER

Output from solar charger controller LT3652 goes to battery to be charged and to the DC conversion unit of the power board, as power required to most of boards is 5Vdc. Other voltage levels can be satisfied by voltage rails provided by BeagleBone itself.
For DC conversion LT1935 was used and document [24] describes it as 1.2 MHz Boost Dc/Dc Converter in ThinSOT with 2A Switch. Furthermore it illustrates its design parameters as follow. This IC has high up conversion voltage up to 38 V. Stable 5V at 1A from 3.3 V is promised in its datasheet, which seems promising for system used in project, which least probable to go above 1A@5Vdc. It has constant frequency, internally compensated current mode pulse width modulation (PWM) architecture which makes very low output noise. Low ESR ceramic capacitor thus can be used at output to prevent it. Document [24, Page 4] describes operation with the help of block diagram, interested one can read it. For design parameters, one has to find out value of Inductor (for controlling ripple current) used and also feedback resistor values are required (to control average output voltage). Following calculations help in finding requirements.
Assumption: Ripple Current is 1/3 of maximum current. Considering L = L1, R1 = R1, & R2= R2 from Figure 21 & 22, Equations (7), (8), & (9) represent design solution for LT1935.

$$L = \frac{3\left(\frac{V_{in}}{V_{out}}\right)(V_{out}-V_{in})}{I_{max}*f}) \quad (7)$$

$$I_{max} = 2\ A \tag{8}$$

$$V_{out} = 1.265V\left(1 + \frac{R_1}{R_2}\right) \tag{9}$$

[24, pages 5-6] describes tradeoff between settling time and ripples in output voltage by changing output capacitor. Also layout considerations are given as high speed operation demands careful placement of components and large ground plane. Following is the suggested layout in the referenced document. Figures 22 to 26 show recommended layout, schematic for LT1935; actual schematic, LT SPICE simulation results and final layout of complete power board.

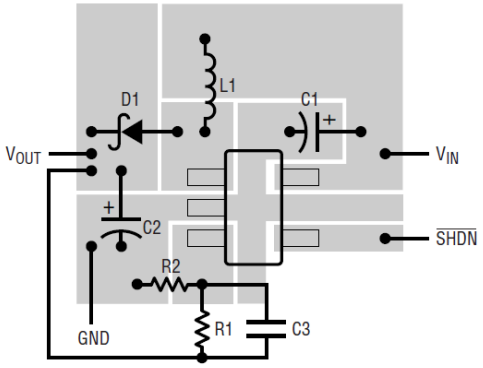


Figure 21. Suggested Layout for Boost Converter.

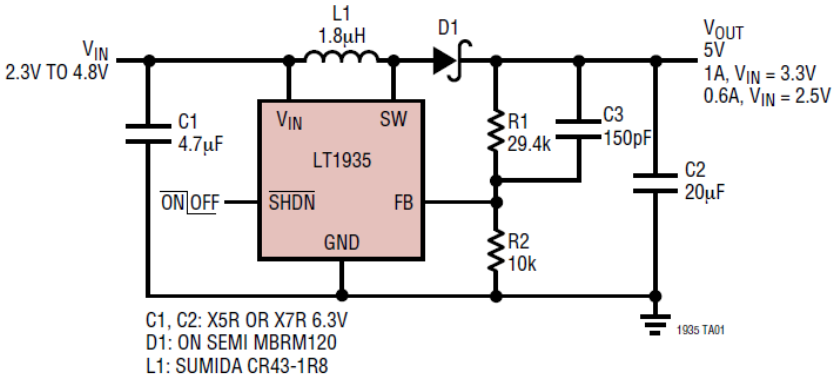


Figure 22. Typical 5V Boost Converter.

Figures 23 to 25 gives schematic of power board designed in LT SPICE, simulation in same environment, and two-layer layout designed in EAGLE. All Printed Circuit Boards (PCB) were subject to line width > 150 µm, isolation > 210 µm, footprint > 100 µm, & minimum drill = .5 mm (500 µm) on 1.6 and .8 mm thick FR4 boards. Size of power board was 40 x 60 mm^2.

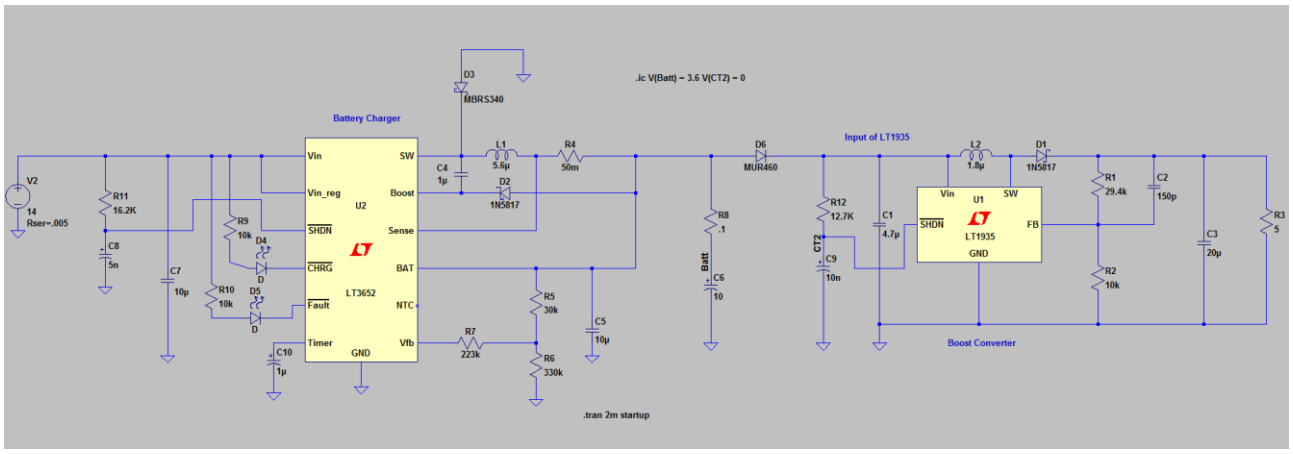


Figure 23. Complete Power Board.

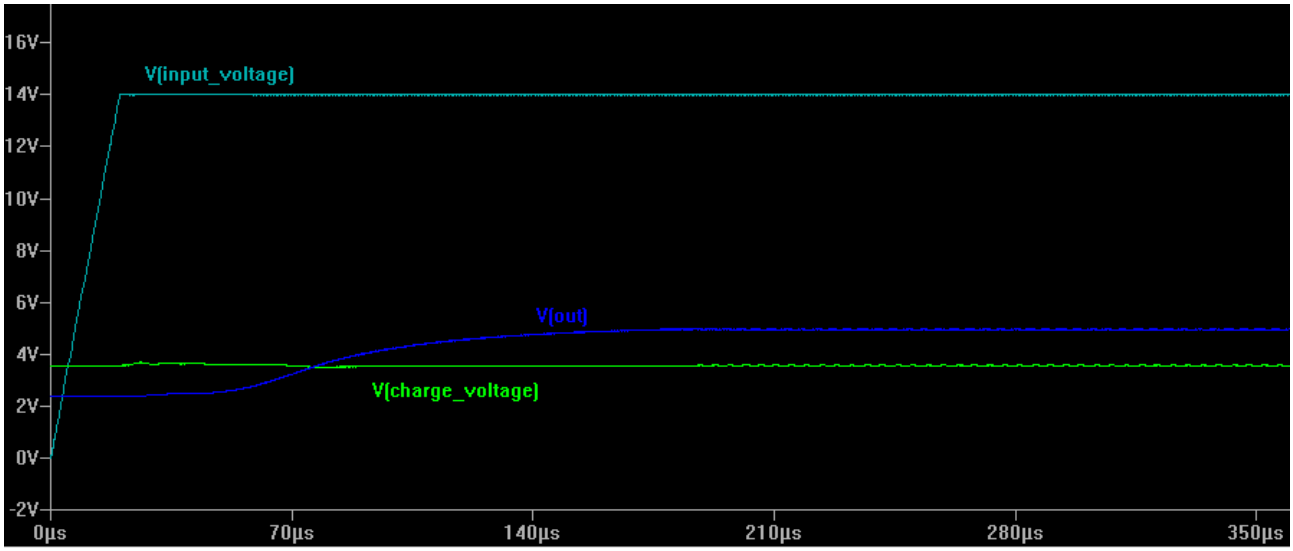


Figure 24. LT SPICE Simulation of Charge Voltage and Output Voltage.

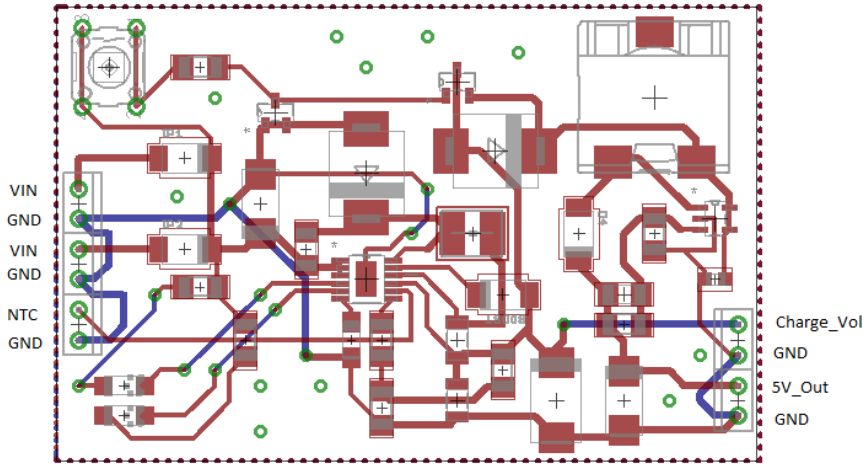


Figure 25. Final Layout of Power Board.

Seeing power requirements of BB and Arduino Board, one can confidently conclude: 1A @5V for power line feeding IB, BB & Arduino Mini Board should be enough. So options with load current

equal to or more than 1A @5 V serve the purpose. LT1935 Guarantees over 85% efficiency for 1A output current & Vout = 5 V with Vin = 3.3 V (where as Vin is 3.78 V in this case). In LTC3402 guarantees power efficiency at 1A @ 5V output from 3.6 V Input is above 85%. Price, easy connections and packaging (5-TSOT-23 vs 10-MSOP) made clear advantage for LT1935. Another High efficiency DC converter LTC3425 for higher rating of current wasn't suitable because of its QFN package and complex circuit structure.

4.3. INTERFACE BOARD

Interface board has been the most important part of this project. Not only hardware design consumed much of total design time but software coding was completely dependent to it. Following are the main parts of Interface Board:

- 13 Capacitive Sensors
- 4 wire Resistive Touch Sensor
- LED current driver (10 x 10) (20 outputs)
- LED voltage driver (10 x 10) (20 outputs) (Voltage = 0 to 15 VDC)
- LED Voltage drivers (2) (Voltage = 0 to 5 VDC)

In following text, all parts of the interface board will be discussed separately.

4.3.1. CAPACITIVE SENSOR

Here, capacitance measurement of proximity sensors is serving the purpose; its concept is illustrated in Figure 29. Proximity sensor phenomenon is not rare in embedded systems and many ready-made solutions are available; for example, Arduino Pro Mini (micro controller board) with ATmega328 already have mechanism to sense proximity through capacitance measurement in "CapacitiveSensor" [25] C++ library and "Atmel QTouch Library" [26].

Though CapacitiveSenor library provided for Arduino boards is specifically meant for exact type of sensors which were used in this project, but tests conducted proved its performance was dissatisfactory in terms of stability and reliability, otherwise these standard methods are very quick in implementation.

Stable measurements can be done by CapTouch programmable controller for single-electrode capacitance sensors: AD7147. Though printed capacitance sensors cannot be considered very reliable, environmental effects are not negligible; which cause measurement level shifts for some time span. [27] describes functionality of the controller and programming references. There are two version of capacitance-to-digital converter (CDC) IC AD7147 with SPI & AD7147-1 with I2C communication compatibility. For ease of programming and uniformity of design most of the components were chosen for I2C unless other communication standard appeared to be a must.

This CapTouch controller is intended for implementations of functionalities like buttons and scroll bar wheel. It has 13 input channels; final value of each channel is converted to digital through 250 kHz sigma delta ($\sum$-Δ) converter. It's actually intended for a grounded sensor. To minimize noise pick up a shield output is also provided. It has calibration logic to compensate for environment changes. This is well suited for 3.3 V, and above 4 V supply should be avoided, though in design considerations of this project only stable 5 V were generated through power board; therefore, we can use desired voltage level from BeagleBone 3.3V rail (Pin 3 & 4 of header P9).

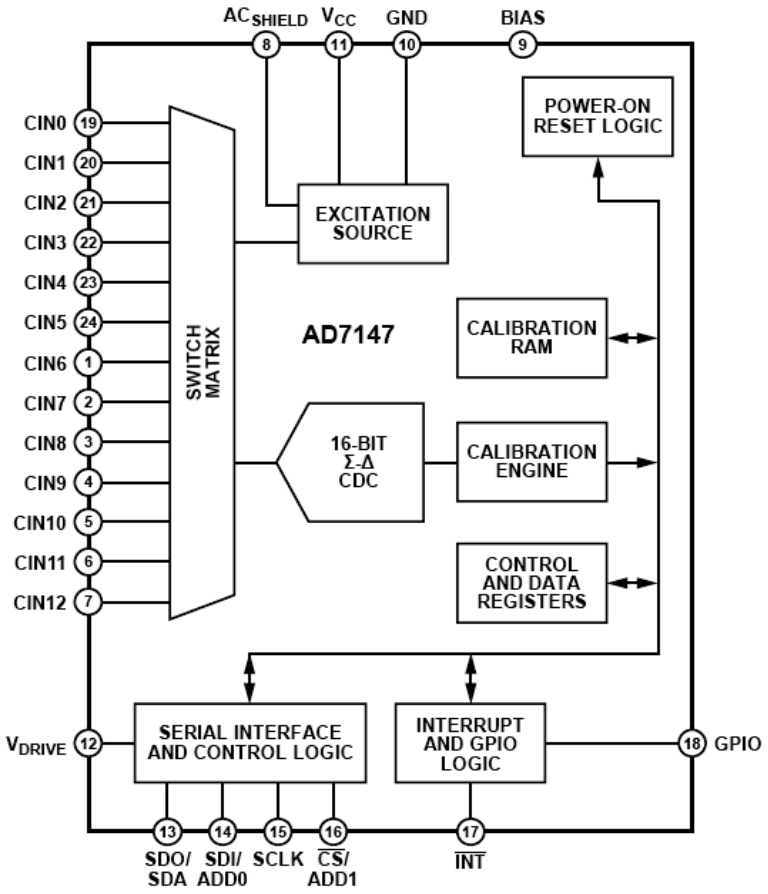


Figure 26. Functional Block Diagram of CDC IC.

In above Figure 26, one can see the blocks of this controller to familiarize oneself with IOs and functionalities of the controller. Actually main task of designer is to understand how switch matrix is configured, and for which state what kind of control parameters are to be adjusted? Much of this will be explained in next chapter. In this chapter stage and CDC ports are defined as below. For ease, stage can be defined as a time slot: with total of 12 stages this CDC timing architecture. Configuration (i.e. how many and which capacitive inputs are to be connected to CDC in which stage?) for each stage can be done through programming microcontroller to send specific I2C communication messages. There is one CDC, which is a 16 bits converter with differential input. Thirteen capacitance inputs are named as CIN0 to CIN12, as it is depicted in Figure 27. At each stage a number of capacitance inputs can be attached to either input or port of differential input of CDC. Normally one sensor is attached to positive or negative port. This will lead to measure twelve sensors in twelve stages STAGE0 to STAGE11. If one input is attached to positive and one sensor is attached to negative at one stage, then twelve sensors can be measured in six stages. If one capacitive senor connected to positive port is touched (or human body is in proximity), measurement goes from 32767 to 65535 in decimal notation. If negatively connected senor is touched than measurement result in value smaller than 32767 up to 0 depending on sensitivity configured and how far is the object having some connection with ground (this rule goes with positively connected sensor as well). If both sensors are touched for example (they should cancel each other's affect), value should remain near 32767.

In Figure 27 an example has been portrayed, two sensors are connected in different stages i.e., STAGE0 & STAGE1. CIN3 is connected to positive terminal of differential CDC in STAGE0. One can access a register to find measurement result of STAGE0 which corresponds to measurement of CIN3. Similarly one can access a register to find measurement result of STAGE1 to find measurement of CIN10. If at the time of measurement CIN10 is touched, STAGE1 measurement value must be below 32767 to 0 depending on other sensitivity configurations.

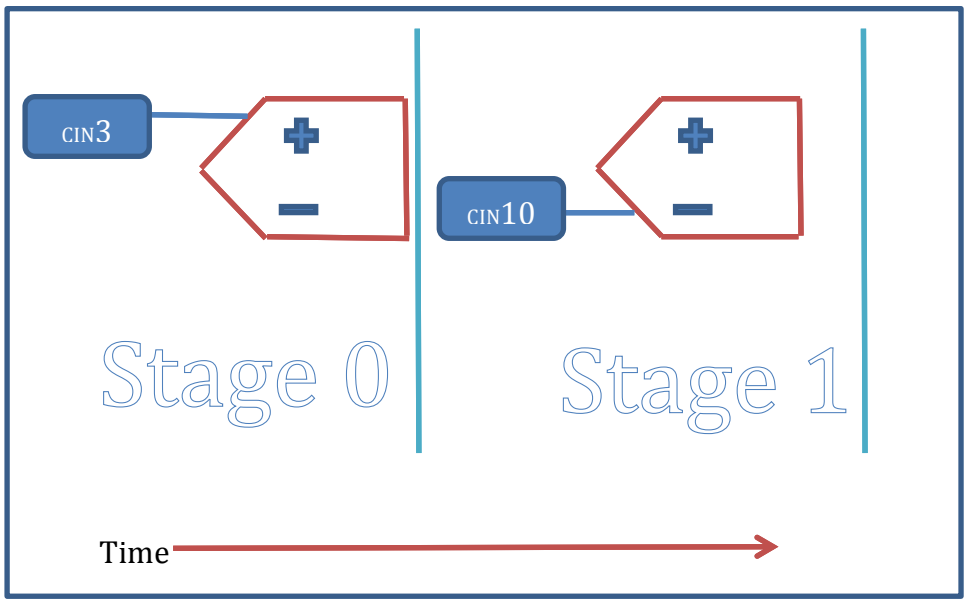


Figure 27: Example of CIN connections.

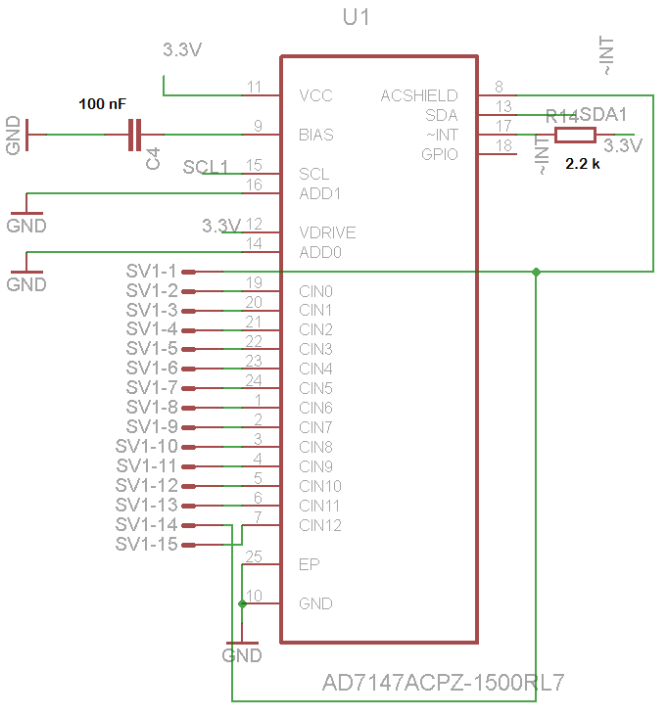


Figure 28: Schematic of CDC (EAGLE software environment).

Figure 28 shows capacitance sensor controller, with capacitance range: typically ±8 pF for the controller. Table 3 generated through reference manuals and IC prices available at fi.farnell.com [28].

Table 3. Comparison of alternative solutions for Capacitive Sensing

IC / Technique	No. of Channels	Typical Capacitance	Tested for printed Capacitive Sensors	Update	Price
AD7147	13 / IC	8 pF	Worked	9 ms	3,74 € / IC
Capacitive Sensing Library (Arduino)	2 GPIO pins / Channel	-	Unstable for Printed Sensors	~ ms	Available with Arduino Mini Board
AD7150	2 / IC	up to 13 pF	-	10 ms	3,98 € / IC
AD7747	2 / IC	8 pF	-	5 Hz to 45 Hz	11,90 € / IC
AD7151	1/ IC	up to 13 pF	-	10 ms	5,31 € / IC

Above table clearly depicts that most convenient option for a small and less expensive IB was AD7147.

Issues with Capacitive Sensor Controller

This kind of measurement is very sensitive to environment change. It might require some sort of threshold (of detection) change with the change of environment or programmer has to write a code to nullify environment effect using some good algorithms in application layer at least. In case of AD7147, there are certain cases where controller nullify environmental effects itself; for instance, if user hovers over the proximity sensor controller forces a recalibration to ensure proximity flag is not set. But that was not sufficient practically on application level to have good results. One can configure AD7147 for different settings of calibration and proximity detection.

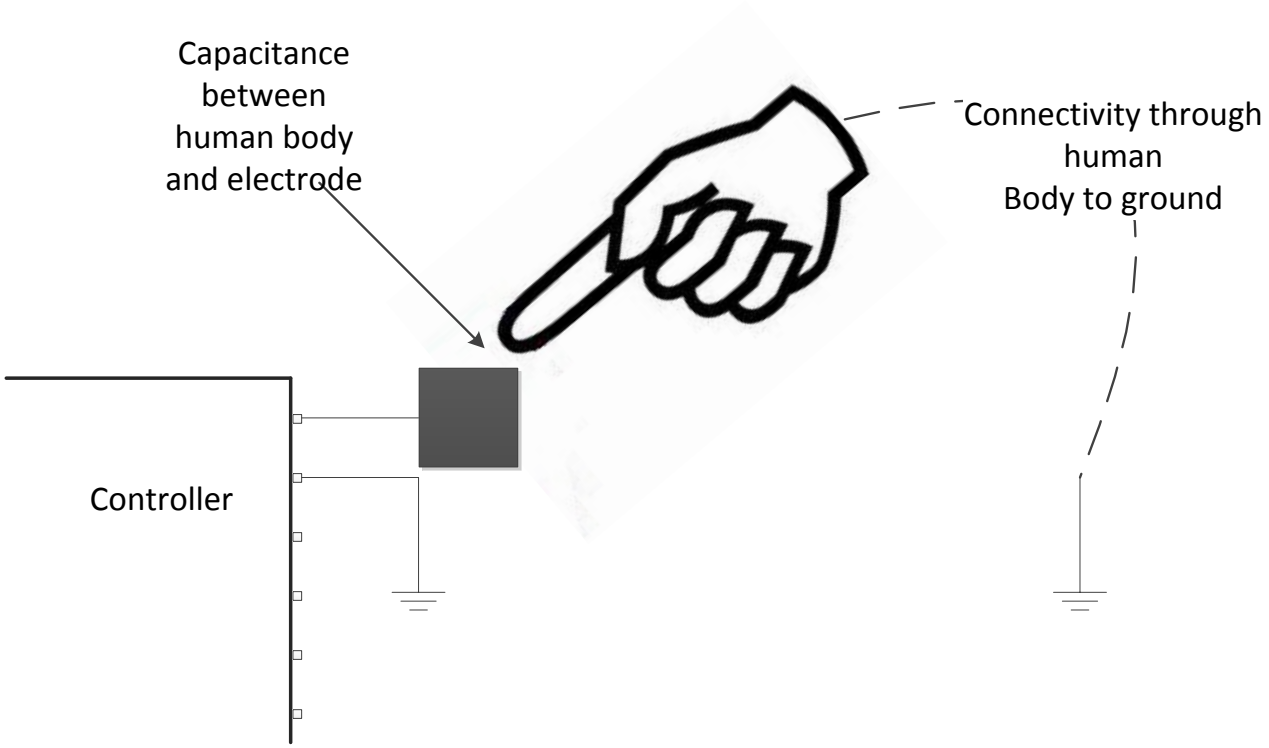


Figure 29: General Mechanism of detecting proximity through simple printed electrode.

A general problem comes with the nature of measurement; i.e., proximity sensing via capacitance measurement from (a) electrode on printed sensing board to (b) ground through human body. Figure 29 depicts how it works by creating some capacitance between sensor and human body; therefore, a perfect grounding and zero potential is required between controller's ground pin and actual ground surface where person is standing for ideal system, which is difficult to achieve, but still the technique works without perfectness of ground potential.

4.3.2. RESISTANCE TOUCH SCREEN CONTROLLER

In this part, requirement was to measure resistance of a four port touch sensor. Equivalent resistance model of a four-port touch screen is shown in following Figure 31: four port model's connections with pin 2, 3, 4, & 5. In [29] one can get the specification of TSC2003: an I2C compatible, 4-wire touch screen controller. There are some additional IOs provided in controller which will not be discussed under this section, and interested reader is referred to datasheet for its detail. Following Figure 30 taken from document [29] describes it all.

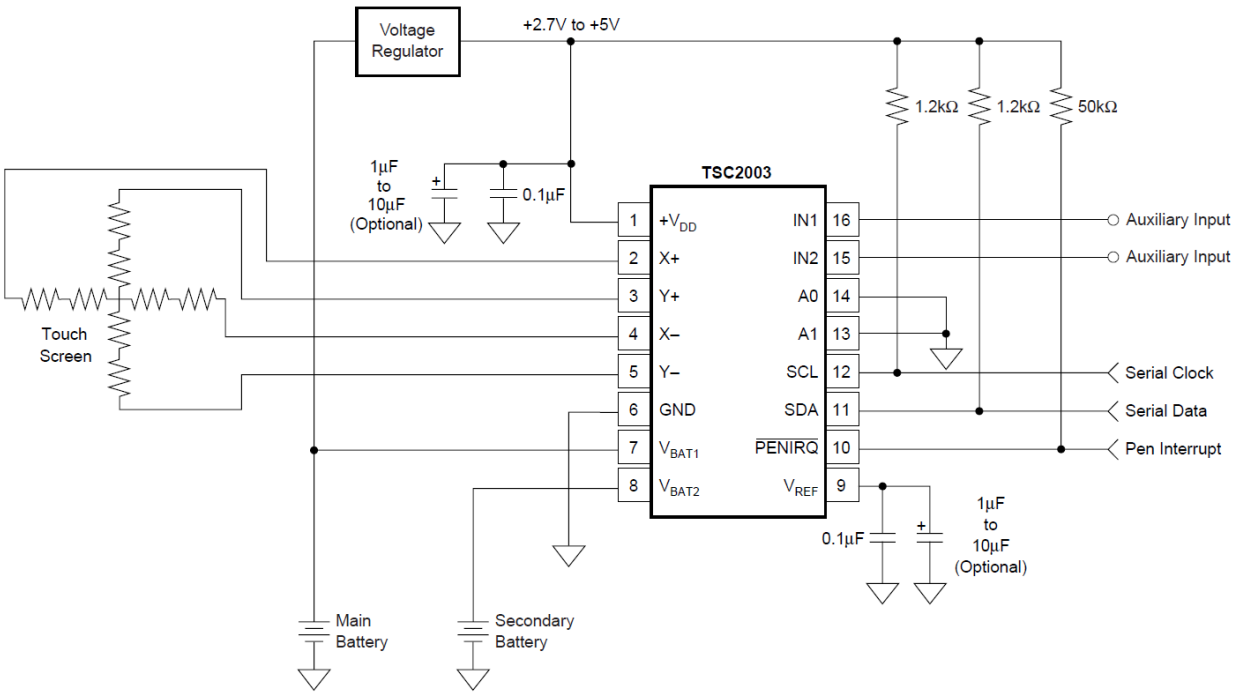


Figure 30. Basic Operation of TSC2003.

This controller utilizes successive approximation register (SAR) ADC (for measurement of voltage). Four wires: X+, Y+, X-, & Y- ports, are used for approximating coordinates; i.e., X, Y, Z1 & Z2. In some cases only two of the coordinates are enough to define the touch position. In short, measurement process can be seen as controller applying differential or single ended voltages on different ports and measuring voltage on other ports; hence, finding the resistances (shown in touch screen equivalent model in Figure 30), and approximating the position using resistance measurements. The touch screen equivalent circuit is formed as there are two resistive layers: one represents Y plane, and other represents X plane. If one touch the screen at a point, both X-plane layer and Y-plane layer meet at that point, forming unique resistive node for touching a unique point in screen. Controller has 25pF (charging) capacitor which form RC circuit, fit for measurement of charging time which is low (fit for high throughput rate such as 50 KSPS) if resistance node is of few hundred milliohms. Terminal resistances in touch screen are in fact few hundred milliohms [30]. Figure 31 represents the schematic drawn in EAGLE environment.

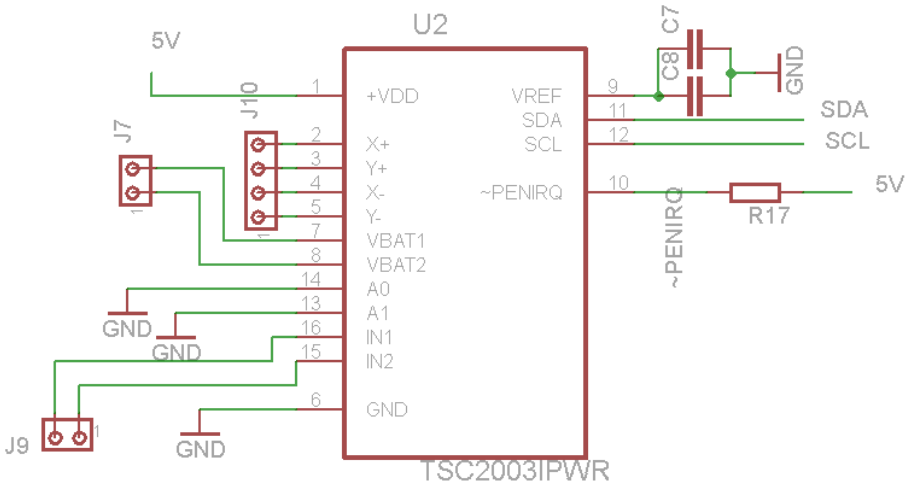


Figure 31. Connection detail of Resistive Touch Screen Controller on Interface Board.

Following Table 4 generated through reference manuals and IC prices available at fi.farnell.com [28].

Table 4. Comparison of alternatives for resistive touch screen controller

IC	No. of Wire	Conversion Rate	Interface	Resolution	Price	Additional
ADS7845 (TI)	5	125 kHz	Serial	8 or 12 bit	12,61 €	
AD7873 (AD)	4	25 kSPS	Serial	8 or 12 bit	4,34 €	
AR1000 SERIES (MicroChip)	4,5 or 8	140 Reports/s Sensor dependent	Serial, I2C, UART	10 bit (max)	1,37 € - 4,09 €	
TSC2004 (TI)	4	50 kSPS	I2C	10 or 12 bit	5,13 €	AUTO POWER DOWN
TSC2007 (TI)	4	20kHz (8-Bit) or 10kHz (12-Bit)	I2C	8 or 12 Bit	4,43 €	
TSC2003 (TI)	4	50 kSPS	I2C	8 or 12 bit	7,01 € (4,88 € on purchase date)	AUTO POWER DOWN

From the above table, TSC2003 had advantage of low power, high throughput, ease of programming, circuit design (I2C was preferred throughout over Serial plus TI TSC series had simple circuit design characteristic), and low price for 4 wire touch screen control.

4.3.3. CURRENT BASED LED DRIVER:

Maxim, 2-wire interfaced 4-Digit 5 x 7 Matrix LED Display Driver (MAX6953) was used for this purpose. In [31], one can find following (and more) important details regarding this current controller for led matrix display. This controller is a cathode-row display driver which enables 5 x 7 dot-matrix led displays interfaced with microprocessor or microcontroller. Initially intended for four display words (alphabets and numbers: 'a', 'b', 10 etc., are stored in RAMs), which are 5 x 7 in dimension. But it can be programmed so, to display any (random) image. There are 104 ASCII characters, column and row drivers, static RAM that store each digit, and 24 user defined characters available in this controller. Those user defined characters can be used to store image. For power characteristics of controller one must be referred to the datasheet. But important properties of power control of LED are: 16-Step Digital Brightness Control, max. source current 50 mA, and it can be programmed (via C_{set} andR_{set}) to any value below the max. Following is the typical application circuit described by [31] with edition of digits used in this project. In Figure 32, it is shown Digit 0,1 are used fully; Digit 2, 3 are used partially to fulfil requirement of 10 x 10 matrix.

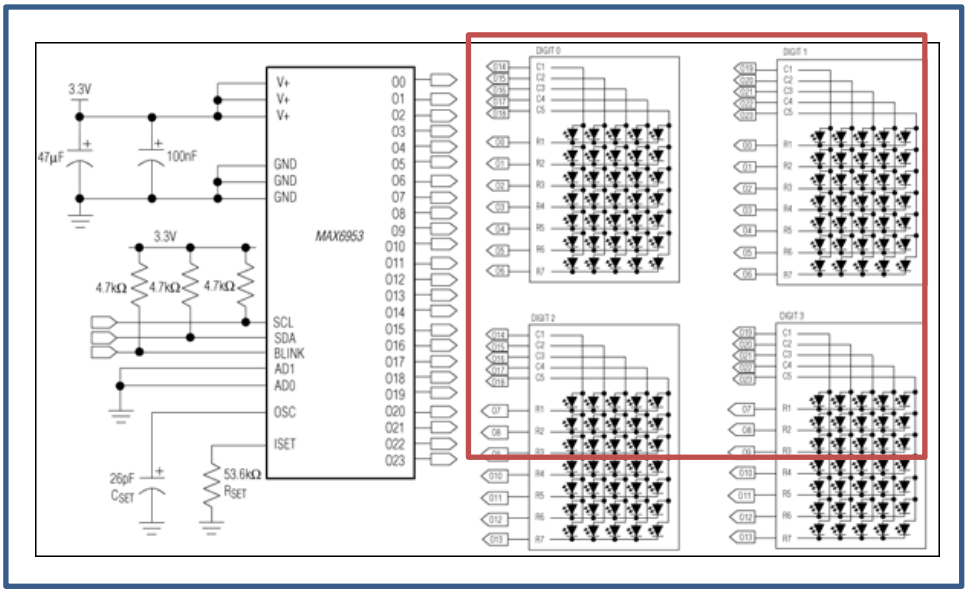


Figure 32: 10 x 10 Matrix taken from Typical Application Diagram used in Project

Moreover detail of how registers and RAM data are formed is also provided in referenced document, which will be used in next chapter for software structure. Following equations i.e., (10) & (11) show how to set value of resistance and capacitance for maximum value segment (led) current (I_{SEG}).

$$I_{SEG} = \frac{K_I}{R_{SET}}\ mA \tag{10}$$

$$f_{OSC} = \frac{K_F}{R_{SET} \times (C_{SET} + C_{STRAY})} MHz \tag{11}$$

Allowed range of oscillating frequency (f_{OSC}) (in above configuration one is using external RC oscillator) is 1 to 8 MHz, and to use internal oscillator we can ground OSC pin. K_I & K_F are (current & frequency) constants. Values of the constants are: $K_I = 2144$ and$K_F = 6003$.

C_{STRAY} is stray capacitance from OSC pin to GND in pF, typically 2 pF and it depends on layout.

Following Figure 33 shows connection detail on Interface board, designed on EAGLE 6.4.0.

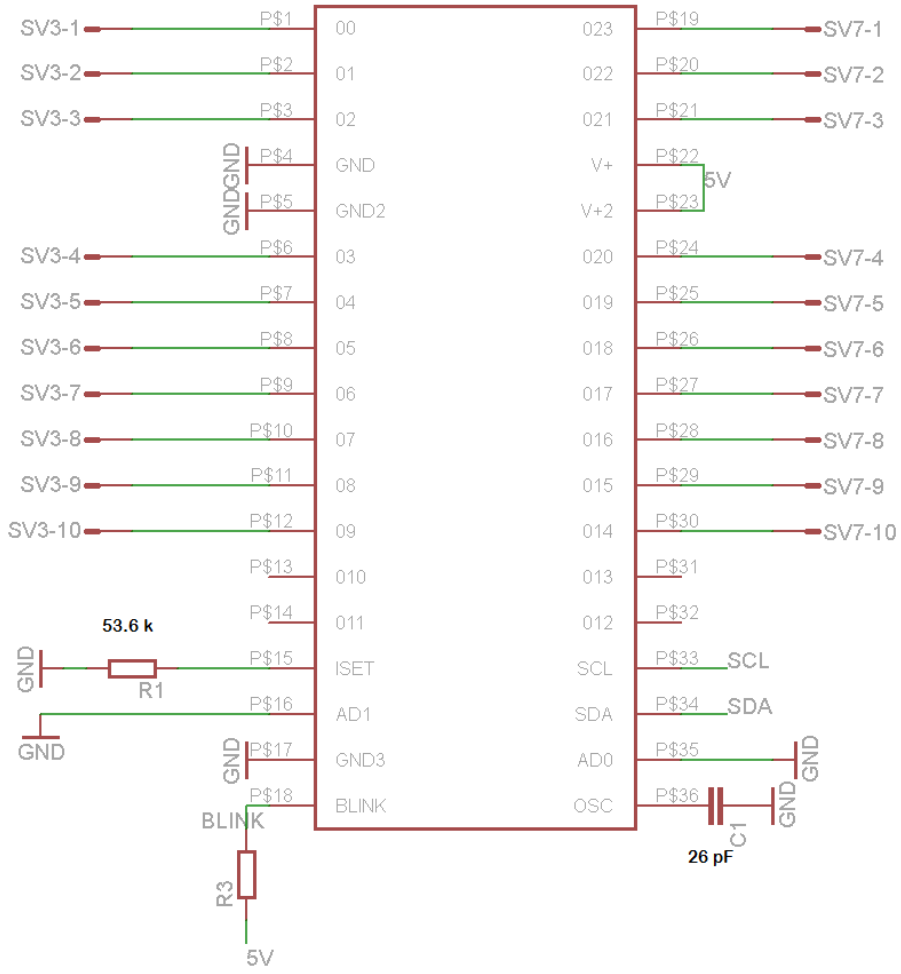


Figure 33. Connection detail of MAX6953 on Interface Board.

In MAX6953 peak segment current can be set 50 mA. After setting the maximum current it can be adjusted in 16 levels i.e., 0/16 to 15/16 of maximum through brightness control messages. This range of current can drive nearly all types of displays (described earlier), though voltage requirement might pose a restriction as segment voltage can only go a little higher than V+ (depicted in Figure 34) provided to power IC. There are not many solutions available for 10 x 10 current driven led matrices. TLC5920 [32] provides a good alternate for MAX6953 as it offers 16 x 8 dots control with single IC; however, two of those are required to drive at least 10 x 10 matrix display. In comparison, it provides 3 to 30 mA constant current output for segments; totaling 640 mA can be taken from one IC. Regarding segment voltage it has same characteristic as it can go a little higher than V+ provided to power IC which is normally 5 V for both of compared ICs. Though it is significantly low price solution, circuit design complexities for 10 x 10 matrices pose restrictions as board size was required to be as less as possible.

4.3.4. VOLTAGE BASED 10 X 10 DISPLAY MATRIX CONTROLLER

In this section, Simple electronic design methodology has been adopted to provide voltage control for printed display matrices. It consists of two parts: row driver and column driver. Column driver consists of amplified outputs from DACs, serve as current source. Row driver serves as current

sink, providing switch output to ground. So in led matrix, to select row, one of the channel is closed to ground and all other outputs are kept open. Concept gets clearer in Figure 52 of next Chapter.

Row Driver

Row driver consists of ten switches to ground taken from Analog Devices, ADG1414. Detailed function of ADG1414 has been described in [33]. Important points are described here, for more details one is referred to its datasheet. It is described as, 9.5 Ω $R_{ON,}$ ±15 V/±12 V/±5 V iCMOS, Serially-Controlled Octal SPST Switches. Following Figure 34 shows the block diagram taken from [33].

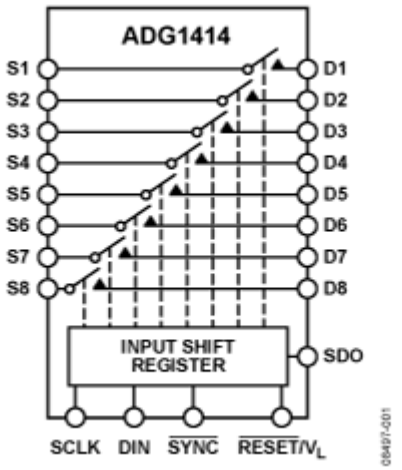


Figure 34: Functional Block Diagram of ADG1414.

Function of these switches in interface board is to provide ten outputs (using two of these controllers), with only one connected to ground at a time. Hence one row at a time is selected. One important point to be noted is that when dealing with SPI one could proceed with any of the topology i.e., daisy chain and independent slave. Here one could proceed with both, as subjected switch IC has enabled both functionalities. Figure 35 depicts schematic with independent slave configuration.

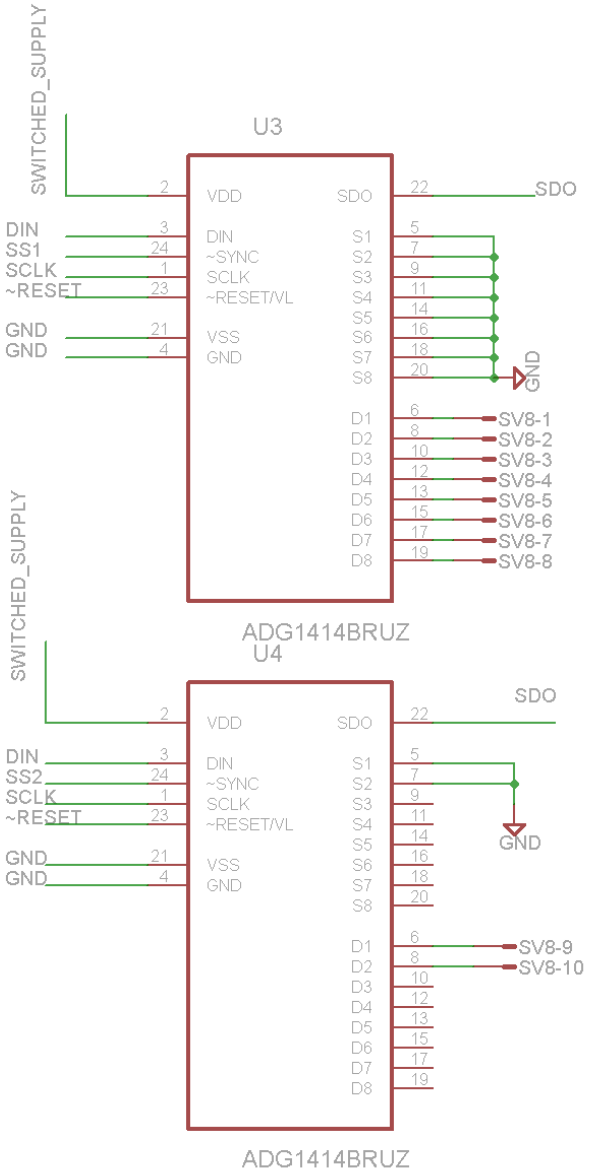


Figure 35: Schematic of Row Controller.

Column Driver

In column driver, outputs from DACs series have been taken to same numbers of amplifiers with some gain (in this case its 3), whereas the VCC of amplifiers can be +5 V or +15 V. So, each output of DAC (with range 0 to 5 V) can be a configuration source from 0 to 15 V at output header. Ten of such outputs are required. Figure 36 shows the conceptual schematic of column driver.

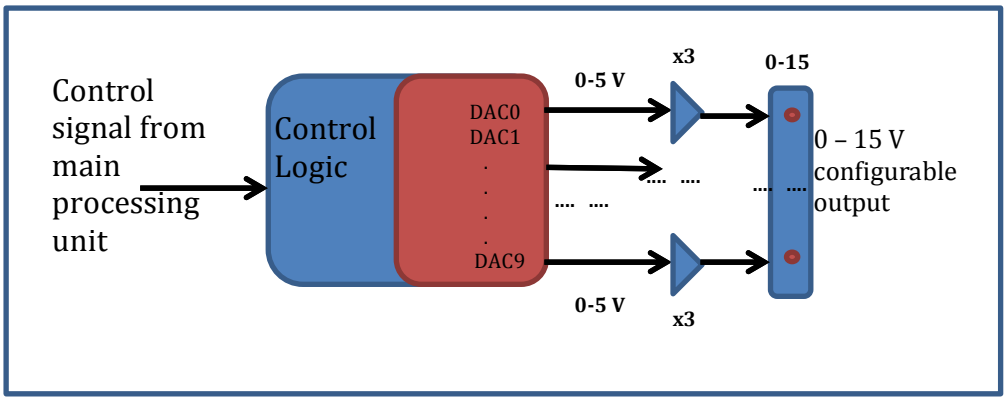


Figure 36. Logic Diagram of Column Voltage Driver.

LTC2635 is high accuracy quad 12/10/8 bit voltage-output I2C DACs IC as described in [34]. With its address pin one can connect it to VDD, GND or leave it unconnected to attain three different I2C addresses (in MSOP package, QFN package can have 27 different options). So, three of such integrated circuits can be there on one board leaving option for twelve I2C controlled voltage outputs. One can use internal reference of 2.5 V Full-Scale 10 ppm/^{0}C (LTC2635-L), 4.096 V Full-Scale 10 ppm/^{0}C (LTC2635-H), or an external reference (i.e. 5 V). LM224 quad amplifier with output 20 mA max [35] can be used for gain of 3. Ten of the outputs are used in above mentioned configuration; the two left outputs are used for any other differential or single ended outputs without gain (i.e. 0 to 5 V) suitable for electro chromic display (or any other application). Figure 37 shows Quad DAC IC with amplifier having gain set through resistors R4 to R9.

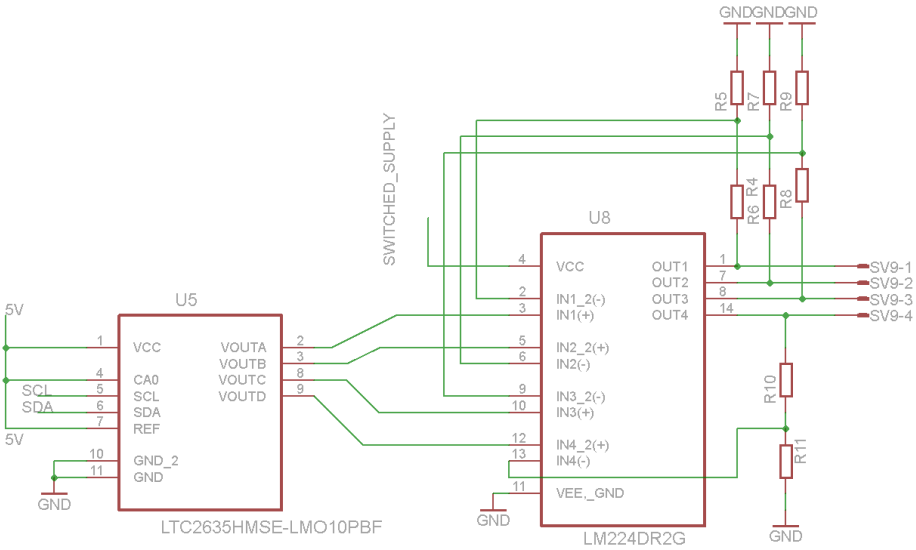


Figure 37. Schematic of Quad DAC IC with Amplification.

For Row driver there are many possibilities starting from MOS FETs to other switch ICs. But in short it can be comfortably stated that ADG1414 provides low on resistance, easy to program, easy circuit design, and small area required on board. For Column driver, again there is wide range of possibilities. One possible solution is to use IO expanders; there are many types of IO expanding chips available in the market for 8 bit, 16 bit, or even more resolution.

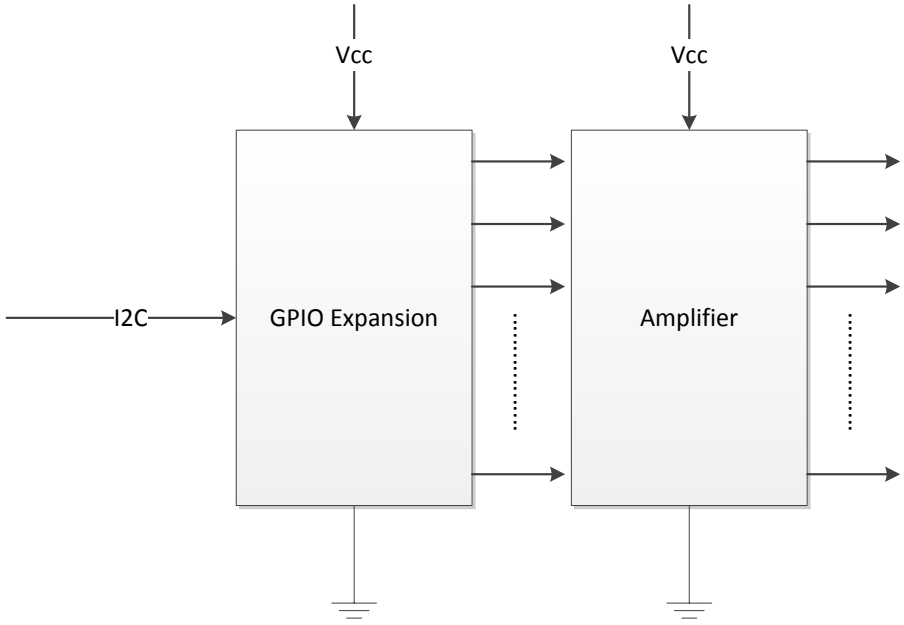


Figure 38. Alternate Column Driver.

In Figure 38, it is depicted that column driver can be made through GPIO Expansion controllers, but problem with this type of system would be that it doesn't provide configurable scaling of voltage levels for different type of printed displays, one solution can be manual switching of VCC for Amplifier to provide desired voltage level which cannot be considered a good solution. There are many choices for low cost IO expanders but as for variable voltage DACs were suitable for the task one can look for Single / Dual / Quad / Octal DACs. AD5308 (AD IC with SPI interface) & DAC108S085 (TI IC with SPI interface) are the example of eight DACs in single package. More DACs (especially 10 which might have suited this project) in single package are rare, AD5516 with 16 Channels and 74 lead CSPBGA high cost package is one such example of more than 8 DACs in one package. Analog Devices do have some dense DACs up to 16 bit resolution and 40 channels. DAC ICs AD8802/AD8804 have 12 channel 8 bit DACs, electrical specs are completely satisfactory and low cost is highly advantageous, but it wasn't selected for obvious reason of I2C communication bus preference, but can be tried in future extensions of the work.

Figure 39 shows the final layout of Interface Board. PCB specifications are already defined in description of Figure 25. Size of board was 64 x 62 mm^2.

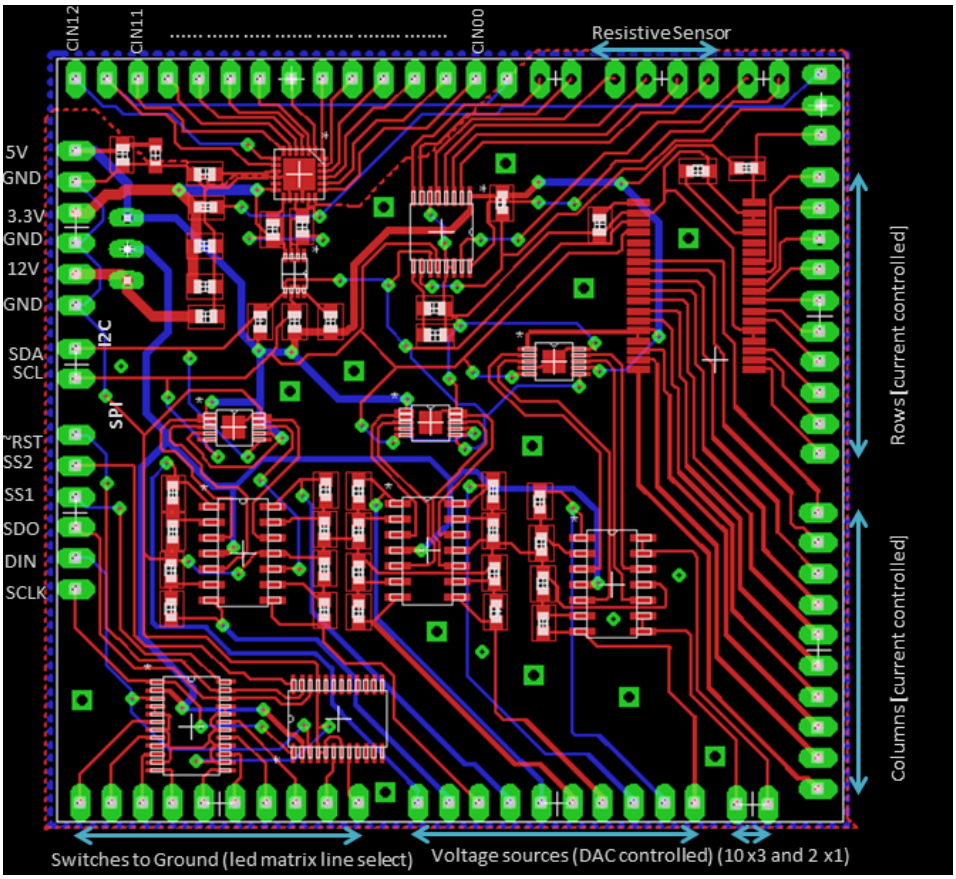


Figure 39: IB layout.

4.4. IV Curve Tracing Board

The purpose of this board was originally to trace IV curve of solar cell and based on this information, microcontroller or processor can decide to take input from it, as primary power source or at-least plot the curve through HTTP calls just for demonstration purpose.

Principle of work for this board has been the simplest of curve tracing instruments. It simply sweeps a voltage to gate of MOSFET to change load on the solar cell under illumination condition to vary current and voltage. For measurement either ADC input of BeagleBone or ADC IC can be used. A quad DAC IC which is described early was used. Quad DACs provide option to have four MOSFETs with different drain resistance to cover different parts of curve shown in Figure 40 and 41.

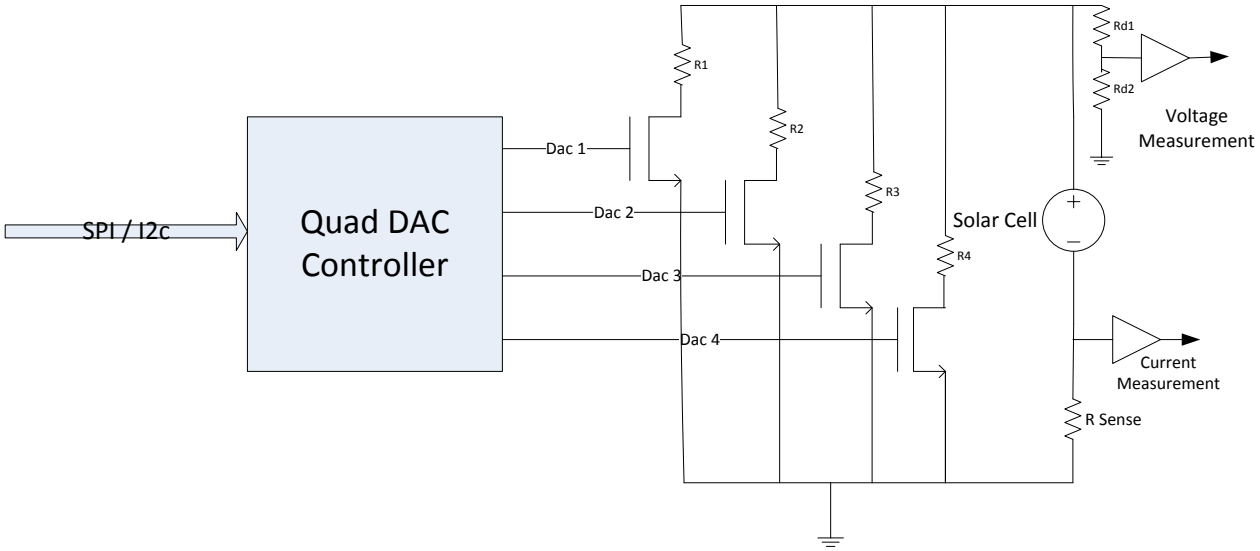


Figure 40: Logic Schematic of Variable Load.

To find voltage and current at particular time instance, one has to take care about maximum voltage which can be at ADC port of BeagleBone. For the sake of ease, small resistance with low tolerance; e.g., sense resistance (R_{sense}) ≥ 1Ω can be used to sense current. For regular solar cell, very small resistance i.e. 50mΩ may also works as larger current will output more voltage at non-ground terminal of resistance. But for printed solar cells, due to very low current it's not been working well. Voltage from solar input is divided via voltage divider circuit to make maximum of 1.8 V It is passed through one of amplifier in voltage follower configuration of dual precision LT1013 op-amp. With assumption that maximum current is known for particular solar cell, one knows about maximum voltage appearing onR_{sense}. It is amplified via second op-amp of dual precision amplifier LT1013 so that it can reach 1.8 V. Both outputs are taken to ADC inputs of BeagleBone and measured. Another possibility kept in this board was to measure voltage and current via MCP3426 dual channel ADC. Resistance R0 to R3 described above are set so that it cover maximum operating points on IV curve (though setting just one resistance R = 0 Ω serves the purpose).

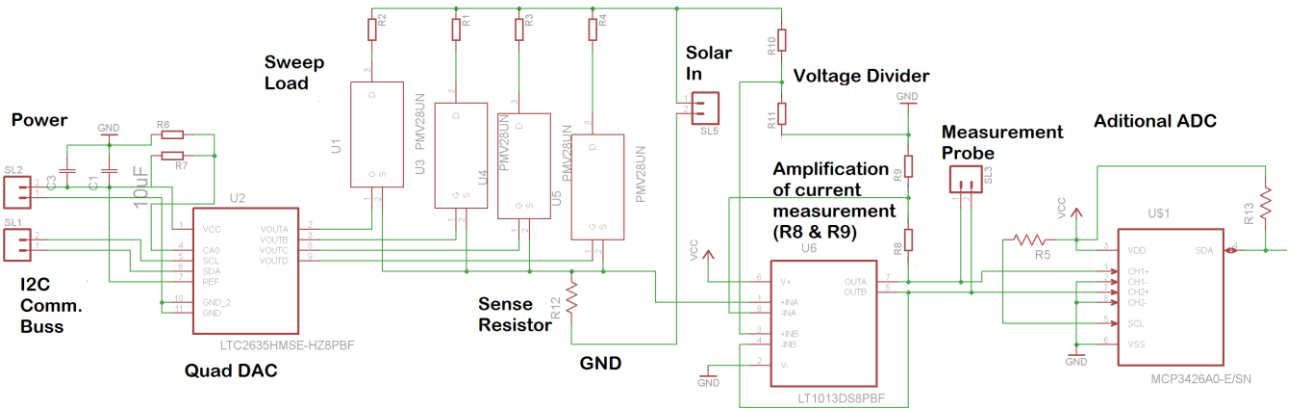


Figure 41: Schematic of IV Curve Tracing Board.

Figure 42 shows final layout of IV curve tracing board. PCB specifications are already defined in description of Figure 25. Size of board was 33 x 29 mm2.

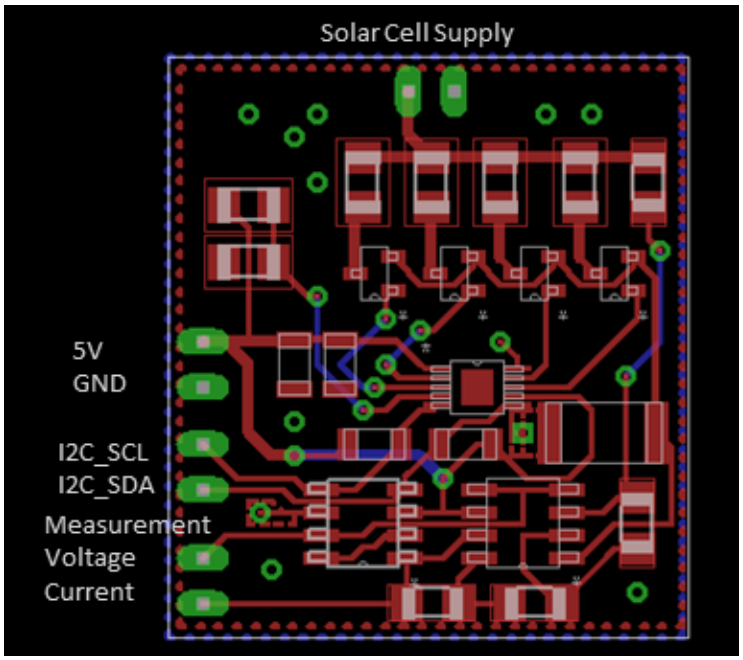


Figure 42: Layout of IV curve tracing board.

Alternate choices for DAC ICs have already been discussed in Voltage Based led driver section; here, it is good time to mention different techniques for making digital potentiometer for such measurement scheme.

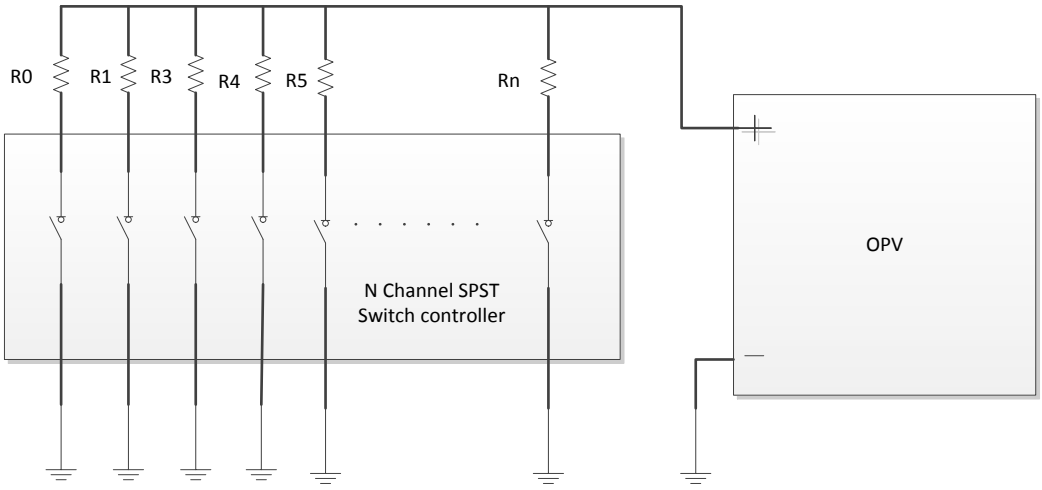


Figure 43. N channel SPST controller based digital potentiometer.

In Figure 43, digital potentio-meter capable of closing N SPST switches at a time, and selecting $\sum_{k=0}^{N}\binom{N}{k}$ total combinations, where N = total channels, k is no. of switch closed for making possible combinations and $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. So, four channel SPST switch controller will be able to make $\binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 1 + 4 + 6 + 4 + 1 = 16$ different parallel resistance combinations. Eight channel SPST switch controller is able to make 255 different combinations. One can chose resistors carefully so that desired graph by changing load is having enough information to plot.

4.5. ARDUINO MINI 05 LIGHT BOARD:

A microcontroller board was placed between IB and BB. Main purpose of this board was to serve as low power interface for future. In current structure, this board seems to be completely spare as BB has all the functionalities which this board has (no additional feature is provided in terms of processing or application). Necessary information about this board can be found in [36].
It has ATmega238 microcontroller, with operational voltage 5 V, and 14 digital IO pins of which 8 can be used as Analog Input Pins. It has 1 KB EEPROM and 2 KB SRAM. Clock speed is 16 MHz. It has been shown in Figure 44.

For programming, there are many environment solutions, but easiest is Arduino software which uses C-like language. It can be programmed with USB Serial adapter. Schematic of each and every revision is provided in official website so that one can easily understand its hardware for further changes.

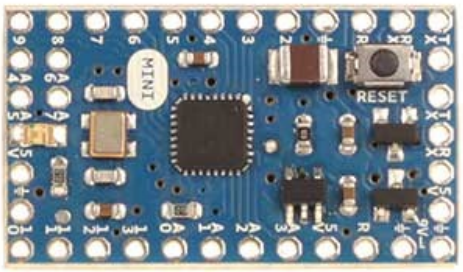


Figure 44. Arduino Mini.

Arduino Nano with all same specs, a little more in mechanical dimensions, with additional built in FTDI USB-to-TTL Serial chip used for programming the microcontroller, but comparatively higher price can serve the purpose as well.

5. SOFTWARE STRUCTURE FOR DEMONSTRATION DEVICE

In this chapter, software structure for BB and Arduino board, and their interaction with each other as well as with IB will be discussed. An example demonstration was created for the presentation in this project, but software structure was created so that it is not limited to the example demonstration rather code can be edited to change demonstration results extensively. Complete program code is given in Appendix 1 to 3.

5.1. SOFTWARE STRUCTURE FOR BB

Embedded Linux comes with the comfort for programmer to choose any language for making any kind of application. Real time (RT) processing and multitasking were required in this project which is conveniently offered by Linux distribution Angstrom provided with BB. In this chapter basic structure of program will be discussed, choices (alternatives of software structure) are mostly skipped, as purpose of this chapter was to convey the reader a detail of how this demonstration was constructed, rather than to provide detail about what other choices one actually has to do the same task. It is high time to mention that 'Python 2.6' [37] was used as programming language for BB, though there are other powerful choices available as well, such as Java, C/C++. Reason for choosing Python over other languages was it requires from programmer very small time for building very powerful applications like web server. It is also good time to mention that Arduino software is easily available, and is used to download program in Arduino board which uses C-like language syntax. Reader of this chapter is assumed to have introductory background with Python and C.

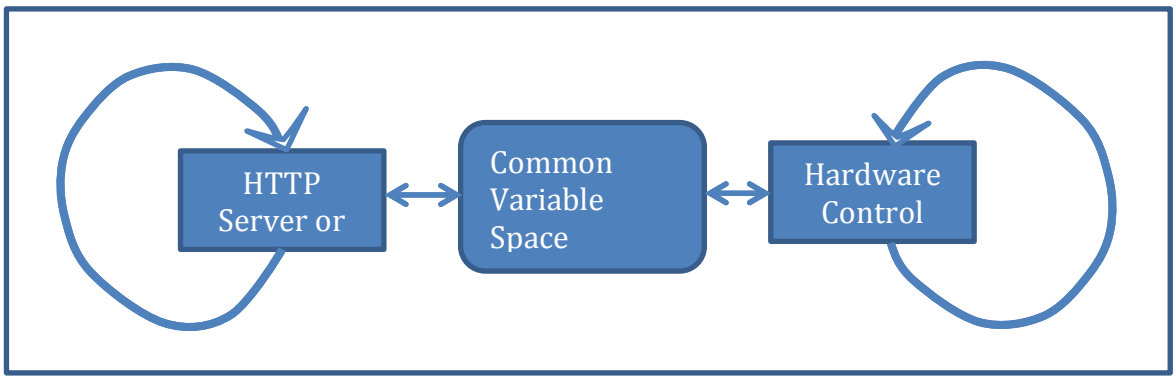


Figure 45. HTTP and Hardware Loop structure.

Figure 45 shows an overall view of software structure for BB with two simultaneous parts, (a) server part and (b) hardware control part which exchange messages with Arduino board etc. One should recall the purpose of project: a general demonstrating device for printed electronics. Wireless router can be a good example of user interface for that machine. Once router is in one's network domain, he can type its IP address in web browser like Mozilla Firefox or Google Chrome and configure it, or take any RT measurement from it; therefore, part of Linux machine's program should interact to internet/web calls. Simply, one can open an interface on his web browser to see some measurement results and to do some configurations as well. Part of that program is actually controlling printed components and taking measurement through IB. So, in this way multiple processes at a single time can create a system like the example given above.

Possible solution for above mentioned problem can be several. One can initialize more than one process with shared file or other resources etc. More often used technique is 'Threading'. A single process can have multiple threads. Those threads have access to all the resources (globally defined

variables etc.) of the main process. So this technique poses lesser burden on processor as different processes have access to different resources.
In following python example taken from main program of demonstration, there is a thread defined as hardware thread, a scheduler will schedule main thread (HTTP server) and hardware thread. Thread for hardware have access to some global useful variable which can either be updated by feedback through user interface or measurement data arriving from Arduino board. Then it has a function defined which is recurrent in nature, and interacts with Arduino board to update variables like CGI_ARG (argument variable for Common Gateway Interface i.e., CGI call [38], will be clear after reading following text).

```
class ThreadClass_Hardware(threading.Thread):       # In this class, hardware thread is defined,
                                                    # Loop function contain all of the hardware control
                                                    #sequence
def run(self):                  #function which describes function of thread
# Global variables describes the shared variables between main thread and hardware thread
        global CGI_ARG    #Input from capacitive sensor changes this argument
        global cycle_length    #input from http form
        global on_time    #input from http form


        run(setup, loop) #setup function runs once, loop function run as loop when this thread starts
```

Following part of the code is the main thread, here it starts hardware thread (which separates itself from the main) and then starts the Tornado HTTP non-blocking server. This server is faster than regular SimpleHTTPServer class (provided in Python standard libraries). Also it can handle very large number of clients, though it was not required in this project. To define complete HTTP server application, one needs to understand the html code which is sent as reply to main query '/' from http client or in other words one needs to write ip 192.168.7.2:8000/ in http browser (if there is nothing written after '/' it is called main query and server has to send main html code to client). Here 8000 is the port no. described in python script, 192.168.7.2 is the IP address is BeagleBone (though it can be changed). One needs to understand what other handlers are required depending on html code sent by main handler. Here, a form handler to catch the fill-ups of forms (on_time, cycle_length etc.), CGI Handler to handle CGI call to update button position on http browser and a static file handler from address /html/form was required. It is noteworthy both of the threads are in continuous loop unless they receive a keyboard interrupt.

```
def main():
    global CGI_ARG                #global variables shared by all functions
    global cycle_length           #global variables shared by all functions
    global on_time                #global variables shared by all functions
    CGI_ARG=0               #initialization
    on_time = 60            #initialization
    cycle_length = 3        #initialization
    print 'main program running at pid = ',os.getpid() #PID display (not necessary block)
    t=ThreadClass_Hardware()        #hardware thread
    t.start()                       #starts here!

    tornado.options.parse_command_line()
#HTTP server code starts here
    application = tornado.web.Application([
#application definition
        (r"/", MainHandler),
#constituents of application: main handler
        (r"/form", FormHandler),
#constituents of application: Form handler
        (r"/cgi-bin/ajax_example.cgi", CGIHandler),
#constituents of application: CGI handler
        (r"/html/form/(.*)",tornado.web.StaticFileHandler, {"path": "./html/form"},),
#constituents of application: static file handler

    ])
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(options.port)
```

```
    tornado.ioloop.IOLoop.instance().start()
```

In previous discussion, CGI handler example was described. Below mentioned example code handle CGI in such a way that it runs a shell script and send its output to HTTP client which is waiting for an html code. That html code replaces previous code section (mentioned in form.html which is sent via main handler) and displays it on web browser.

```
class CGIHandler(tornado.web.RequestHandler):

    def get(self):
        global Flag                 #Global variable
        global Command_to_Thread    #Global variable
        global CGI_ARG              #Global variable
        command=''                  #initialization
        command = "python ./cgi-bin/ajax_example.py " #shell command (ajax_example is python file to be run,
its output is grabbed and sent to HTTP client)
        a = random.randrange(3)  #for test purpose #randomly change the switch status #not used in demo
        command += str(CGI_ARG)  #string concatenate
        output = subprocess.check_output(command, shell=True)
        self.write(output)
```

The ajax_example.py file which is mentioned for CGI script is responsible for changing any display information at HTTP Browser, at regular interval. It contains simple check on input argument and prints a series of statements (forming) html code accordingly. It is noteworthy, that this is not a standard way to handle CGI call but it will work well for the subjected demonstration. An alternate way is to build a standard CGI server using BaseHTTPServer and CGIHTTPServer libraries. Moreover Tornado WSGIApplication class could be used to handle a standard CGI script.

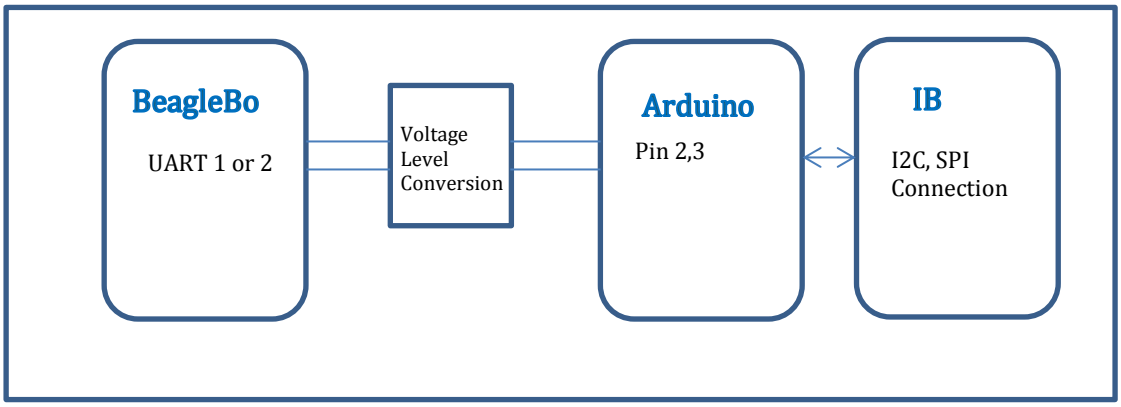


Figure 46. Serial Communication block diagram between Arduino Board and BB.

Hardware loop (function called in Hardware Thread) is comprised of serial inputs and outputs to microcontroller from BB as hardware is depicted in Figure 46. In above mentioned example Serial2 which is BB pins P9_21 and P9_22 found in Table 11 of [19], is used to communicate a user defined Serial (pin 2, 3 of Arduino pro mini) and to alter CGI_ARG (global function) which is input argument of python CGI file, and to take arguments from Arduino which are actually readings taken through Interface Board.

5.2. PROGRAM STRUCTURE FOR ARDUINO

Program Structure of Arduino boards is such that it has a setup () function and loop () function. Setup function is responsible for initialization of variables, pin modes and starts using libraries. [39] Arduino loop function does what its name suggest i.e. it loops consecutively. It is used to actively control Arduino board [40]. In simple words Setup function runs once and Loop function is concurrent.
In example given before, there is a hardware thread communicating with Arduino board continuously and taking readings and process it and send data to Arduino board for further process, it consists of instruction to perform a demo also required parameters for that.
In following communication of Arduino with Interface board ICs through SPI/I2C is discussed.

5.2.1. CAPACITIVE SENSOR CONTROLLER AD7417

[27] describes I2C format for controller. In previous chapter it was described in hardware different sensors are connected to differential CDC in different stages. Now it is time to show the sequence of commands sent to controller via I2C. Figure 47 shows I2C communication messages sequence for AD7417.

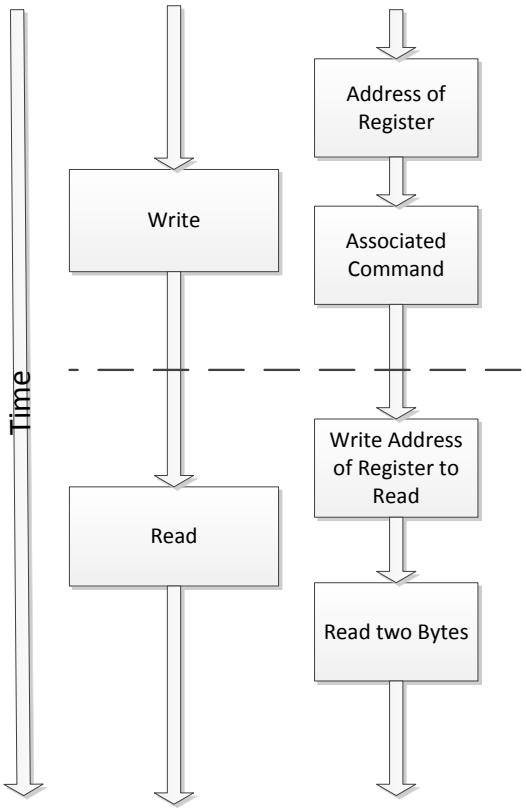


Figure 47: How to read measurement for capacitance via AD7147.

There are register addresses provided by document to configure each stage, and take measurement data back to host process. For example, Registers at address 0x0080 and 0x0081 are configured for Stage 0. And to read result of stage 0 one needs to point at register 0x000B (also given in Table 31 of referred document) before requesting data as shown in Figure 48 taken from [27].

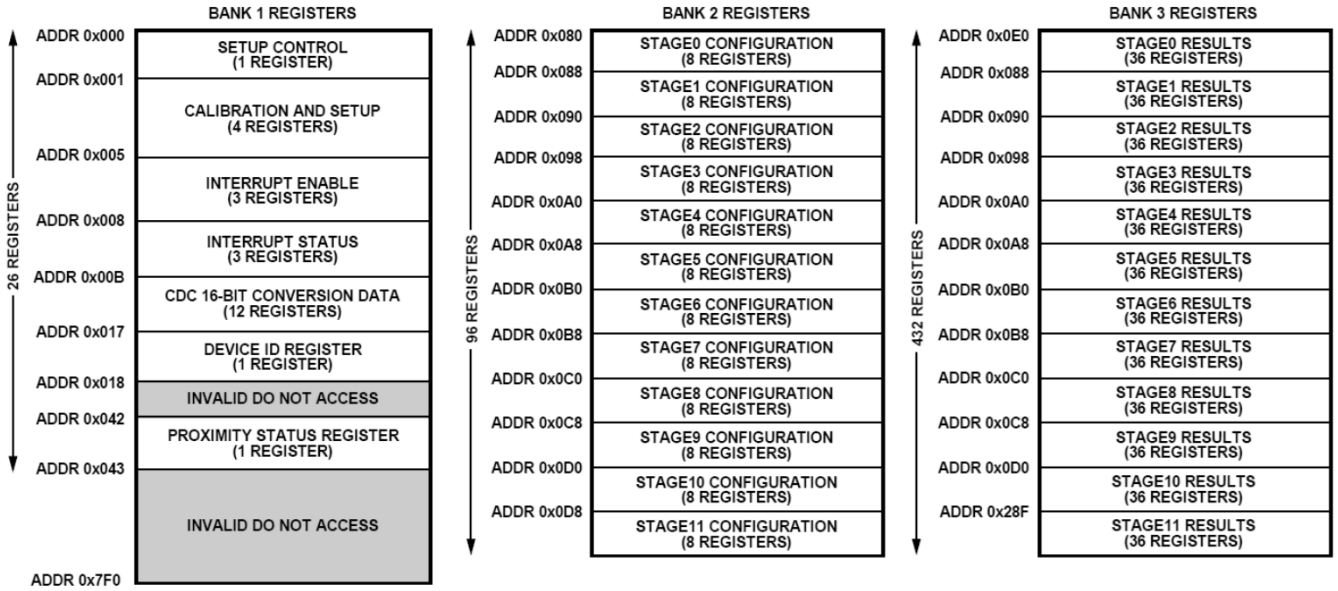


Figure 48. Register Banks of Ad7147.

Detail for configuration word for 0x0080 and 0x0081 can be found in Table 34 and 35 of referred document. Registers are named as STAGEx_CONNECTION[6:0] & STAGEx_CONNECTION[12:7].

In following (Arduino C program) Example, Sensor 1 is connected to negative port and Sensor 0 is connected to positive port of Differential CDC. Inputs In1 to In4 are referred to as configuration register data for Stage Zero.

```
unsigned int cap_op;
cap_op = CAP_AD7147_I(0x00,0x06,0xf0,0x00); //  Connect Sensor 1 to negative and Sensor 0 to positive input
of Cap to digital converter differential inputs

unsigned int CAP_AD7147_I(byte in1, byte in2, byte in3, byte in4)
{

 unsigned int var_AD7147;
 Wire.beginTransmission(0x2C); // AD7147 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0x00);            //
  Wire.write(0x80);            //
  Wire.write(in1);
  Wire.write(in2);

  Wire.endTransmission();      // stop transmitting
  delay(10);
  ////////////////////////////////////////////////////////////////////////////////
  Wire.beginTransmission(0x2C); // AD7147 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0x00);            //
  Wire.write(0x81);            //
  Wire.write(in3);            //
  Wire.write(in4);            //
  Wire.endTransmission();      // stop transmitting
  delay(10);
  ////////////////////////////////////////////////////////////////////////////////
  Wire.beginTransmission(0x2C); // AD7147 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0x00);            //
  Wire.write(0x0B);            //
  Wire.endTransmission();      // stop transmitting
```

```
  delay(10);


  ////////////////////////////////////////////////////////////////////////////
  Wire.requestFrom(0x2C, 2);    // request 2 bytes from slave device 0x2C
  byte b_AD7147;

    b_AD7147 = Wire.read(); // receive a byte
  var_AD7147 =  b_AD7147*256;
  //Serial.print(b_AD7147); Serial.print (" , ");
    b_AD7147 = Wire.read(); // receive a byte
  var_AD7147 = var_AD7147 + b_AD7147;
  //Serial.print(b_AD7147); Serial.print (" -> ");
  //Serial.println(var_AD7147);
  delay(10);
return var_AD7147;

}
```

5.2.2. TOUCH SCREEN CONTROLLER TSC2003

As it is described in [29], TSC2003 is an I2C touch screen controller so basic structure of program code will remain same as it was for AD7147. Command word shown in Figure 49, can be found in referred document.

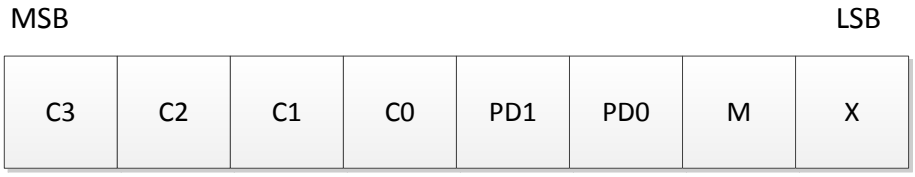


Figure 49. Command Word for TSC2003.

Where C3..C0 are command bits, PD1 & PD0 is power down bits and must be zero for power saving, M is mode bit i.e. 0 for 12 bit mode and 1 for 8 bit, X is don't care (figure 50). Command detail is found in TABLE I. Possible Input Configurations. C = 1101 for measurement of x-Position. Figure 50 shows touch screen printed with four buttons (represents original touch screen). X measurements were taken from min to max, and same procedure was done with Y position. So mid points of both were known to differentiate four different buttons. Therefore Z_1 & Z_2 Positions were not required.

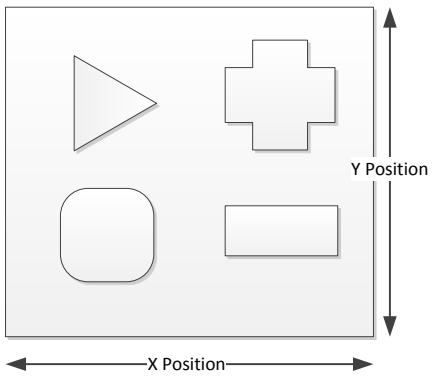


Figure 50. Touch Buttons printed screen.

In following Figure 51, flow of code is defined, for more details Figure 12 and 13 of [29] must be consulted.

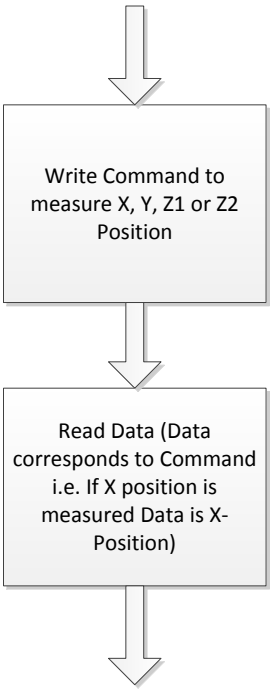


Figure 51: Commands flow to measure X, Y positions

Following Arduino C code example is for measuring X position of Touch screen via TSC2003.

```
Wire.beginTransmission(0x48); // TSC2003 address
                              // device address is specified in datasheet
                              // sends instruction byte  (command)
  Wire.write(0xC0);             // C = 1101 (command, measure X position) , 0 for PD1, PD0, M, X //Figure 11
of ref manual
  Wire.endTransmission();     // stop transmitting
  delay(10);


  Wire.requestFrom(0x48, 2);    // request 2 bytes from slave device 0x48



    b = Wire.read(); // receive a byte
  Xpos =  b*16;
    b = Wire.read(); // receive a byte
  Xpos = Xpos + b/16;
```

5.2.3. DACs/SPI Controlled Switches for LED Matrix Driver

As it was shown in previous chapter of System Structure, this is a simple Row and Column driver made up of DACs and Switches. In actual, it is designed for a 10 x 10 matrix, but in below mentioned example the concept is illustrated once more, with a smaller matrix. In following example, illustrated through Figure 52, Bit information is retrieved from image, suppose image is just two color, and third pixel is having color (only white or color can be the option), so one can represent it with ‘one’ and rest three pixels of image with ‘zero’. One is multiplied by max voltage level, and if first Quad DAC is controlling four segments one of those four has to output max voltage, while others should remain to voltage zero. And for selecting first row, only that row is connected to ground. Rows are switched quickly with fetched column data and display is shown. Two 8-SPST switch ICs are used here for ground connection. Data transmission is so simple for

SPST controller that a Byte represents switch on and off e.g. 0b00000000 mean all switches are off and 0b0000001 mean one switch is on.

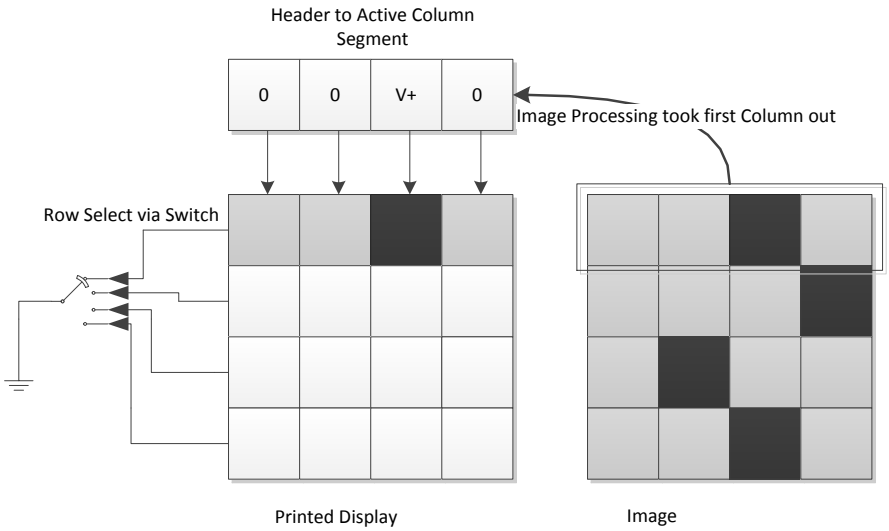


Figure 52: Concept of DAC/Switch based Driver.

Figure 53 illustrates one need DAC addresses to send bit information multiplied with voltage to make a column header (with MSOP package IC one can have three addresses of LTC2635 as described in previous chapter).

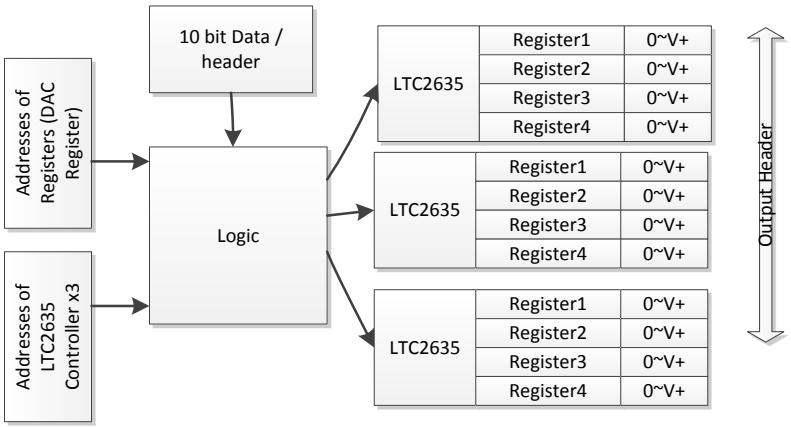


Figure 53: Block level diagram of I/Os of Code structure for control of DACs.

So logic is defined to take a 10 bit number, shift its bits at every cycle, take corresponding IC & DAC address from Address Array, multiplying that zero or one (bit) with voltage level and send I2C message. Table 2, Slave Address Map (MSOP Package) in document [34] describes possibilities of addresses of IC, Table 3 represents Command possibilities, and Table 4 represents Register address of DAC1 to DAC4. Suitable Command (4 bits) + Address (4 bits) is formed and sent to suitable IC address.

PSEUDO Code example to change voltage output of a DAC

```
I2C Start Transmission  # Begin Transmission
I2C write -> IC_Address # Address of IC (0x10, 0x11 or 0x12)
I2C write -> reg        # Reg Address, i.e. which DAC of IC to be configured 0xff for all four dacs of IC
                        # 0x30 to 0x33 for individual DACs
I2C write ->val         #  Value, This is enough for DAC in 8 bit mode, 0x00 to 0xff represent 0 to 5V
I2C write -> 0x00       # Value, for more than 8 bit mode, this must not be 0
                        # But it is must to send as IC is expecting this byte
I2C Stop Transmission   #  stop transmitting
```

5.2.4. CURRENT DRIVER USING MAX6953 CONTROLLER

Image processing is somewhat similar to previous controller only difference is processing is done vertically to fit 5 x 7 digits. As previously discussed, details are provided in [31], Table 15 of document gives detail of RAM Character MAP here one can find RAM00 to RAM23 is free for user to design. For four digit display, 4 RAMs needed to be designed. In Figure 54 it is illustrated how vertical processing is done to make RAM data from Image. Example code is given in Appendix 6.

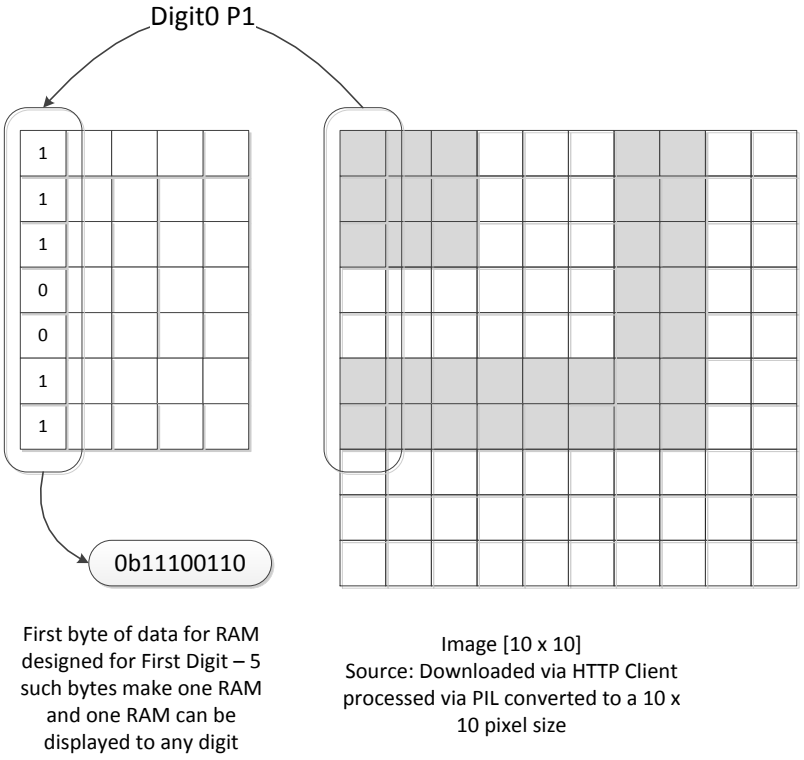


Figure 54. Illustration of Image Processing for Current Controller.

Below mentioned python example converts images first column's seven bits to a Data Byte.

```
im = Image.open("./uploads/image.bmp") # open image
im.thumbnail((10,10),Image.ANTIALIAS)
data11= 0
thresh = (252, 252, 252) # can be changed to see changed effect (0,0,0) mean pure black and (255, 255, 255)
means pure white
pix = im.load() #load variable pix with image data


for i in range(7):
    if pix[0,i]<thresh:
        data11 += 1<<i #bit shifted to ith position

print data11
```

As it is defined earlier, in proposed architecture IB is connected with Arduino Board, So above processed data is sent to Arduino by BB to download into controller, shown in Figure 55.

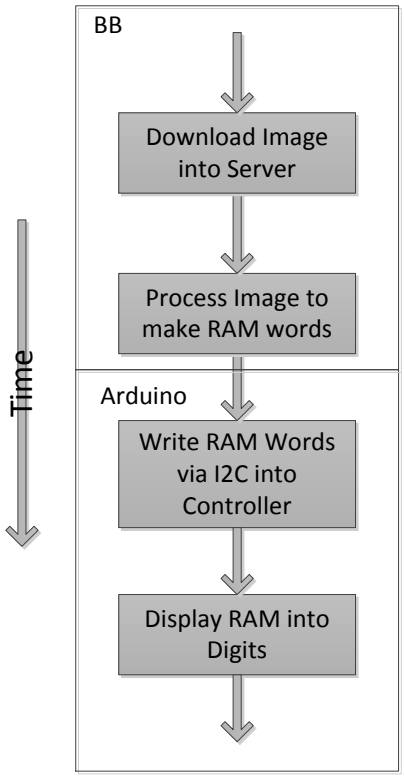


Figure 55. Time Flow Diagram of steps for image formation in MAX6953.

Following Pseudo Code describes how to write RAMs and display RAMs on Digits.
PSEUDO CODE for writing RAM00 DATA

Address of RAM data can be found in Table 18: User-Definable Font Pointer Base Address Table in [31] (Register 0x05 shoulch have data 0x80 to receive data for RAM00). For more examples reader must be referred to Table 19: User-Definable Character Storage Example.

```
I2C start Transmission
I2C write byte -> 0x50 # address of IC
I2C write byte -> 0x05 # Writing DATA for RAM
I2C write byte -> 0x80 #RAM00
I2C write byte -> 0xXX # First Byte
I2C write byte -> 0xXX # Second Byte
I2C write byte -> 0xXX # Third Byte
I2C write byte -> 0xXX # Fourth Byte
I2C write byte -> 0xXX # Fifth Byte
I2C endTransmission
```

PSEUDO CODE for displaying RAMs on DIGITS

```
I2C start Transmission
I2C write byte -> 0x50 # address of IC
I2C write byte -> 0x60 # Pointing Address of first Digit, next is 0x61 Table 5. Register Address Map in [31]
I2C write byte -> 0x00 #RAM 1 taken from Table 15 of [31] – Digit 0
I2C write byte -> 0x01 #RAM 2 – Digit 1
I2C write byte -> 0x10 #RAM 16 –Digit 2
I2C write byte -> 0x11 #RAM 17 – Digit 3
I2C endTransmission
```

In context of previous pseudo code, following is the example in Figure 56 of Two RAMs data downloaded at outpu of two digits.

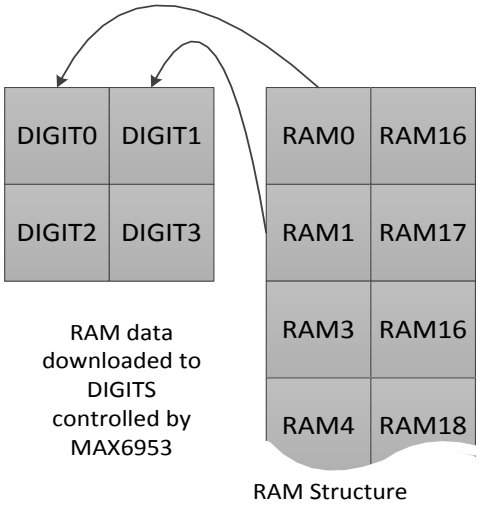


Figure 56. RAM0 and RAM1 displayed at Digit0 and Digit1.

5.3. IV CURVE TRACER

This part was designed separate on BB, though it can be a part of main Server. This part involves a HTTP Server, which receive main call '/' from client. It sweaps voltage on each DAC connected to gate of MOS transistor, separately; takes Current and Voltage measurement, and plot result. Only one DAC with Drain Resistance 0Ω was enough for measurement. But Four MOS transistors are controlled by four DACs of IC LT2635. Here it was assumed that IV mins and maxs of solar cell are aready known for particular type. So voltage regions were divided, to have more points on curve, drain resistances were approximated for minimum voltage (representation on graph) taken from solar cell. This way one can improve IV curve details. Following Figures 57-59 show algorithm and voltage regions created in result. Note: Program structure to change register values is same as DACs used in DAC based LED matrix controller.

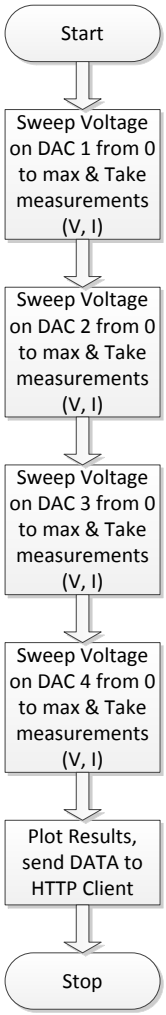


Figure 57. Flowchart of IV Curve Tracing program.

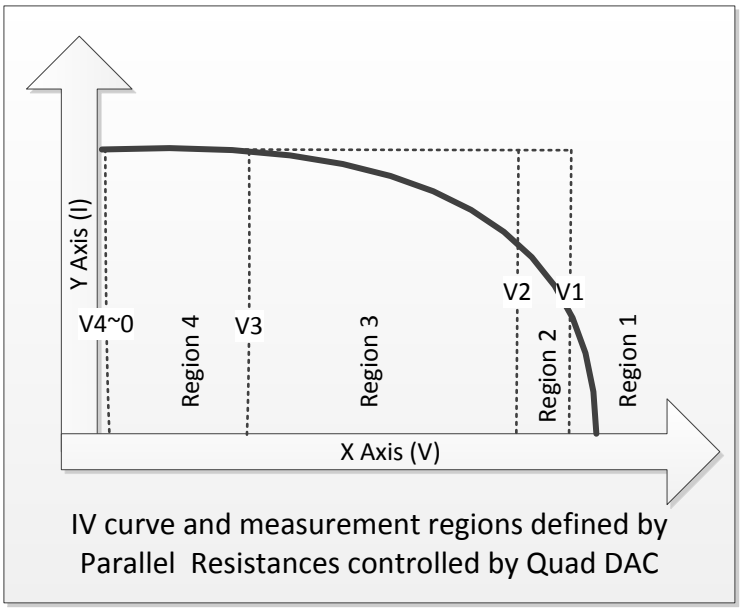


Figure 58. IV Curve and Voltage Regions.

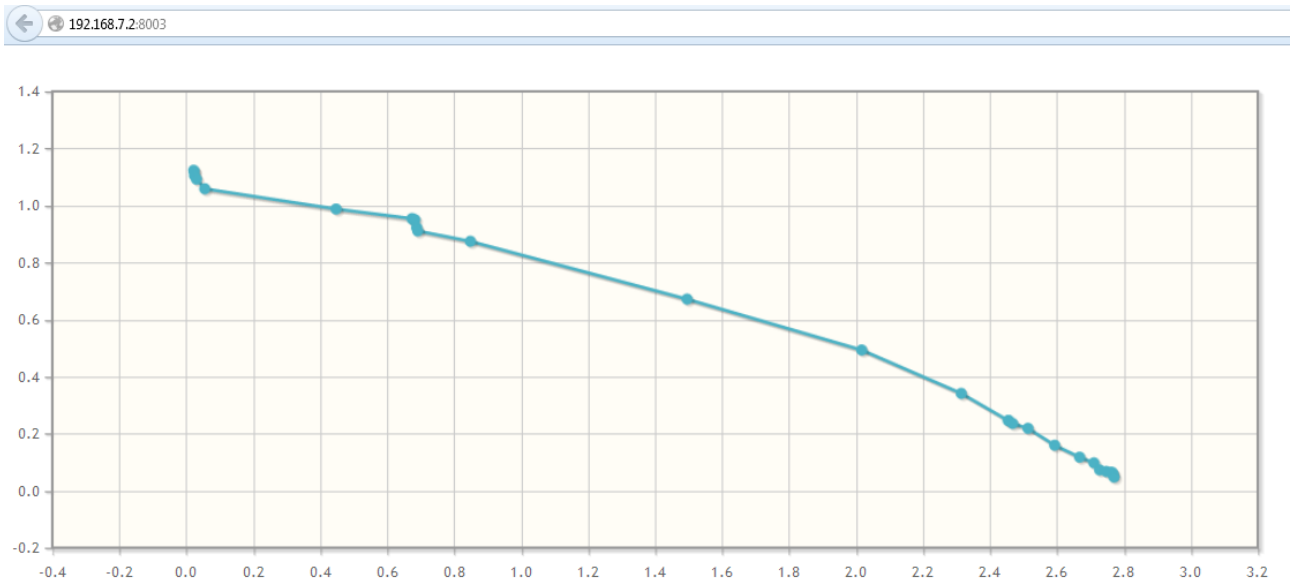


Figure 59. Snapshot of IV graph of a Printed Cell plotted by Server.

In conclusion it can be stated that Software Architecture of the system is such that, multitasking of embedded Linux has been utilized to achieve a relatively complex task, where different parts of systems have to communicate with each other.

6. EXAMPLE DEMONSTRATION

In this chapter, Final example demonstration will be discussed in detail, portion of programming code which can lead to change in demonstrations will also be discussed.

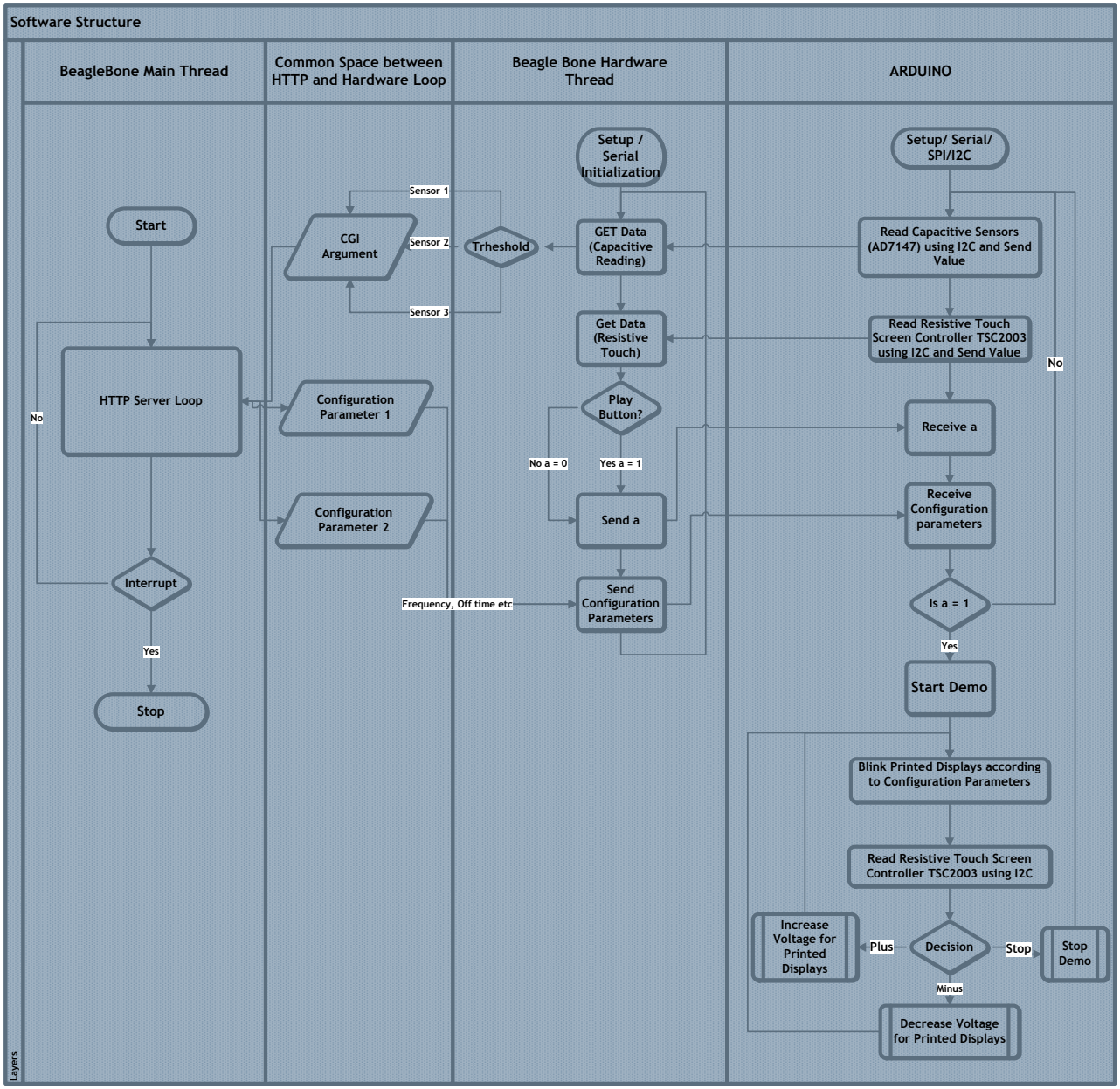


Figure 60. Flowchart for Example Demonstration.

In Figure 60, final example demonstration has been illustrated. This demonstration doesn't utilize full capabilities of IB and software structure, but it was planned for available printed led strings, electrochromic displays, capacitive sensor (3 printed lines), and finally a flexible touch screen with four printed options of play, stop, plus and minus. There are three main Processes which have to communicate with each other. One process is running on microcontroller board which controls the IB by I2C and SPI interfaces, takes data (to control LEDs) from hardware thread of BB process and gives measurement (capacitive & resistive sensor) results back. One process (hardware loop on BB) is actually a thread which is communicating with microcontroller. Finally for user interface there is one process thread which is taking care for messages with HTTP client. Main thread which is responsible to deal with HTTP messages is taking some parameters from user forms present on

HTTP client browser and also display measurement there at User Interface (UI) which it got via hardware thread. In this whole structure IV curve tracing has been exempted, but it can be added up with main thread, with no need to write another server again. One HTTP server is enough to tackle both i.e., UI and IV Graph Plots.

In example demonstration user can download cycle length and LED on-time from UI which reflect the blinking of printed led string, blinking effects can be started by play button of printed touch screen, it can be stopped , intensity can be modified through plus and minus buttons. Button statuses on browser are changed by proximity sensing of three printed capacitive sensing lines.

6.1. USER INTERFACE

For clearer picture one has consider the following web browser snap shot for demonstration user interface in Figure 61.

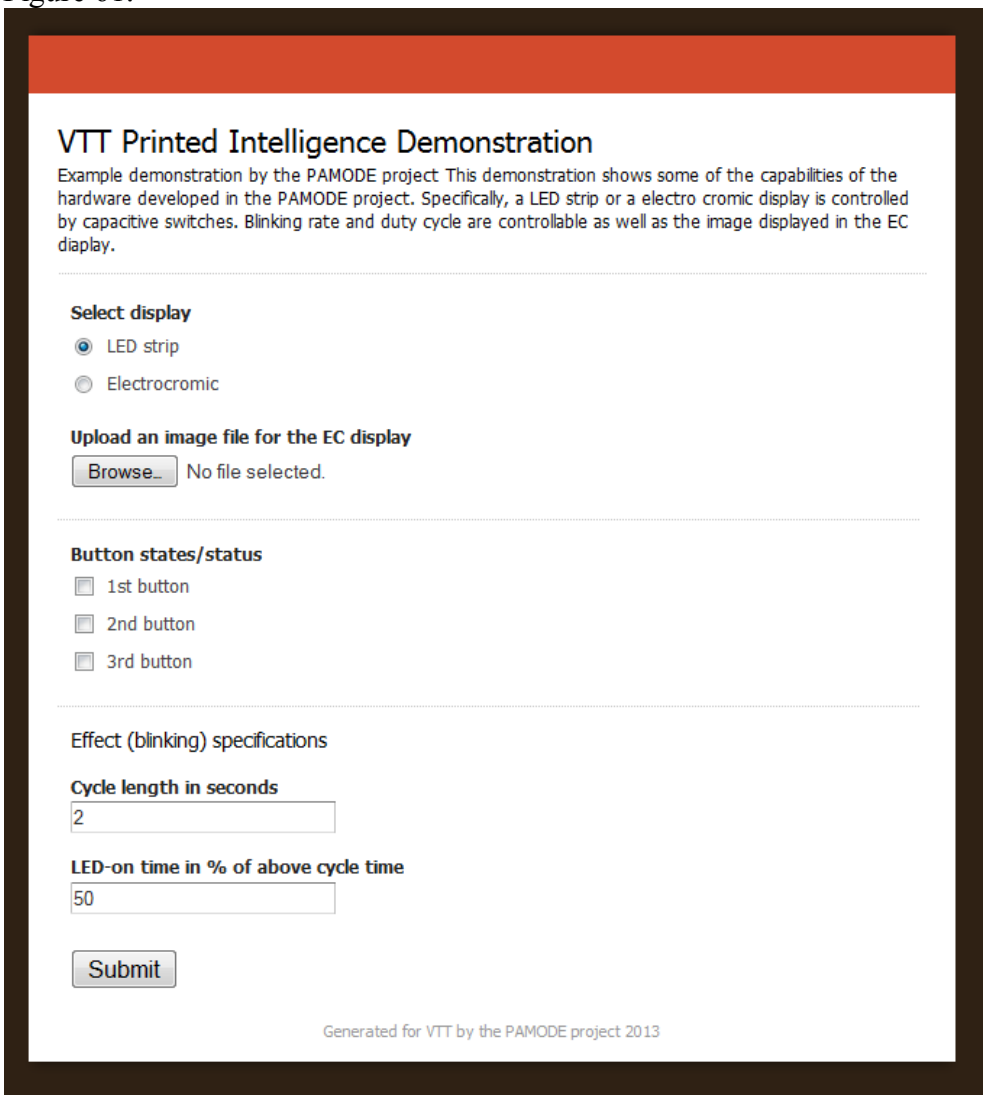


Figure 61. Example Demonstration Web Browser Snapshot.

As it was mentioned before, the demonstration involves configuration downloading and monitoring through web browser. Above Figure shows snapshot of a design demonstration web page. In this demonstration user can download either of option: LED strip or Electrocromic display; for example, to choose where to download processed image (voltage or current based driver outputs). Here it was

assumed that two printed displays are interfaced or in context of previous chapter discussion, it could be displayed via MAX6953 current controller for led matrix or DAC based voltage controller for electrochromic display matrix. It must also be noted, that led display matrix can also be derived by voltage based driver. Then there comes a 'Browse' button to upload an image file for server. Downloaded file is processed, and messages for MAX6953 and DAC based header are generated and sent to microcontroller board. Then there comes a portion of buttons. One of those buttons is ticked at a time. Code behind ticking of button is sent by Web Server in response to a CGI call, so this portion is kept updating repeatedly. Working of CGI call is discussed later, but here it is worth mentioning that in current example demo these button represents three capacitive sensors connected to IB, tick position tells which one of sensor is touched. Then there is section of blinking specification or parameters to be sent to final destination Arduino board so that it can control blinking according to it. Submit button download configuration / image data to server.

Here, detail of all of the interfaces is nearly over. For until now, user has visibility of three main things: a web browser or HTTP Client, demonstration device as a black box connected to user's computer & printed components attached to that black box. For ease one can simply define the whole system in a way that web browser can download led matrices display parameters to device or can finally reads sensor matrix positions coming from it.

6.1.1. HTTP MESSAGES BETWEEN BB MAIN THREAD PROCESS AND HTTP CLIENT

HTTP is a request-response protocol in client-server computing model [41]. At first user write down address of Server and port on web address, this sends a query GET '/' to HTTP server which must be handled in server code. Handler for this query is known as main handler at server side. It is also described in previous chapter; main handler sends HTML code to client. Here client reads that code and display some output on page. There can be static web servers or dynamic web servers. For static web servers static html code is written, HTTP request GET '/' to get the code and display it. It sends this request whenever it is refreshed. For dynamic web servers however one need to write html code which refreshes webserver automatically and outputs new values/text at particular time. HTML code can't perform the tasks, for this Client side code is written using JAVA script or AJAX for example. Following messages appear on terminal when main thread is run.

```
[I 130907 13:04:58 web:1393] 302 GET / (192.168.7.1) 57.80ms
[I 130907 13:04:58 web:1393] 200 GET /html/form/form.html (192.168.7.1) 393.22ms
[I 130907 13:04:58 web:1393] 200 GET /html/form/view.css (192.168.7.1) 24.87ms
[I 130907 13:04:58 web:1393] 200 GET /html/form/view.js (192.168.7.1) 20.14ms
[I 130907 13:04:58 web:1393] 200 GET /html/form/bottom.png (192.168.7.1) 11.63ms
[I 130907 13:04:58 web:1393] 200 GET /html/form/top.png (192.168.7.1) 23.32ms
[W 130907 13:04:58 web:1393] 404 GET /images/shadow.gif (192.168.7.1) 4.09ms
[W 130907 13:04:58 web:1393] 404 GET /favicon.ico (192.168.7.1) 3.88ms
server at port: 8000
cgi handler
0
[I 130907 13:05:00 web:1393] 200 GET /cgi-bin/ajax_example.py (192.168.7.1) 866.24ms
```

So, GET / message is for main handler, it sends html code. HTML code is just like series of print commands and some pictures/files addresses to display or for reference. Client asks view.css, view.js e.g., as it was present in html code. Client request GET /cgi-bin/ajax_example.py to update button states as shown in above Figure 61. Now at serverside there must be a code to handle this GET request. It actually executes ajax_example.py and grabs its output (which might come from series of print statements). Output is sent to web client again.

6.2. REUSABILITY

Suppose one wish to use this example for further enhancement, it is very easy to do that. But one must learn about html and python server libraries beforehand. In this section of the book, a look has been taken on what other possibilities are for configuration with existing programming framework.

6.2.1. CHANGES IN HTML CODE

Basic effect of html code on demonstration is how web browser at user end looks like. For any user facility provided at browser html code has to embed some functions in it, provided that server code is written so that it supports that function too. For example, a simple http server code on BB will be sufficient to handle a static web browser with some static information displayed, whereas if http client code is updating regularly with cgi calls, server code written on BB must handle it. Similarly, with addition of every form or button etc. in html code it is made clear whether BB server code support it or not.

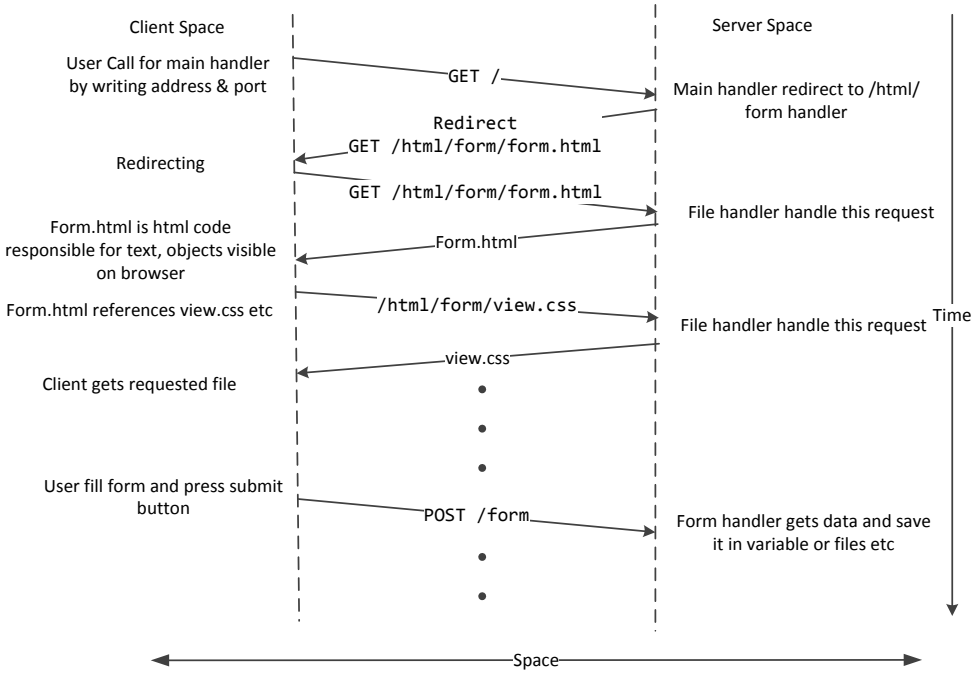


Figure 62. Approximate messages flow between server and client.

Figure 62 makes the point clearer, for each facility available at User end, must be handled at server side. So, any new type of handling can be done by updating html code and server code written in main thread for BB.

6.2.2. PATTERNS ON IMAGE FOR PRINTED DISPLAYS

Two types of hardware interfaces (output headers) are available on Interface board. Animation can be done for both cases. It is already mentioned, how to process Image for MAX IC and DAC based words.

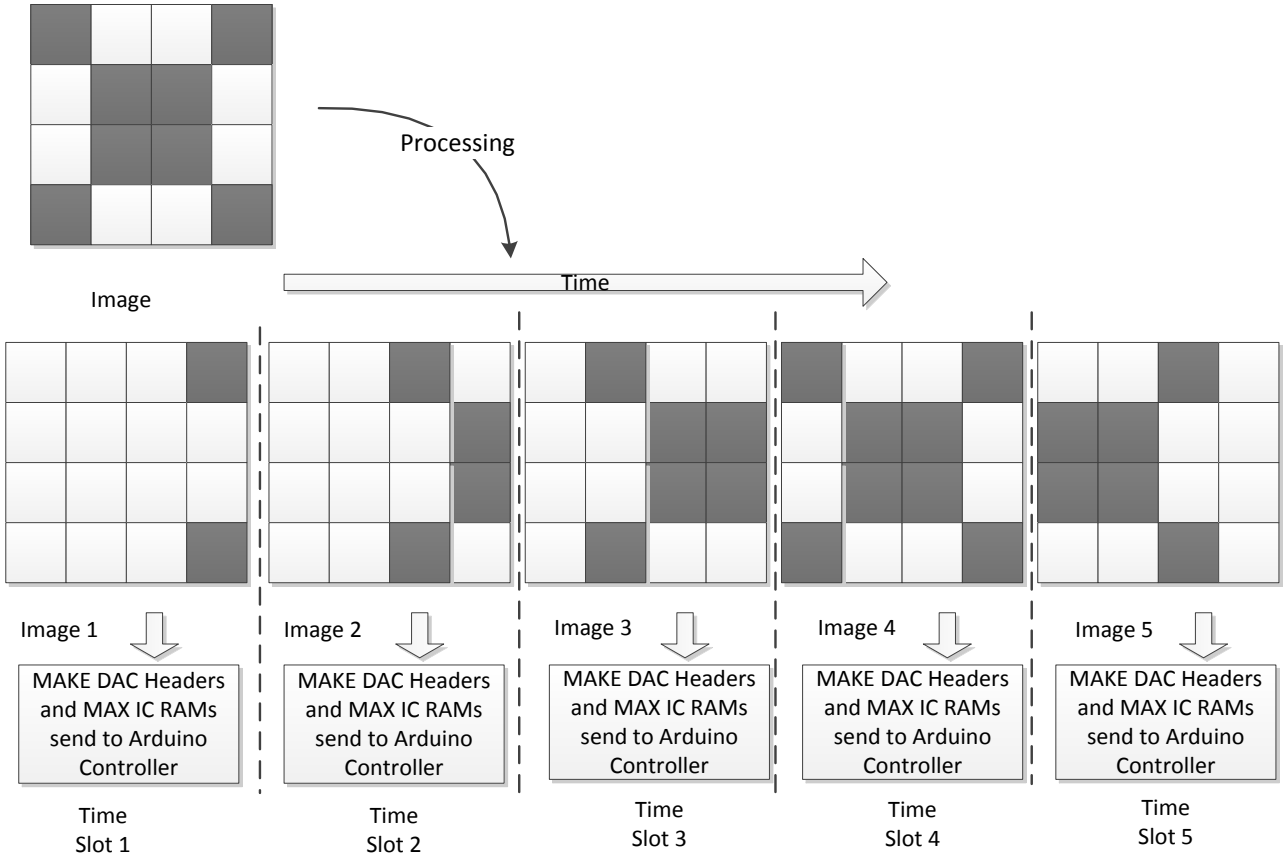


Figure 63: Example for processing of Image Pattern on Printed Displays.

In Figure 63, it is explained, how one can process an image, to make a series of images using python libraries on BeagleBone, data is formed suitable for MAX IC and DAC based controllers to output such sequence.

6.2.3. CHANGES IN CGI SCRIPT

Upon receiving CGI request, web server is designed so that it run a python file present in '/cgi-bin' folder.

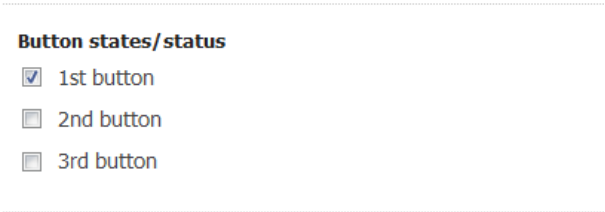


Figure 64. Button States (First Button Checked).

When http client request a CGI call (GET /cgi-bin/ajax_example.cgi) server run script and grab the output which is actually html code for web browser. Following python script generate html code for checked 1st button on input argument to the file based on Arduino reading is zero as shown in Figure 64.

```
if ip==0:
    check0="checked"
print  '<input id="element_2_1" name="element_2_1" class="element checkbox" type="checkbox" value="1" '
```

```
print check0,'>'
```

Actual output of script will be '`<input id="element_2_1" name="element_2_1" class="element checkbox" type="checkbox" value="1" checked>`' when variable input (CGI_ARG) is having 0 value.

This file i.e., *.py or *.cgi is responsible for updating a dynamic web page; one must change output of this script (a different html code) for different display on web browser.

7. DISCUSSION

Under this section of book, general discussion about some relevant topics is covered.

7.1. UTILIZATION OF RESOURCES

The final demonstration didn't utilize all of the resources whether physical or soft, so it can be stated this system can be useful in many other cases. For interfacing extra hardware with the system we are almost left with many physical resources (memory, 3D graphics accelerator, etc.), most of the GPIOs and many of the communication ports. TI document [15] discusses BeagleBone as initially developed for developers, hobbyists, and students providing details for few possible expansions. Figure 65 is taken from the document gives idea of expansion boards for BB.

Figure 65. Expansion boards for BeagleBone Black.

Above figure is provided here for reader to get idea of applications already made their way to market from makers of BB. This can give an insight to the processing power and resources available for very attractive applications for future. Consider following example for POS perspective to know about the processing power of the system.

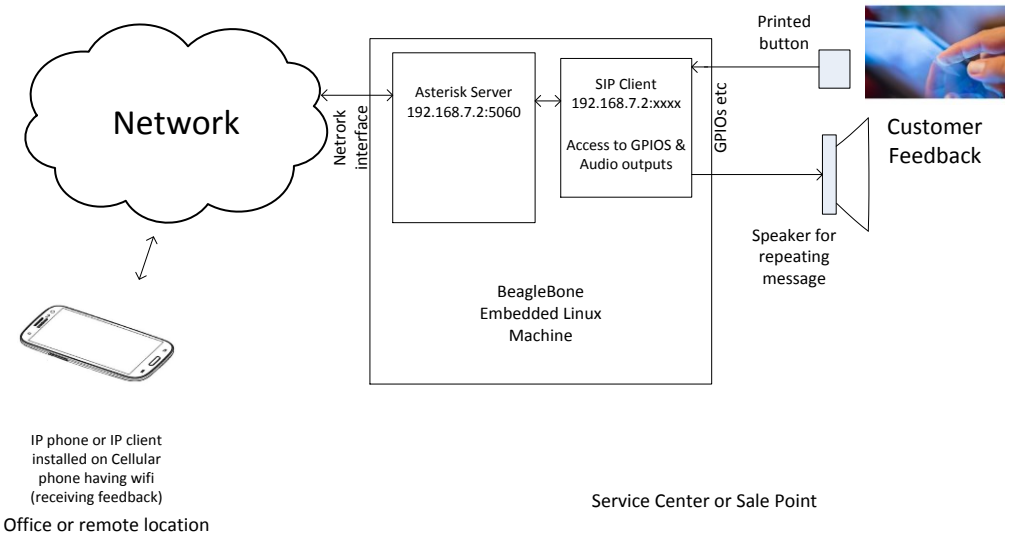


Figure 66. IP Exchange based feedback example on sale point.

Figure 66 shows an example application of BeagleBone alone, to make it a telephone exchange. For this, one can go towards choice of IP based exchange/switch rather than Circuit Switch as it has all the capabilities of IP switching but for circuit switch it lacks the physical resources. Resources required for an IP exchange, like open source Asterisk, are fulfilled and Linux images with Asterisk for BBB are already available online. Soft exchange like asterisk can really perform variety of tasks; for example, IVR, recording complaints remotely, making calls etc. Printed electronics circuits in this project can be used for add-ons with asterisk IP switch to make applications like help calls and complaints from point of sale. It might require writing code for soft phone (VoIP protocol client) application for this kind of demonstration. Overall asterisk application can be that one creates some extensions, few of them are allotted to company employees, and one extension is for user at restaurant or any service center who wants to record feedback. That user must be visible with at least one printed button and a microphone, or may be a speaker as well. One extension can be used for voice recording and by pressing printed button soft phone make call to extension for recording voice. Configurations can be made to record voice and saying 'thank you for feedback' and repeating user's message through speaker. Employees (with other extensions) can use their extensions to retrieve voice messages of feedback.

In above example only printed circuits and BB was used but one can also take advantage of microcontroller board as well. Let us assume one wants to go with already installed libraries (pre-installed in images), for this one must consider three parts of code: html code, BB programming framework & Arduino programming framework. HTML is more related to web browser display techniques rather than a proper language; only mentionable property can be which browser supports which multimedia formats (therefore it is skipped). Following Table 5 shows the different built-in facilities/support available (function libraries only) for Python and Arduino C.

Table 5. Libraries provided with both processing machines used in project

Python 2.7 [42]	Arduino C [43]
Built-in Functions like abs, all, any, basestring, input, open, str etc	Digital I/O
Num & Math: numbers, math, cmath, decimal, fractions, random, itertools, functools, operator	Analog I/O
File & Dir Access: os.path, fileinput, etc	Advanced I/O e.g. tone, shiftIn, PulseIn etc
Data Persistence: pickle, copy_reg, shelve etc	Time
Data compression and Archiving	Math
File Formats	Trigonometry
Cryptographic Services	Random
Generic Operating System Services	Bits and Bytes
Optional Operating Systems Services: thread, mmap etc	External and Internal Interrupts
Interprocess Communication and Networking	Comm. e.g. Serial, Stream
Internet Data Handling: email, json, rfc822 etc	Other Standard Libraries: EEPROM, Firmata, GSM, Servo, Wifi, TFT, Robot

Internet Protocols and Support: webbrowser, cgi, smtp ftplib etc	
Multimedia Services: audioop, imageop etc	
Other: Internationalization, Programm Frameworks (cmd etc), GUI with Tk (Tkinter etc), Development Tools (pydoc etc), Debugging and Profiling (bdb etc), Python Runtime Services (sys etc), Windows/Unix/MAC OS specific services and many more	
Add on Adafruit_BBIO for BBB: GPIO, PWM, ADC, I2C, SPI, UART	
Add on PyBBIO for BB: BBIOServer, GPIO, Serial etc	

In above table one can find allot of combination to make good use of available software in whole device. This is not it, BeagleBone is not even restricted to Linux, besides Angstrom Distribution, Ubuntu, Android, StarterWare (no-OS environment), Debian, ArchLinux, Gentoo, Sabayon, Buildroot, Erlang, Fedorda, QNX & FreeBSD had already made their ways to online community. So user with comfort level with any of the above system can switch according to wish.

7.2. EFFECTS OF LINUX KERNEL ON PROJECT

Linux kernel is defined in third chapter. It is high time to discuss effects of kernel on programming. At initial phase of the project BeagleBone (white) was selected comparing different Linux machines knowing its advantage of shape, size, processing power, GPIOs with minimal power requirements. One update came later from BeagleBoard.org itself; i.e., BeagleBone Black (BBB). BBB came with Angstrom Linux Kernel 3.8 as previously used Kernel 3.2 for angstrom distribution for BeagleBone (white) wasn't suitable. To produce copies of the project compatibility was needed in term of code which unfortunately, wasn't there. As libraries written for older kernel were useless with new system and update was delayed by the developers. One of two possibilities are (a) to use old Linux kernel with new board failed, leaving an option to use different libraries for new board (b) to change older libraries for new Kernel which wasn't tried due to lack of time to study differences in device tree used.

8. SUMMARY

Printed electronics technology is emerging, and expecting to grasp large market size, as it has advantage of low cost substrates, also overall low fabrication cost with speedy printing process on large areas. Currently research has been going on to develop printing process with lower trace widths plus more reliability (for competing conventional silicon electronics) and it is also expected to revolutionize role of electronics in modern lives after achieving the goal. When it comes to applications of printed electronics, one can find RFID tags, sensor array, and displays as most described applications. But this is not the actual case; i.e., these are standalone applications which already made their way after challenging conventional electronics. Actually printed electronics is passing through a phase where stability, speed, and reliability is compared with more stable, reliable, and speedy silicon based electronics, so major developments in wide (scope) areas are rare at present, as hinted earlier but it is expected to spread use of printing in many potential applications in near future.

A demonstration device was needed for the purpose of interfacing some of independent printed components such as sensor matrices and displays. Embedded Linux based solution was preferred for the problem, as use of Linux as embedded OS has caught significant attention because of its compatibility for many types of computer architectures with efficient resource management which is a limited commodity in embedded systems, and is highly scalable from small applications to large systems like switches and routers etc. Big online community is a significant advantage to assure circumvention of loophole amid development phase. Typical embedded Linux setup and application development procedure is very easy to understand and often happen to be same for different kinds of embedded Linux machines available in market. BeagleBone is an embedded Linux board which was used in this project and angstrom Linux distribution was used which is usually shipped with it. There are many options available in terms of Linux distributions and boards but BB with angstrom Linux was very desirable solution.

System structure of whole device was that, it had a microprocessor Board (Linux Board) for all central processing. Power board was added to system so that it charges the battery and boast-convert output to stable input voltage for all boards. Interface board was designed to interface capacitive sensor matrix, resistive touch screen, electrophoretic display, OLED display matrix, and electrochromic display matrix. Interface board had I2C & SPI communication buses IO header to take in commands and give back measurement data. An additional microcontroller board was later added in between Linux machine and IB, so that in future this device can be turned into a low power machine. An I-V characterization board was added with Linux machine to characterize organic photovoltaic cells (printed solar cells), its purpose was to acquire data and decide whether solar cell under current environment condition is able to provide enough power to system or not, and in case yes, its supply can be connected to input of power board.

For software structure, all the work done was on application level. Application program contained multi-threaded process which can communicate via common variables. One thread represented HTTP server and other represented hardware control. Hardware thread was responsible for communication with microcontroller, it exchanged messages: downloading configuration messages taken from user to microcontroller, and taking input data from sensors etc. HTTP server thread was responsible for providing user interface in form of web browser page and taking configuration messages and image diagrams to convert them into data words etc. Microcontroller was programmed to control all IB controllers via I2C and SPI communication buses. It took measurement data from capacitive and resistive sensors AD7147 & TSC2003, sent them to hardware thread running on Linux and took configuration data from it and played demo blinking of led string (for, example demo, it is capable of downloading complete images though) accordingly. Then there is an extra part (wasn't part of example demonstration but can be easily integrated) of

software which control IV characterization board, it sweeps voltage from 0 to max on MOS transistors' gates via DACs available on board, and measure current and voltage via ADCs.

An example demonstration was made at last, joining all hardware and software parts, which shown: user control of blinking of LEDs, using printed resistive buttons and HTTP browser, and displaying proximity detection on web browser. It didn't use full capabilities of embedded Linux and microcontroller resources. One can develop the system more, to add much more functionalities; for example, interactive voice response using open source IP exchange Asterisk.

In conclusion it can be comfortably established that currently printed components are able to interface with conventional electronics with reliability, but in future one may see fully developed standalone printed systems.

9. REFERENCES

[1] Printed Electronics, Wikipedia, (accessed 12.01.2014). URL: http://en.wikipedia.org/wiki/Printed_electronics

[2] Leenen M.A.M., Arning V., Thiem H., Steiger J., Anselmann R. (2009) Printable electronics: flexibility for the future. Phys Status Solidi A, 206, pp. 588–597

[3] Clemens W., Fix W., Ficker J., Knobloch A. & Ullmann A. (2004). From polymer transistors toward printed electronics. Journal of Materials Research, 19, pp 1963-1973. doi:10.1557/JMR.2004.0263.

[4] Hollabaugh C. (2004), Embedded Linux: Hardware, Software, and Interfacing, Addison-Wesley

[5] Vivek S., Chang J. B., Vornbrock A. d. l. F., Huang D. C., Jagannathan L., Liao F., Mattis B., Molesa S., Redinger D. R., Soltman D., Volkman S. K., Zhang Q. (2008) Printed Electronics For Low-Cost Electronic Systems: Technology Status and Application Development, Solid-State Device Research Conference.

[6] Loher T., Heinrich D. M, R., Schmied B., Vanfleteren J., DeBaets J., Ostmann A., and Reichl H. (2006) Stretchable electronic systems, Electronics Packaging Technology Conference (EPTC).

[7] Press Releases: Thinfilm Unveils First Scalable Printed CMOS Memory, THINFILM, (accessed 17.1.2014). URL: http://www.thinfilm.no/news/thinfilm-unveils-first-scalable-printed-cmos-memory/

[8] Press Releases: Thinfilm Builds First Stand-Alone Sensor System in Printed Electronics, THINFILM, (accessed 17.1.2014). URL: http://www.thinfilm.no/news/stand-alone-system/

[9] LG latest to announce curved smartphone: LG Z to sport flexible display and already in production, Yahoo News, (accessed 19.1.2014). URL: http://uk.news.yahoo.com/lg-latest-announce-curved-smartphone-lg-z-sport-093100584.html#ZpP5D5s

[10] Rida A., Yang L., Vyas R., Bhattacharya S., and Tentzeris M. M. (2007) Design and Integration of Inkjet-printed Paper-Based UHF Components for RFID and Ubiquitous Sensing Applications, Proceedings of the 37th European Microwave Conference.

[11] Printed displays flex their muscles, Engineering and Technology Magazine, (accessed 18.1.2014). URL: http://eandt.theiet.org/magazine/2008/10/flex-muscles.cfm

[12] Barr M. & Massa A. (2006) Programming Embedded Systems: with C and GNU Development Tools, 2nd ed, O'REILLY

[13] Berger A. S., Embedded Systems Design: An Introduction to Processes, Tools, Techniques, CMP Books, USA

[14] Hallinan C. (2010) Embedded Linux Primer: A Practical Real-World Approach, 2nd Ed, PRENTICE HALL

[15] Kridner J., & Coley G. (2013) BeagleBone Black opensource Linux™ computer unleashes innovation, (accessed 23.1.2014). URL: http://www.ti.com/lit/wp/spry235/spry235.pdf

[16] BeagleBone Ångström Distribution, BeagleBoard.org, (accessed 27.1.2014). URL: http://beagleboard.org/project/angstrom

[17] Home Page, Arch Linux ARM, (accessed 27.1.2014). URL: http://archlinuxarm.org/

[18] Home Page, Nerves-Project, (accessed 30.1.2014). URL: http://nerves-project.org/

[19] Coley G. (2012) BeagleBone Rev A6, System Reference Manual, Rivision 0.0, BeagleBoard.org, (accessed 10.12.2013). URL: http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf

[20] Beagleboard:BeagleBoneBlack, eLinux.org, (accessed 11.12.2013).URL: http://elinux.org/Beagleboard:BeagleBoneBlack

[21] Getting Started, BeagleBone.org, (accessed 11.12.2013).URL: http://beagleboard.org/Getting%20Started

[22] Raspberry Pi, Wikipedia, (accessed 25.1.2014). URL: http://en.wikipedia.org/wiki/Raspberry_pi

[23] Data Sheet of LT3652, Power Tracking 2A Battery Charger for Solar Power, Linear Technology, (accessed 11.12.2013). URL: http://cds.linear.com/docs/en/datasheet/3652fd.pdf

[24] Data Sheet of LT1935, 1.2MHz Boost DC/DC Converter in ThinSOT with 2A Switch, Linear Technology, (accessed 12.12.2013). URL: http://cds.linear.com/docs/en/datasheet/1935f.pdf

[25] Capacitive Sensing Library, Arduino, (accessed 13.12.2013). URL: http://playground.arduino.cc//Main/CapacitiveSensor?from=Main.CapSense

[26] Datasheet: Atmel 8-bit Microcontroller with 4/8/16/32Kbytes In-System Programmable Flash, Atmel, (accessed 13.12.2013). URL: http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet.pdf

[27] Datasheet: CapTouch Programmable Controller for Single-Electrode Capacitance Sensors, Analog Devices, (accessed 13.12.2013). URL: http://www.analog.com/static/imported-files/data_sheets/AD7147.pdf

[28] Finland Section of Farnell, Farnell, (accessed 3.2.2014). URL: http://fi.farnell.com

[29] Datasheet: TSC2003, Burr-Brown Products from Texas Instruments, (accessed 16.12.2013). URL: http://www.ti.com/lit/ds/symlink/tsc2003.pdf

[30] Brenner N., Sullivan S., & Goh W. (2008), 4-Wire and 8-Wire Resistive Touch-Screen Controller, Texas Instruments, (accessed 28.2.2014). URL: http://www.ti.com/lit/an/slaa384a/slaa384a.pdf

[31] Datasheet: MAX6953 Matrix LED Display Driver, Rev 3, Maxim, (accessed 16.12.2013). URL: http://datasheets.maximintegrated.com/en/ds/MAX6953.pdf

[32] Datasheet for TLC5920, Texas Instruments, (accessed 4.2.2014). URL: http://www.ti.com/lit/ds/symlink/tlc5920.pdf

[33] Data Sheet: ADG1414, ANALOG DEVICES, (accessed 18.12.2013). URL: http://www.analog.com/static/imported-files/data_sheets/ADG1414.pdf

[34] Datasheet: LTC2635, Linear Technology, (accessed 19.12.2013). URL: http://cds.linear.com/docs/en/datasheet/2635fb.pdf

[35] Datasheet: LM224 Single Supply Quad Operational Amplifiers, ON Semiconductor, (accessed 19.12.2013). URL: http://www.farnell.com/datasheets/81842.pdf

[36] ArduinoBoardMini, ARDUINO, (accessed 21.12.2013). URL: http://arduino.cc/en/Main/ArduinoBoardMini

[37] Python 2.6, Python.org, (accessed 28.2.2014). URL: http://docs.python.org/release/2.6/

[38] Common Gateway Interface, Wikipedia, (accessed 28.2.2014). URL: http://en.wikipedia.org/wiki/Common_Gateway_Interface

[39] Arduino-Setup, ARDUINO, (accessed 31.12.2013). URL: http://arduino.cc/en/Reference/Setup

[40] Arduino-Loop, ARDUINO, (accessed 31.12.2013). URL: http://arduino.cc/en/Reference/loop

[41] Hypertext Transfer Protocol, Wikipedia, (accessed 1.9.2014). URL: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[42] Python 2.6, Python.org, (accessed 2.3.2014). URL: http://www.python.org/download/releases/2.7/

[43] Language Reference, Arduino, (accessed 2.3.2014). URL: http://arduino.cc/en/Reference/HomePage#.UxNXgj_iHq4

10. APPENDICES

Appendix 1: Python code for main program

```
#code for BBB is exempted, this code is for BB
import sys
import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web
from tornado.options import define, options
import threading
import os
import subprocess
os.system("clear")
import random
import os
from bbio import *

fp = open('./html/form/form.html', 'rb')
msg = fp.read()
fp.close()

Port = 8000

Command_to_Thread = 0
Flag = 0
CGI_ARG=0

on_time = 60
cycle_length = 3
led_on_off = 0



define("port", default=Port, help="run on the given port", type=int)




def setup():
  # Start Serial2 at 9600 baud:
  Serial2.begin(9600)
  pinMode("GPIO2_7", OUTPUT)

def getword():

    data = ''

    data = Serial2.read()
    word_data = ord(data)
    data = Serial2.read()
    word_data += ord(data)<<8
    return word_data

def getbyte():


    data = ''

    data = Serial2.read()
    byte_data = ord(data)
```

```
    return byte_data

res = {1: 'triangle', 2: 'square', 3:'minus', 4:'plus', 5:'nothing'}

def loop():

    global CGI_ARG
    global on_time
    global cycle_length
    global led_on_off
    digitalWrite("GPIO2_7", LOW)
    if (Serial2.available()):
        digitalWrite("GPIO2_7", HIGH)
        # There's incoming data
        #print "---------------"
        os.system("clear")
        r = getbyte()
        print res[r]

        c = getword()
        print c
        if c>40000:
            CGI_ARG = 0
            print 'capacitive switch 0'
        if c<30000:
            CGI_ARG = 1
            print 'capacitive switch 1'

        c = getword()
        print c
        if c>40000:
            CGI_ARG = 0
            print 'capacitive switch 0'
        if c<30000:
            CGI_ARG = 2
            print 'capacitive switch 2'

        print "transmitting 0xff to spi switch"
        Serial2.write(0xff) #way to write 0xff to arduino -- for spi switch

        print "transmitting 0 or 1 to led"
        if res[r]=='triangle':
            Serial2.write(0x01)
        else:
            Serial2.write (0x00)

        print "transmitting 0 or 1 to ec"
        if res[r]=='plus':

            Serial2.write(0x01)
        else:
            Serial2.write(0x00)
        print on_time
        print cycle_length


        if res[r]!='nothing':
            #CGI_ARG = 2
            pass
```

```
        Serial2.write(cycle_length)
        Serial2.write(on_time)


class MainHandler(tornado.web.RequestHandler):

    def get(self):
        global Flag
        global Command_to_Thread
        print "server at port:",Port
        print "main handler"
        self.redirect("/html/form/form.html")



class FormHandler(tornado.web.RequestHandler):

    def post(self):
        global Flag
        global Command_to_Thread
        global on_time
        global cycle_length

        try:
            file_data = self.request.files['image_filename'][0]
            output_file = open("uploads/" + file_data['filename'], 'w')
            output_file.write(file_data['body'])
        except:
            pass

        try:
            cycle_length = int(self.get_argument('led_cycle_length'))
            print 'led_cycle_length'
            print cycle_length
        except:
            pass
        try:
            on_time =int( self.get_argument('led_on_time'))
            print 'led_on_time'
            print on_time
        except:
            pass
        self.redirect("/html/form/form.html")


class CGIHandler(tornado.web.RequestHandler):

    def get(self):
        global Flag                 #Global variable
        global Command_to_Thread    #Global variable
        global CGI_ARG              #Global variable
        command=''                  #initialization
        print "server at port:",Port
        print "cgi handler"
        command = "python ./cgi-bin/ajax_example.py " #shell command (ajax_example is
python file to be run, its output is grabbed and sent to HTTP client)
        a = random.randrange(3)  #for test purpose #randomly change the switch status
#not used in demo
```

```
        command += str(CGI_ARG)  #string concatenate
        output = subprocess.check_output(command, shell=True)
        self.write(output)
        #print "command :",command        #for test purpose
        #print "output"                   #for test purpose
        #print output                     #for test purpose
        print CGI_ARG
        #CGI_ARG = 0
class ThreadClass_Hardware(threading.Thread):       # In this class, hardware thread is
defined,
                                                    # Loop function contain all of the
hardware control sequence
    def run(self):                                  # Global variables describes the
shared variables between main thread and hardware thread
        global Flag
        global Command_to_Thread
        global CGI_ARG
        global led_on_off #for extention purpose used with timer thread, not used here.
        global cycle_length #input from http form
        global on_time  #input from http form

        #Code of thread which will controll hardware
        run(setup, loop)

class ThreadClass_Timer(threading.Thread):
    def run(self):
        while 1:
            global on_time
            global cycle_length
            global led_on_off
            #print on_time, cycle_length
            time = 1/cycle_length
            time = on_time * time / 100
            time = time * 1000
            led_on_off = 1
            #print "led_on_off:",led_on_off
            delay(time)
            led_on_off = 0
            #print "led_on_off:",led_on_off
            delay (1000/cycle_length-time)

def main():
    global Flag                         #global variables shared by all functions
    global Command_to_Thread            #global variables shared by all functions
    global CGI_ARG                      #global variables shared by all functions
    global on_time                      #global variables shared by all functions
    global cycle_length                 #global variables shared by all functions
    global led_on_off                   #global variables shared by all functions
    CGI_ARG=0               #initialization
    Flag = 0                #initialization
    Command_to_Thread = 0   #initialization
    on_time = 60            #initialization
    cycle_length = 3        #initialization
    led_on_off = 0          #initialization
    print 'main program running at pid = ',os.getpid() #PID display (not necessary
block)
    t=ThreadClass_Hardware()        #hardware thread
    t.start()                       #starts here!
    #t2=ThreadClass_Timer()
```

```
    #t2.start()
    tornado.options.parse_command_line()
#HTTP server code starts here
    application = tornado.web.Application([
#application definition
        (r"/", MainHandler),
#constituents of application: main handler
        (r"/form", FormHandler),
#constituents of application: Form handler
        (r"/cgi-bin/ajax_example.cgi", CGIHandler),
#constituents of application: CGI handler
        (r"/html/form/(.*)",tornado.web.StaticFileHandler, {"path": "./html/form"},),
#constituents of application: static file handler

    ])
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(options.port)
    tornado.ioloop.IOLoop.instance().start()




if __name__ == "__main__":
    main()
```

Appendix 2: HTML code for form.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<script>
function loadXMLDoc()
{
var xmlhttp;
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("buttons").innerHTML=xmlhttp.responseText;
  setTimeout('loadXMLDoc()',1000);
    }
  }
xmlhttp.open("GET","/cgi-bin/ajax_example.cgi",true);
xmlhttp.send();
}
window.onload=function(){
setTimeout('loadXMLDoc()',1000);
}
</script>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>VTT Printed Intelligence Demonstration </title>
<link rel="stylesheet" type="text/css" href="view.css" media="all">
<script type="text/javascript" src="view.js"></script>

</head>
<body id="main_body" >

  <img id="top" src="top.png" alt="">
  <div id="form_container">

    <h1><a>VTT Printed Intelligence Demonstration </a></h1>
    <form id="form_701242" class="appnitro" enctype="multipart/form-data" method="post"
action="/form">
          <div class="form_description">
      <h2>VTT Printed Intelligence Demonstration </h2>
      <p>Example demonstration by the PAMODE project

This demonstration shows some of the capabilities of the hardware developed in the
PAMODE project. Specifically, a LED strip or a electro cromic display is controlled by
capacitive switches. Blinking rate and duty cycle are controllable as well as the image
displayed in the EC diaplay.</p>
    </div>
      <ul >

          <li id="li_7" >
    <label class="description" for="element_7">Select display </label>
```

```
    <span>
      <input id="element_7_1" name="element_7" class="element radio" type="radio"
value="1" checked="checked"/>
<label class="choice" for="element_7_1" >LED strip</label>
<input id="element_7_2" name="element_7" class="element radio" type="radio" value="2"
/>
<label class="choice" for="element_7_2">Electrocromic</label>

    </span>
    </li>   <li id="li_5" >
    <label class="description" for="element_5">Upload an image file for the EC display
</label>
    <div>
      <input id="element_5" name="image_filename" class="element file" type="file"/>
    </div>

    </li>   <li id="li_2" class="section_break" >
    <label class="description" for="element_2">Button states/status </label>
    <span id="buttons">
      <input id="element_2_1" name="element_2_1" class="element checkbox"
type="checkbox" value="1" />
<label class="choice" for="element_2_1">1st button</label>
<input id="element_2_2" name="element_2_2" class="element checkbox" type="checkbox"
value="1" />
<label class="choice" for="element_2_2">2nd button</label>
<input id="element_2_3" name="element_2_3" class="element checkbox" type="checkbox"
value="1" />
<label class="choice" for="element_2_3">3rd button</label>

    </span><p class="guidelines" id="guide_2"><small>Check boxes allow control of
effects and display the status of the capacitive switches listed below

1st button = capacitive button no 1
2nd button = capacitive button no 2
3rd button = capacitive button no 3</small></p>
    </li>   <li class="section_break">
      <h3>Effect (blinking) specifications</h3>
      <p></p>
    </li>   <li id="li_3" >
    <label class="description" for="element_3">Cycle length in seconds </label>
    <div>
      <input id="element_3" name="led_cycle_length" class="element text medium"
type="text" maxlength="255" value="2"/>
    </div>
    </li>   <li id="li_4" >
    <label class="description" for="element_4">LED-on time in % of above cycle time
</label>
    <div>
      <input id="element_4" name="led_on_time" class="element text medium" type="text"
maxlength="255" value="50"/>
    </div>
    </li>

          <li class="buttons">
          <input type="hidden" name="form_id" value="701242" />
          <input type="submit" value="Submit">
    </li>
      </ul>
    </form>
```

```
    <div id="footer">
      Generated for VTT by the PAMODE project 2013
    </div>
  </div>
  <img id="bottom" src="bottom.png" alt="">
  </body>
</html>
```

Appendix 3: C code for Arduino Board

```
#include <SoftwareSerial.h>
#include <CapacitiveSensor.h>
#include <Wire.h>
#include "SPI.h"
int ss=10; //in this demo, ss is select pin for spi of switch controllers
//here daisy chain topology was followed, but later with Beagle Bone Black there were
two chip select pins ss1 & ss2

SoftwareSerial mySerial(2, 3); // RX, TX
//There is one regular Serial Port, User can define several serial ports by
SoftwareSerial Library


CapacitiveSensor   cs1 = CapacitiveSensor(4,5);        // 10 megohm resistor between
pins 4 & 5, pin 5 is sensor pin, add wire, foil
CapacitiveSensor   cs2 = CapacitiveSensor(4,6);        // 10 megohm resistor between
pins 4 & 6, pin 6 is sensor pin, add wire, foil
//CapacitiveSensor was not used with this demo but can be useful in future
demonstrations

void setup()
{


       pinMode(7, INPUT_PULLUP);// this was thought to be for sync, but it is also un-
necessary
  while (digitalRead(7)==1)
  {
  Serial.println("BB program not started");
  }


  Serial.begin(9600); //serial for arduino software
  mySerial.begin(9600); //user defined serial at pins 2,3
  //mySerial.println("Hello, world?");
  Wire.begin(); // join i2c bus (address optional for master)


pinMode(ss, OUTPUT);
SPI.begin();
//SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SPI_MODE3);
SPI.setClockDivider(SPI_CLOCK_DIV128);
}

//spi fn starts

void setValue(byte value)
{
digitalWrite(ss, LOW);
SPI.transfer(value);
SPI.transfer(value);
digitalWrite(ss, HIGH);
}

//fpi fn ends
void SendWord(int input)
```

```
{    byte sendtobb;
     sendtobb = input;
     mySerial.write(sendtobb);
     sendtobb = input>>8;
     mySerial.write(sendtobb);

}

void SendByte(byte input)
{    byte sendtobb;
     sendtobb = input;
     mySerial.write(sendtobb);

}

unsigned int total1;
unsigned int total2;

//total1 = 0;
//total2 = 0;

byte b= 0;
//unsigned int var = 0;
unsigned int Xpos = 0;
unsigned int Ypos = 0;




//////////////////////////////////////////Cap Ad
fn//////////////////////////////////////



unsigned int CAP_AD7147_I(byte in1, byte in2, byte in3, byte in4)
{

 unsigned int var_AD7147;
 Wire.beginTransmission(0x2C); // AD7147 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0x00);            //
  Wire.write(0x80);            //
  Wire.write(in1);
  Wire.write(in2);

  Wire.endTransmission();      // stop transmitting
  delay(10);
  ////////////////////////////////////////////////////////////////////////////////
  Wire.beginTransmission(0x2C); // AD7147 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0x00);            //
  Wire.write(0x81);            //
  Wire.write(in3);            //
  Wire.write(in4);            //
  Wire.endTransmission();      // stop transmitting
  delay(10);
```

```
////////////////////////////////////////////////////////////////////////////////
  Wire.beginTransmission(0x2C); // AD7147 address
                                // device address is specified in datasheet
                                // sends instruction byte  (command)
  Wire.write(0x00);             //
  Wire.write(0x0B);             //
  Wire.endTransmission();       // stop transmitting
  delay(10);



////////////////////////////////////////////////////////////////////////////////
  Wire.requestFrom(0x2C, 2);    // request 2 bytes from slave device 0x2C
  byte b_AD7147;

    b_AD7147 = Wire.read(); // receive a byte
  var_AD7147 =  b_AD7147*256;
  //Serial.print(b_AD7147); Serial.print (" , ");
    b_AD7147 = Wire.read(); // receive a byte
  var_AD7147 = var_AD7147 + b_AD7147;
  //Serial.print(b_AD7147); Serial.print (" -> ");
  //Serial.println(var_AD7147);
  delay(10);
return var_AD7147;

}

//////////////////////////////////////////Cap Ad
fn//////////////////////////////////////


// LED (voltage) //// LED (voltage) //// LED (voltage) //// LED (voltage) //

byte EC_Disp=0;
byte LED=0;



void DAC_Op(byte ad, byte reg, byte val)
{
//bus.write_word_data(0x12,0x2f,0) //SMBUS for initializing dac with zero
  Wire.beginTransmission(ad); // transmit to all Dacs
  Wire.write(reg);        // sends byte
  Wire.write(val);              // sends byte
  Wire.write(0x00);              // sends byte
  Wire.endTransmission();    // stop transmitting
}



byte DAC_Address[10] =   {0x12, 0x12, 0x12, 0x12, 0x11, 0x11, 0x11, 0x11, 0x10, 0x10};
byte DAC_Regesters[10] = {0x30, 0x31, 0x32, 0x33, 0x30, 0x31, 0x32, 0x33, 0x30, 0x31};

void writeword(unsigned int number, byte voltage)
{

  for (int i=0; i<10; i++)
  {
```

```
  Serial.print(number>>i & 1);
  byte tmp = 0;
  tmp = (number>>i & 1)* voltage*17;
  //Serial.println(tmp);
  DAC_Op(DAC_Address[i], DAC_Regesters[i], tmp);
  }
//  Serial.println("*****************");
Serial.println();
//DAC_Op(0x11, 0x33, 80);//For Test

}

void Initialize_Dacs()
{
//bus.write_word_data(0x12,0x2f,0) //SMBUS for initializing dac with zero
  Wire.beginTransmission(0x73); // transmit to all Dacs
  Wire.write(0x2f);        // sends byte
  Wire.write(0x00);              // sends byte
  Wire.write(0x00);              // sends byte
  Wire.endTransmission();    // stop transmitting
}



//electrochromic display (last two spare pins from voltage driver)
void Change_EC_State(byte state)
{



  if (state == 0) {
  DAC_Op(0x10, 0x33, 0);
  DAC_Op(0x10, 0x32, 0);
  }

  if (state == 1) {
  DAC_Op(0x10, 0x33, 51);
  DAC_Op(0x10, 0x32, 51);
  }


}


// LED (voltage) //// LED (voltage) //// LED (voltage) //// LED (voltage) //

byte sw;
byte ld;
byte ec;

////////////extra function for demo oct 14th

byte scan_resistive_sensor(){

    b= 0;

  //Serial.println(String("X Position"));
  Wire.beginTransmission(0x48); // TSC2003 address
                               // device address is specified in datasheet
```

```
                              // sends instruction byte  (command)
  Wire.write(0xC0);             // C = 1101 (command, measure X position) , 0 for PD1,
PD0, M, X
  Wire.endTransmission();     // stop transmitting
  delay(10);


  Wire.requestFrom(0x48, 2);    // request 2 bytes from slave device 0x48



    b = Wire.read(); // receive a byte
  Xpos =  b*16;
    b = Wire.read(); // receive a byte
  Xpos = Xpos + b/16;

  //Serial.println(Xpos);
  delay(10);

////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////

b= 0;


 //Serial.println(String("Y Position"));
  Wire.beginTransmission(0x48); // TSC2003 address
                              // device address is specified in datasheet
                              // sends instruction byte  (command)
  Wire.write(0xD0);             // C = 1101 (command, measure X position) , 0 for PD1,
PD0, M, X
  Wire.endTransmission();     // stop transmitting
  delay(10);

  Wire.requestFrom(0x48, 2);    // request 2 bytes from slave device 0x48

    b = Wire.read(); // receive a byte
  Ypos =  b*16;
    b = Wire.read(); // receive a byte
  Ypos = Ypos + b/16;

  //Serial.println(Ypos);
  delay(10);

////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////
Serial.println(String("Resistive Measurements Start"));
// Serial.print(Xpos);                  // test
// Serial.print("\t");
 //Serial.println(Ypos);                  // test
if (Xpos > 0 && Ypos < 4095)
  {
    //Serial.println(String("Detected"));
if (Xpos > 2000 && Ypos > 1866)
{Serial.println(String("Triangle"));
return byte (0x01);}
else if (Xpos > 2000 && Ypos < 1866)
{Serial.println(String("Square"));
return byte (0x02);}
```

```
else if (Xpos < 2000 && Ypos < 1866)
{Serial.println(String("Minus"));
return byte (0x03);}
 else
{Serial.println(String("Plus"));
return byte (0x04);}


  }
  else
  {Serial.println(String("Not Detected"));
return byte (0x05);}

}

////////////extra function for demo oct 14th
void loop() // run over and over
{
 Initialize_Dacs(); // added for demo 2 14 oct
//  while (digitalRead(7)==1)
//  {
//  Serial.println("BB program not started");
//  }

  //------------------Cap Sense Start-----------------------//
   //mySerial.println("Data Starts here ->");
   //mySerial.write('S'); // Start

   //mySerial.println("Data Ends here <-");
   Serial.println("Demo Running");

   //capacitive sensors through capsense

    // total1 =  cs1.capacitiveSensor(60);
    // total2 =  cs2.capacitiveSensor(60);

     //sendtobb = total1;
     //mySerial.write(sendtobb);
     //sendtobb = total1>>8;
     //mySerial.write(sendtobb);
     //Serial.println("still sending not started");
     //SendWord(total1);
     //SendWord(total2);

   //mySerial.print('s'); //stop
   //------------------Cap Sense Ends-----------------------//

   //------------------Resistive Switch Code Starts-----------------------//

    b= 0;

  //Serial.println(String("X Position"));
  Wire.beginTransmission(0x48); // TSC2003 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0xC0);             // C = 1101 (command, measure X position) , 0 for PD1,
PD0, M, X //Figure 11 of ref manual
  Wire.endTransmission();     // stop transmitting
  delay(10);
```

```
  Wire.requestFrom(0x48, 2);    // request 2 bytes from slave device 0x48



    b = Wire.read(); // receive a byte
  Xpos =  b*16;
    b = Wire.read(); // receive a byte
  Xpos = Xpos + b/16;

  //Serial.println(Xpos);
  delay(10);

////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////

b= 0;


 //Serial.println(String("Y Position"));
  Wire.beginTransmission(0x48); // TSC2003 address
                               // device address is specified in datasheet
                               // sends instruction byte  (command)
  Wire.write(0xD0);             // C = 1101 (command, measure X position) , 0 for PD1,
PD0, M, X
  Wire.endTransmission();     // stop transmitting
  delay(10);

  Wire.requestFrom(0x48, 2);    // request 2 bytes from slave device 0x48

    b = Wire.read(); // receive a byte
  Ypos =  b*16;
    b = Wire.read(); // receive a byte
  Ypos = Ypos + b/16;

  //Serial.println(Ypos);
  delay(10);

////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////
Serial.println(String("Resistive Measurements Start"));
// Serial.print(Xpos);                  // test
// Serial.print("\t");
 //Serial.println(Ypos);                  // test
if (Xpos > 0 && Ypos < 4095)
  {
    //Serial.println(String("Detected"));
if (Xpos > 2000 && Ypos > 1866)
{Serial.println(String("Triangle"));
SendByte(byte (0x01));}
else if (Xpos > 2000 && Ypos < 1866)
{Serial.println(String("Square"));
SendByte(byte (0x02));}
else if (Xpos < 2000 && Ypos < 1866)
{Serial.println(String("Minus"));
SendByte(byte (0x03));}
 else
{Serial.println(String("Plus"));
```

```
SendByte(byte (0x04));}


  }
  else
  {Serial.println(String("Not Detected"));
SendByte(byte (0x05));}

  Serial.println(String("Resistive Measurements End"));
   //------------------Resistive Switch Code Ends----------------------//



    // Cap Sense
Print**************************************************************************
**********************
   Serial.println(String("Cap Sense Starts"));

    //Serial.println(mySerial.read()/2);
    Serial.print(total1);                  // print sensor output 1      // commented
to test other part
    Serial.print("\t");
    Serial.println(total2);                  // print sensor output 2     // commented
to test other part

Serial.println(String("Cap Sense Ends"));

    // Cap Sense
Print**************************************************************************
**********************



   // Cap ad7147
starts*************************************************************************
**********************
Serial.println("------------**********----------------***************------------
----**********------------");
Serial.println("Capacitance AD7147");
unsigned int cap_op;
cap_op = CAP_AD7147_I(0x00,0x06,0xf0,0x00); //  Connect Sensor 1 to negative and Sensor
0 to positive input of Cap to digital converter differential inputs
SendWord(cap_op);
Serial.println(cap_op);
cap_op = CAP_AD7147_I(0x00,0x12,0xf0,0x00); //  Connect Sensor 2 to negative and Sensor
0 to positive input of Cap to digital converter differential inputs
SendWord(cap_op);
Serial.println(cap_op);
//cap_op = CAP_AD7147_I(0x00,0x20,0xf0,0x04); //  Connect Sensor 8 to negative and
Sensor 2 to positive input of Cap to digital converter differential inputs
//SendWord(cap_op);
//Serial.println(cap_op);
//cap_op = CAP_AD7147_I(0x00,0x80,0xf0,0x10); //  Connect Sensor 9 to negative and
Sensor 3 to positive input of Cap to digital converter differential inputs
//SendWord(cap_op);
//Serial.println(cap_op);
//cap_op = CAP_AD7147_I(0x00,0x00,0xf4,0x80); //  Connect Sensor 10 to negative and
Sensor 12 to positive input of Cap to digital converter differential inputs
//SendWord(cap_op);
```

```
//Serial.println(cap_op);
//cap_op = CAP_AD7147_I(0x08,0x00,0xf1,0x00); //  Connect Sensor 11 to negative and
Sensor 5 to positive input of Cap to digital converter differential inputs
//SendWord(cap_op);
//Serial.println(cap_op);

// upper code was commented because not all values were
//CAP_AD7147_I(0x00,0x02,0xf4,0x00);// Connect Sensor 12 to negative and Sensor 0 to
positive
Serial.println("-------------**********----------------***************-------------
----**********------------");
// Cap ad7147
ends**************************************************************************
********************

// SPI data to switch
   sw = mySerial.read();
   Serial.println("Data coming from beaglebone to switch IC ");
   Serial.println (sw);
setValue(byte(sw)); // origional
   //setValue(0xff);testing
// SPI data to switch



// LED
   ld = mySerial.read();
   Serial.println("Data coming from beaglebone to led ");
   Serial.println (ld);


  unsigned int input_word_dac = 0xffff;
  byte EC_State = 0;
  byte voltage = 9;
  //Initialize_Dacs();
  delay(10);
   //byte V = voltage * (ld & 0x01);
 //writeword(input_word_dac, V);
// LED

ec = mySerial.read();
Serial.println("EC state");
Serial.println(ec);
//Change_EC_State(ec); // commented in 2nd demo 14th oct
byte cycle_len;
byte on_time;
cycle_len= byte(mySerial.read());
on_time = byte(mySerial.read());
if (cycle_len>5)
{cycle_len=5;
}
if (on_time>90)
{on_time=90;
}

Serial.println(cycle_len);
Serial.println(on_time);

unsigned long ton;
```

```
unsigned long toff;
unsigned long T;
T = cycle_len *1000;
//ton = T * (on_time /100);
ton = T * on_time;
ton = ton /100;
toff = T - ton;
Serial.println(ton);
Serial.println(toff);
if (ld > 0){

  input_word_dac = 0b10;
  writeword(input_word_dac, 1);
  delay(1000);

  input_word_dac = 0b100;
  writeword(input_word_dac, 1);
  delay(1000);

  input_word_dac = 0b1000;
  writeword(input_word_dac, 1);
  delay(1000);

    input_word_dac = 0b10000000;
  writeword(input_word_dac, 1);
  delay(1000);

 Change_EC_State(1);
 delay(1000);
 Change_EC_State(0);

input_word_dac = 0b1;
int i=0;
while (i==0){
   byte V = voltage * (ld & 0x01);
   writeword(input_word_dac, V);
   Serial.println(voltage);
   byte n = scan_resistive_sensor();
   switch (n) {
  case 1:
    break;
  case 2:
    i =1;
    break;

  case 3:
   voltage -= 1;
   if (voltage < 5){
   voltage = 5;}
   break;
  case 4:
   Serial.println("yes");
   voltage += 1;
   if (voltage > 9){
   voltage = 9;}
   break;

  default:
    break;// statements
```

```
};
for (int k = 0; k<3; k++)
{  int ton_k = ton /3;
   delay(ton_k);
   byte n = scan_resistive_sensor();
   switch (n) {
  case 1:
    break;
  case 2:
    i =1;
    break;

  case 3:
   voltage -= 1;
   if (voltage < 5){
   voltage = 5;}
   break;
  case 4:
   Serial.println("yes");
   voltage += 1;
   if (voltage > 9){
   voltage = 9;}
   break;

  default:
    break;// statements
};
}
   writeword(input_word_dac, 0);
   for (int k = 0; k<3; k++)
   {int toff_k = toff/3;
   delay(toff_k);
   byte n = scan_resistive_sensor();
   switch (n) {
  case 1:
    break;
  case 2:
    i =1;
    break;

  case 3:
   voltage -= 1;
   if (voltage < 5){
   voltage = 5;}
   break;
  case 4:
   Serial.println("yes");
   voltage += 1;
   if (voltage > 9){
   voltage = 9;}
   break;

  default:
    break;// statements
};
   }
}

}
```

```
    delay(500);
}
```

Appendix 4: Understanding Steps to Make Demonstration Device for Printed Electronics

1. Turn on Beagle Bone
    I. Plug USB cable into windows computer and other side to Beagle Bone usb slot below four indication LEDs
    II. Eject USB appearing as Beagle Bone Drive, Therefore Beagle bone is acquiring IP address which is 192.168.7.2
    III. Shell root@192.168.7.2 with empty password
    IV. Make a Demo folder where demo files are intended to be placed
2. Install Tornado-2.4.1 in Beagle Bone Black (Tornado is python non-blocking fast web server, without tornado we can also make webservers but they are slower)
    I. You can search any version on main website http://www.tornadoweb.org/en/stable/
    II. need to update opkg and install distutils before installation root@beaglebone:/home/tornado-master/tornado-master# opkg install python-distutils
    III. Build Tornado root@beaglebone:/home/tornado-master/tornado-master# python setup.py build
    IV. Now it's time to install i.e. python setup.py install
    V. But it will fail with error "`File "/usr/lib/python2.7/distutils/util.py", line 532, in byte_compile`"
    VI. Probably Solution for the time being is to leave it as it is. atleast our test codes run without any problem. Also our demo code runs without any problem. Hint: If you don't manage to install Tornado, you can use regular python library for HTTP web servers. Typically SimpleHTTPServer class can be used. Or simply try again after installing "python-pip python-setuptools python-smbus" then it shouldn't fail.
3. Set up Beagle Bone Python IO Library (for Beagle Bone white PyBBIO library should be used, for Beagle Bone black Adafruit-BeagleBone-IO-Python should be used) here I am writing procedure to install Adafruit-BeagleBone-IO-Python on Beagle Bone Black with Linux Kernel 3.8. [ref: http://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/installation]
    I. update opkg and install dependencies: root@beaglebone:/# opkg update && opkg install python-pip python-setuptools python-smbus
    II. pip install Adafruit_BBIO
    III. If pip install command fails, you need to update date and time: /usr/bin/ntpdate -b -s -u pool.ntp.org and try again
    IV. In case of failuare you can also install it manually.
    V. You may need to install Python Serial Library (PySerial) opkg install python-pyserial and to enable UART through dts file.
    VI. In case one wants to enable another UART port for serial comm., He should find its dts file in Arduino BBIO library provided by Adafruit.
4. Setting up Hardware: Connection is so that, UART 2 pins of Beagle Bone correspond to Pin 21 and 22 of header P9 (P9_21 & P9_22).

5. We need to start demo files in Beagle Bone Black; we can also make it so that it starts automatically after booting. A programme downloaded in Arduino Microcontroller does not need any reset or something; it starts as power is given to microcontroller. 5V required to run Arduino Board can be provided by Beagle Bone Pins 5-8 of P9 Header (P9_5 to P9_8). Or can be given separately. We have a power board in our project which is sufficient to power on beagle bone and Arduino board.
6. For connection detail, one must know about the headers information of Interface board, Arduino Microcontroller board and Beagle Bone/Beagle bone Black. One can read Expansion Header P9 Pinout of Beagle Bone / Beagle Bone Black to know I2C2 pins are 19 (SCL) and 20 (SDA). UART2 [configured with beagle bone black] pins are 21 (TXD) and 22 (RXD). Connect power pins to IV curve tracing board and I2C2 pins of Beagle Bone there without level translation. Level translation is required between UART2 of beagle bone and Arduino board. UART2 pins are connected with (configured mySerial) 2 and 3 of Arduino. For Arduino connections with Interface board, i.e. I2C and SPI one should look for pin order. Details presented in Figure A4.1, A4.2, & A4.3.

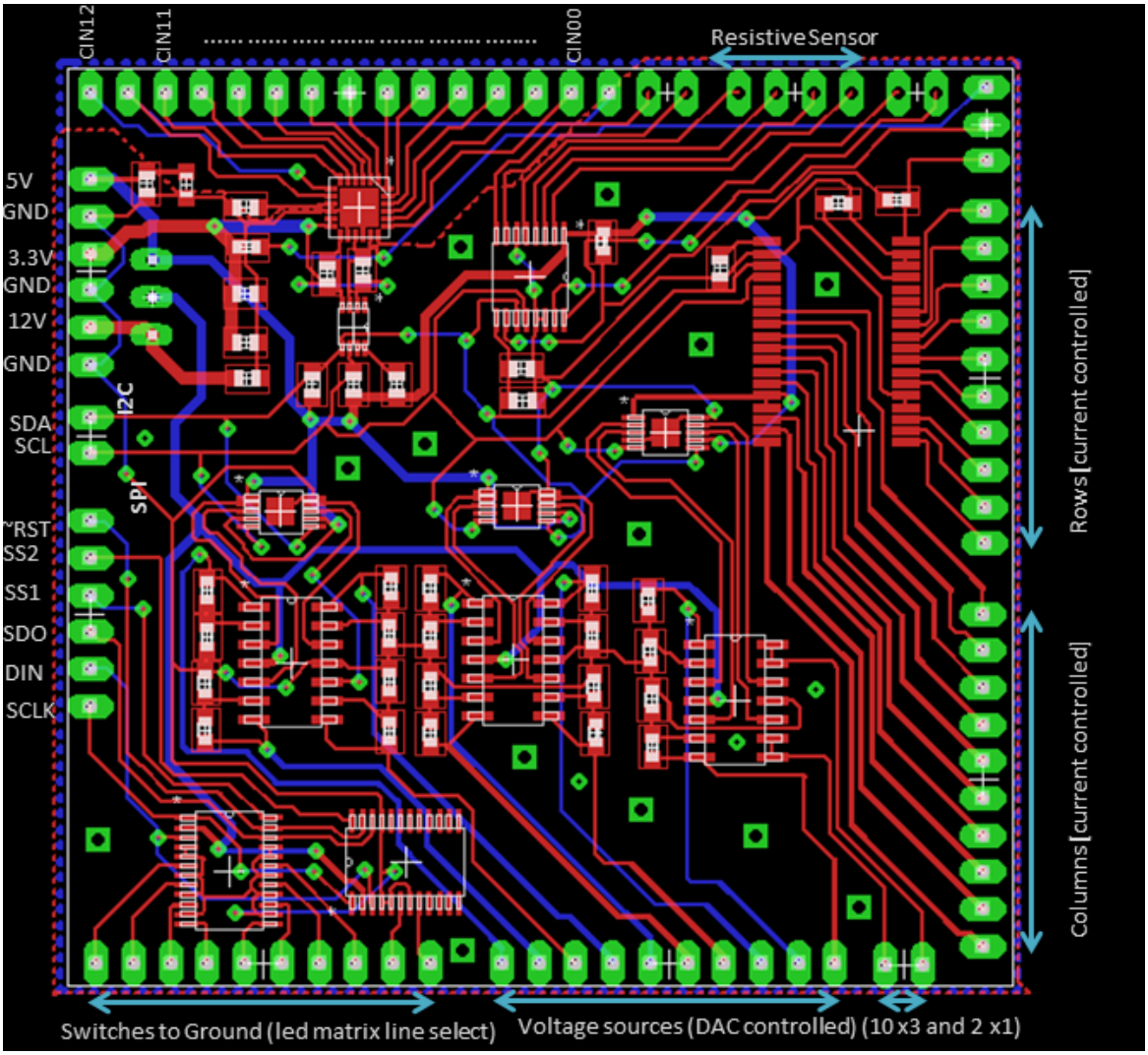


Figure A4.1. Interface Board layout and pin information.

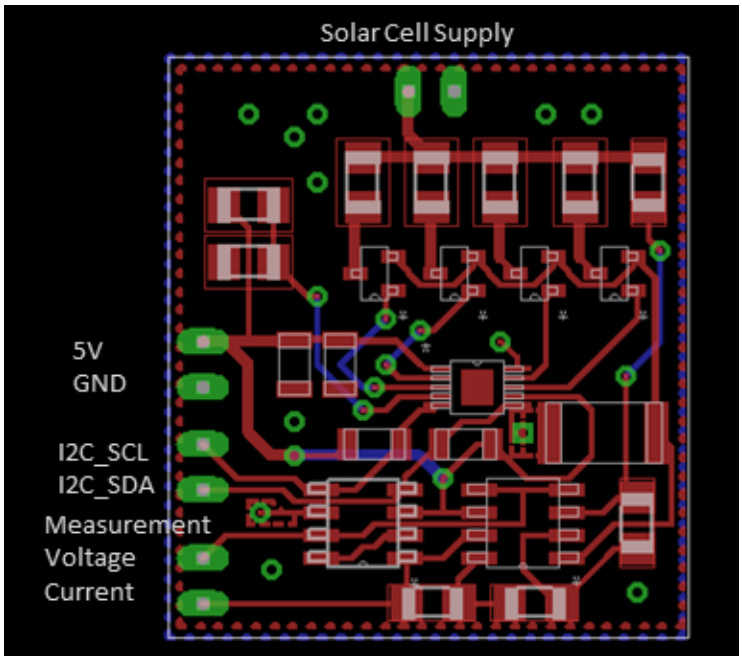


Figure A4.2. I-V curve tracer pin detail.

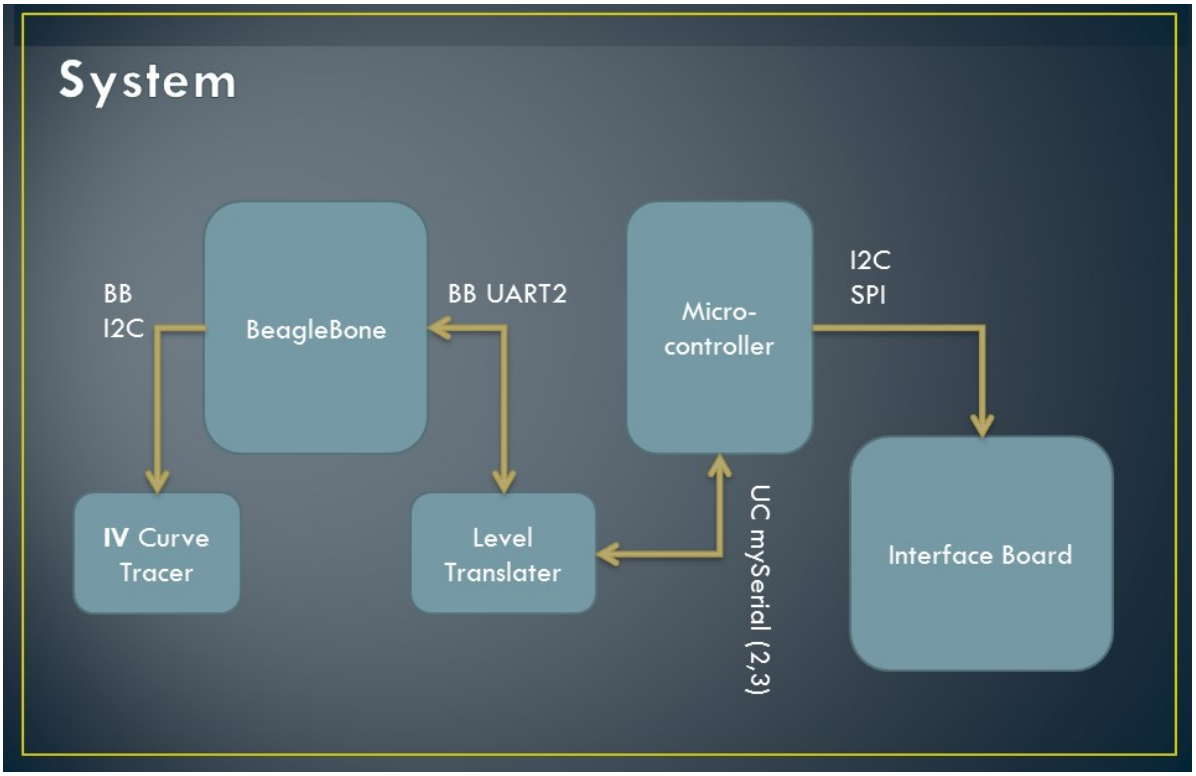


Figure A4.3. Internal Connections of device.

Appendix 5: Code for IV curve tracing

```
#Main code for IV curve tracing
#with server running at following location and port
#root@beaglebone:/home/work/dist# python -m SimpleHTTPServer 8002


import sys
import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web
from tornado.options import define, options
from html_graph import *


Port = 8003


#fp = open('./HTML/gr1.htm', 'rb')
#msg1 = fp.read()
#fp.close()

#global msg2
#msg2 = ''

#fp = open('./HTML/gr2.htm', 'rb')
#msg3 = fp.read()
#fp.close()



define("port", default=Port, help="run on the given port", type=int)


class MainHandler(tornado.web.RequestHandler):

    def get(self):


        #global msg2
        #msg2 += ''
        #msg = msg1 + msg2 + msg3
        self.write(IV_curve_measurement())
        #self.write(msg)
        print "server at port:",Port




def main():



    tornado.options.parse_command_line()
    application = tornado.web.Application([
        (r"/", MainHandler)
    ])
```

```
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(options.port)
    tornado.ioloop.IOLoop.instance().start()



if __name__ == "__main__":
    main()


#html_graph.py

import sys
from test_I2C2 import coordinates
def IV_curve_measurement():


    m = ''
    m += ' <html>'
    m += '  <head>'
    m += '     <meta http-equiv="Content-Type" content="text/html; charset=utf-8">'
    m += '     <title>University of Oulu Demo</title>'
    m += ' '
    m += ' <br>'
    m += '    <div id="chartdiv" style="height:400px;width:1000px; "></div>'
    m += ' <br>'



    m += '     <!--[if IE]><script language="javascript" type="text/javascript"
src="excanvas.js"></script><![endif]-->'
    m += '     <script language="javascript" type="text/javascript"
src="http://192.168.7.2:8002/jquery.min.js"></script>'
    m += '     <script language="javascript" type="text/javascript"
src="http://192.168.7.2:8002/jquery.jqplot.min.js"></script>'
    m += '     <link rel="stylesheet" type="text/css"
href="http://192.168.7.2:8002/jquery.jqplot.css" />'
    m += '  </head>'
    m += '     <body>'
    m += '     <h1>IV curve or solar cell under test</h1>'

    m += '     <script id="source" language="javascript" type="text/javascript">'
    m += '         $.jqplot("chartdiv",  [['

    ############################


    m += coordinates()



    ############################
    m+= ']]);'
    m += '     </script>'
    m+= '<p> Y axis: current (mA) X axis: voltage (V) </p>'


    m += '  </body>'
    m += ' </html>'
```

```
    m += ' <html>'
    return m

#This code works with jqplot library placed in folder named dist
#test_I2C2.py

import sys
sys.path.append('./Library/Basic_Library')
from mrbbio import *

import smbus
import time
import math
bus = smbus.SMBus(3)

############################
global msg
msg = ''

global t
t = .001

############################

DAC = 0x11 # address for address pin :  0x11 for floating, 0x12 for vcc and 0x10 for
gnd, 0x13 for global
ADC = 0x68
############################
def current_read():
    current= analogRead("P9.33")
    c=0;
    try:
        c=int(current[:4])*1.8/4095
    except:
        pass
    if (c==0):
        try:
            c=int(current[:3])*1.8/4095
        except:
            pass
    if (c==0):
        try:
            c=int(current[:2])*1.8/4095
        except:
           pass
    if (c==0):
        try:
            c=int(current[:1])*1.8/4095
        except:
            pass
    return c

def voltage_read():
    voltage= analogRead("P9.35")
    v=0;
    try:
        v=int(voltage[:4])*1.8/4095
    except:
        pass
```

```
    if (v==0):
        try:
            v=int(voltage[:3])*1.8/4095
        except:
            pass
    if (v==0):
        try:
            v=int(voltage[:2])*1.8/4095
        except:
            pass
    if (v==0):
        try:
            v=int(voltage[:1])*1.8/4095
        except:
            pass
    return v
 ##########################

 ##########################




 ##########################
def setup():
    pass


def loop():
    global t
    global msg
    v_buff = 7
    c_buff = 0
    msg = ''
    bus.write_word_data(0x11,0x2f,0)
    time.sleep(t)
    for i in range(0x00,0xff):

        bus.write_word_data(DAC,0x32,i)
        time.sleep(t)
        c=0
        v=0
        for j in range(1,5):

            c +=current_read()
            v += voltage_read()
            time.sleep(t)
        c = c/5
        v = v/5
        print "current:",c#, current
        print "voltage:",v#, voltage
        print "---------------------"

        #print "c_buff",c_buff #commented during testing on demo day
        #print "v_buff",v_buff #commented during testing on demo day
        print".........1............."
        if c > c_buff and v < v_buff:
        #if 1:
        #if v < v_buff:
```

```
    #if c > c_buff:
    #if  v < v_buff -.01 and c > c_buff -.005:
        msg += '['
        v1= v*2.17
        msg += str(v1)
        msg += ','
        c1 = c / 181
        #c1 = c1/50
        #c1= c1*1000
        c1 = c1 * 1000
        msg += str(c1)
        msg += ']'
        msg += ','
        v_buff = v
        c_buff = c
bus.write_word_data(0x11,0x2f,0)
time.sleep(t)
for i in range(0x00,0xff):

    bus.write_word_data(DAC,0x33,i)
    time.sleep(t)
    c=0
    v=0
    for j in range(1,5):

        c +=current_read()
        v += voltage_read()
        time.sleep(t)
    c = c/5
    v = v/5
    print "current:",c#, current
    print "voltage:",v#, voltage
    print "----------------------"

    #print "c_buff",c_buff
    #print "v_buff",v_buff
    print".......2..............."
    if c > c_buff and v < v_buff:
    #if 1:
    #if v < v_buff:
    #if c > c_buff:
    #if  v < v_buff -.01 and c > c_buff -.005:
        msg += '['
        v1= v*2.17
        msg += str(v1)
        msg += ','
        c1 = c / 181
        #c1 = c1/50
        #c1= c1*1000
        c1 = c1 * 1000
        msg += str(c1)
        msg += ']'
        msg += ','
        v_buff = v
        c_buff = c
bus.write_word_data(0x11,0x2f,0)
time.sleep(t)
for i in range(0x00,0xff):
```

```
    bus.write_word_data(DAC,0x30,i)
    time.sleep(t)
    c=0
    v=0
    for j in range(1,5):

        c +=current_read()
        v += voltage_read()
        time.sleep(t)
    c = c/5
    v = v/5
    print "current:",c#, current
    print "voltage:",v#, voltage
    print "----------------------"

    #print "c_buff",c_buff
    #print "v_buff",v_buff
    print"........3............."
    if c > c_buff and v < v_buff:
    #if 1:
    #if v < v_buff:
    #if c > c_buff:
    #if  v < v_buff -.01 and c > c_buff -.005:
        msg += '['
        v1= v*2.17
        msg += str(v1)
        msg += ','
        c1 = c / 181
        #c1 = c1/50
        #c1= c1*1000
        c1 = c1 * 1000
        msg += str(c1)
        msg += ']'
        msg += ','
        v_buff = v
        c_buff = c

bus.write_word_data(0x11,0x2f,0)
time.sleep(t)
for i in range(0x00,0xff):

    bus.write_word_data(DAC,0x31,i)
    time.sleep(t)
    c=0
    v=0
    for j in range(1,5):

        c +=current_read()
        v += voltage_read()
        time.sleep(t)
    c = c/5
    v = v/5
    print "current:",c#, current
    print "voltage:",v#, voltage
    print "----------------------"

    #print "c_buff",c_buff
    #print "v_buff",v_buff
    print".........4............."
```

```
        if c > c_buff and v < v_buff:
        #if 1:
        #if v < v_buff:
        #if c > c_buff:
        #if  v < v_buff -.01 and c > c_buff -.005:
            msg += '['
            v1= v*2.17# voltage dividor output = .45 * Vin
            msg += str(v1)
            msg += ','
            c1 = c / 181 #gain = 181 of voltage amplification across sense resistor
            #c1 = c1/50 # for 50 mOhm resistance
            #c1= c1*1000# for 50 mOhm resistance
            c1 = c1 * 1000#for one ohm resistance
            msg += str(c1)
            msg += ']'
            msg += ','
            v_buff = v
            c_buff = c
    bus.write_word_data(0x11,0x2f,0)
def coordinates():
    global msg
    run (setup, loop, 1)
    m = msg
    return m[0:len(m)-1]
    bus.write_word_data(0x11,0x2f,0)
 ##########################


bus.write_word_data(0x11,0x2f,0)
#print coordinates()
```

Appendix 6: Example code for making MAX6953 Word 1

```
from PIL import Image
import os

im = Image.open("./uploads/circle.bmp")
s=im.size
#print s
scale = 1
#im2.save("./uploads/output.gif", quality = 10)
#im.convert('RGB')
im.thumbnail((10,10),Image.ANTIALIAS)
#im.save('./uploads/out.png', 'PNG')

data11= 0
data12= 0
data13= 0
data14= 0
data15= 0

thresh = (252, 252, 252) # can be changed to see changed effect (0,0,0) mean pure black
and (255, 255, 255) means pure white

#print im.getcolors() #just to print no. of pixel colors appeared, no direct use
pix = im.load() #load variable pix with image data


for i in range(7):
    print pix[0,i]
    if pix[0,i]<thresh:
        data11 += 1<<i
        #print pix[0,i]

print data11

for i in range(7):
    print pix[1,i]
    if pix[1,i]<thresh:
        data12 += 1<<i
        #print pix[0,i]


print data12

for i in range(7):
    print pix[2,i]
    if pix[2,i]<thresh:
        data13 += 1<<i
        #print pix[0,i]

print data13



for i in range(7):
    print pix[3,i]
    if pix[3,i]<thresh:
        data14 += 1<<i
        #print pix[0,i]
```

```
print data14


for i in range(7):
    print pix[4,i]
    if pix[4,i]<thresh:
        data15 += 1<<i
        #print pix[0,i]

print data15
```

Appendix 7: Electrical Characteristics of Input and Output Pins

Figure A7.1 shows pin numbers of power board and corresponding Table A7.1 shows electrical specifications for pins.

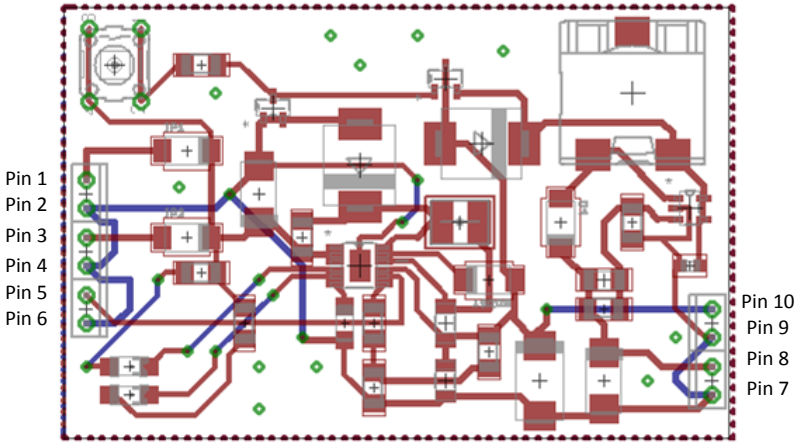


Figure A7.1. Pin Description of Power Board.

Table A7.1. Input/ Output Specification of Power Board

Pin Name	Input / Output	Electrical Name / Spec
Pin 1	Input	
Pin 2	-	Ground
Pin 3	Input	
Pin 4	-	Ground
Pin 5	-	NTC resistor 10k
Pin 6	-	Ground
Pin 7	-	Ground
Pin 8	Output	Vout, 1A @ 5V
Pin 9	-	Ground
Pin 10	Output	Battery Charging, 2A @ 3.78V

Figure A7.2 shows pin numbers of Interface Board and Table A7.2 shows electrical specifications for pins.

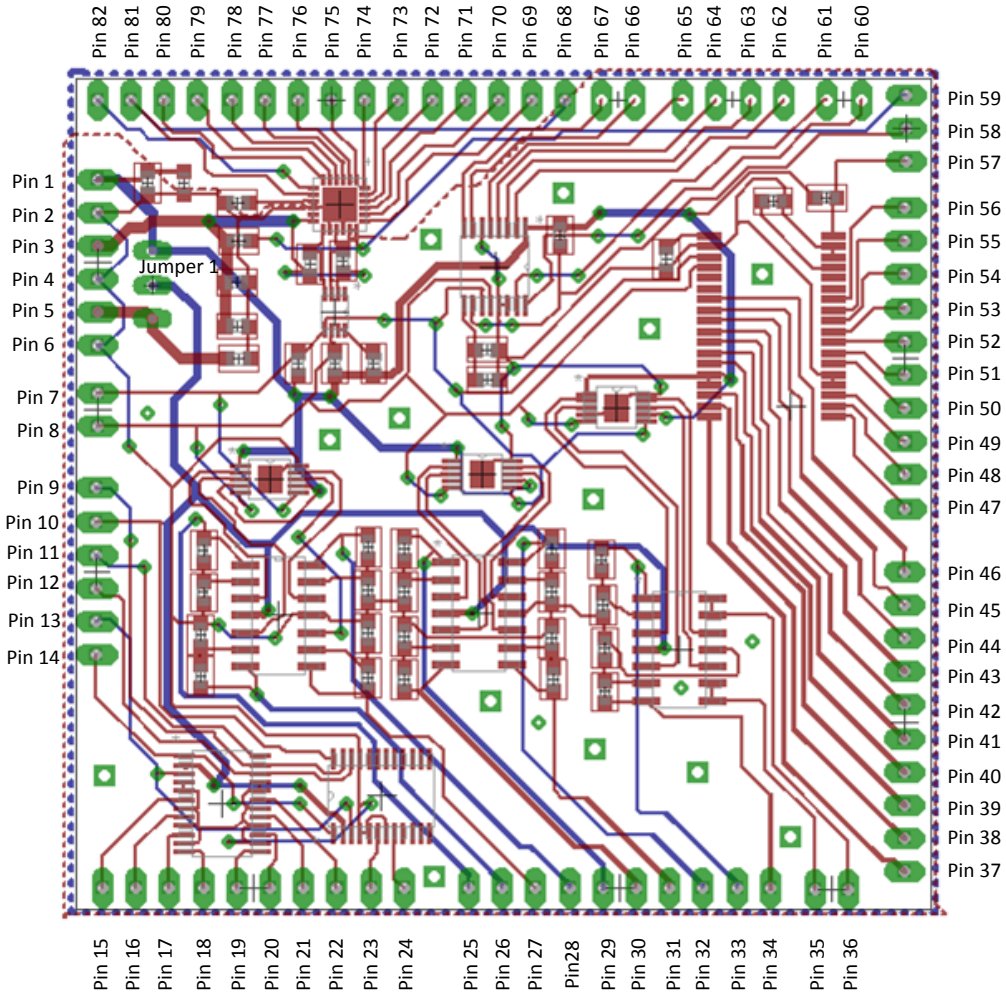


Figure A7.2. Pin numbers of IB.

Table A7.1. Input/ Output Specification of Interface Board

Pin Name	Input/Output	Electrical Name / Spec
Pin 1	Input	5 V = Vin1
Pin 2	-	Ground
Pin 3	Input	3.3 V
Pin 4	-	Ground
Pin 5	Input	~12 V = Vin2 (or15 + 1.7 = 16.7 V for 15 V display matrix)
Pin 6	-	Ground
Pin 7-8	Com Bus	I2C Communication Bus
Pin 9-14	Com Bus	SPI Communication Bus
Pin 15-24	Output	Row Driver, 169 (one channel) to 67(all channels on) mA Sink current, Switch to ground, 9.5 Ω on resistance, ± 15 V tolerable, rail-to-rail operation with input voltage = Vin1 or Vin2
Pin 25-34	Output	40 mA (max) @ at 25 °C, output voltage = (Vin1 or Vin2) - 1.7 V
Pin 35-36	Output	40 mA (max) @ at 25 °C, output voltage = 5 or (5-1.7) V

Pin 37-46	Output	Column Driver, -.3V to (Vin+.3V), 50 mA peak Source Current
Pin 47-56	Output	Row Driver, 500 mA Sink Current
Pin 57	Output	Blink, Open Drain, from MAX6953
Pin 58	Output	Pen Interrupt, Open Drain, from TSC2003
Pin 59	Output	Interrrupt, Open Drain, from AD7147
Pin 60-61	Input	IN1 & IN2, from TSC2003, Auxiliary A/D Converter Input
Pin 62-65	Input	4-wire touch screen connection, Typical On-Resistance (switch) for X+, & Y+ = 5.5 Ω; and for X-, Y- = 7.3 Ω, Drive Current = 50 mA, typical measurement resistance between terminals is few hundred ohms
Pin 66-67	Input	VBAT1 & VBAT2 for battery measurement
Pin 68 & 81	Output	ACSHIELD, shield signal to protect system from stray capacitance
Pin 69-80, 82	Input	CIN0 to CIN12, 13 capacitance sensor inputs, typically 8 pF is allowed range for controller

Figure A7.3 shows pin numbers of I-V curve tracing board and Table A7.3 shows electrical specifications for pins.

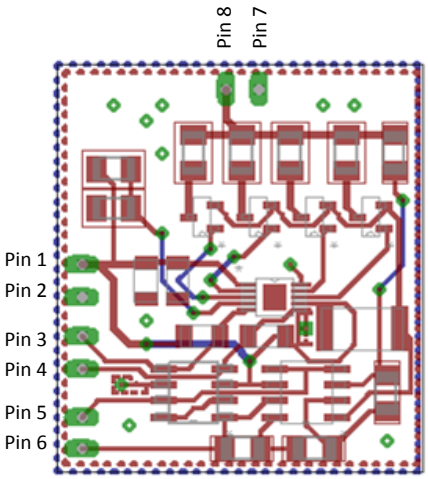


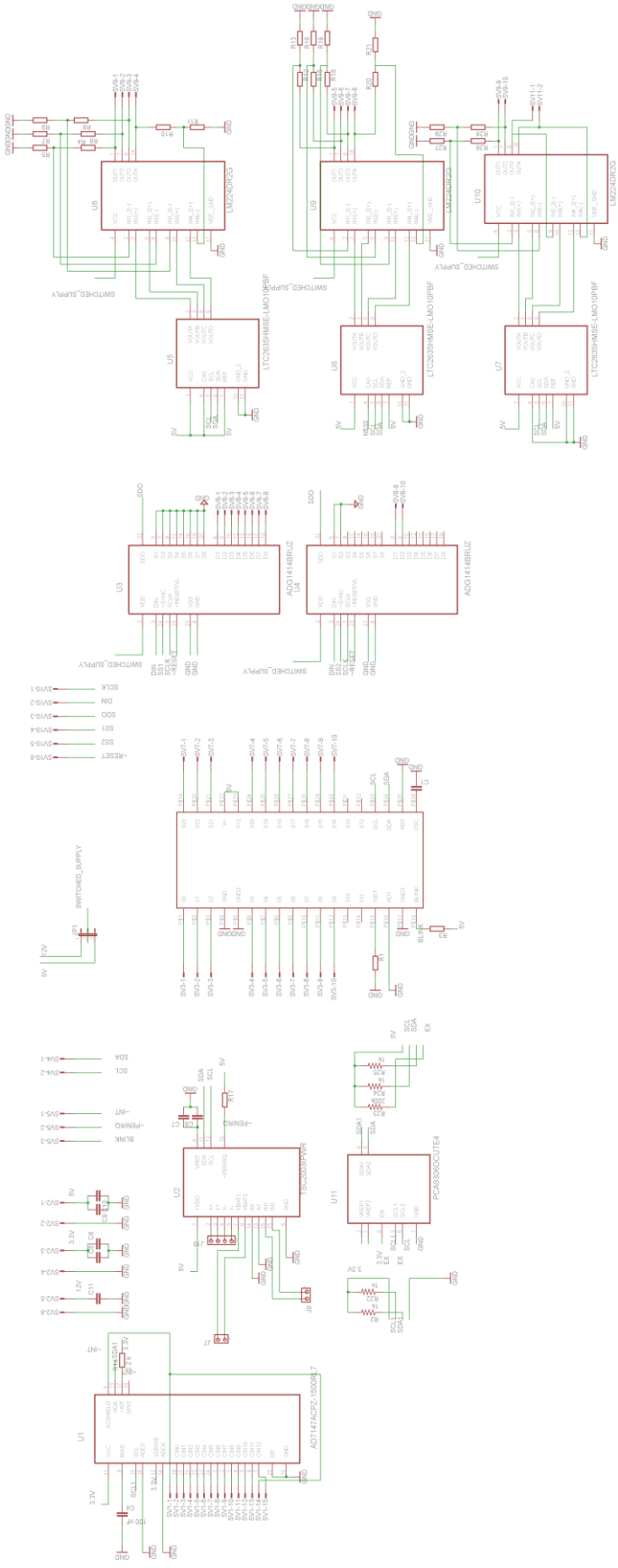
Figure A7.3. Pin numbers of I-V curve tracing board.

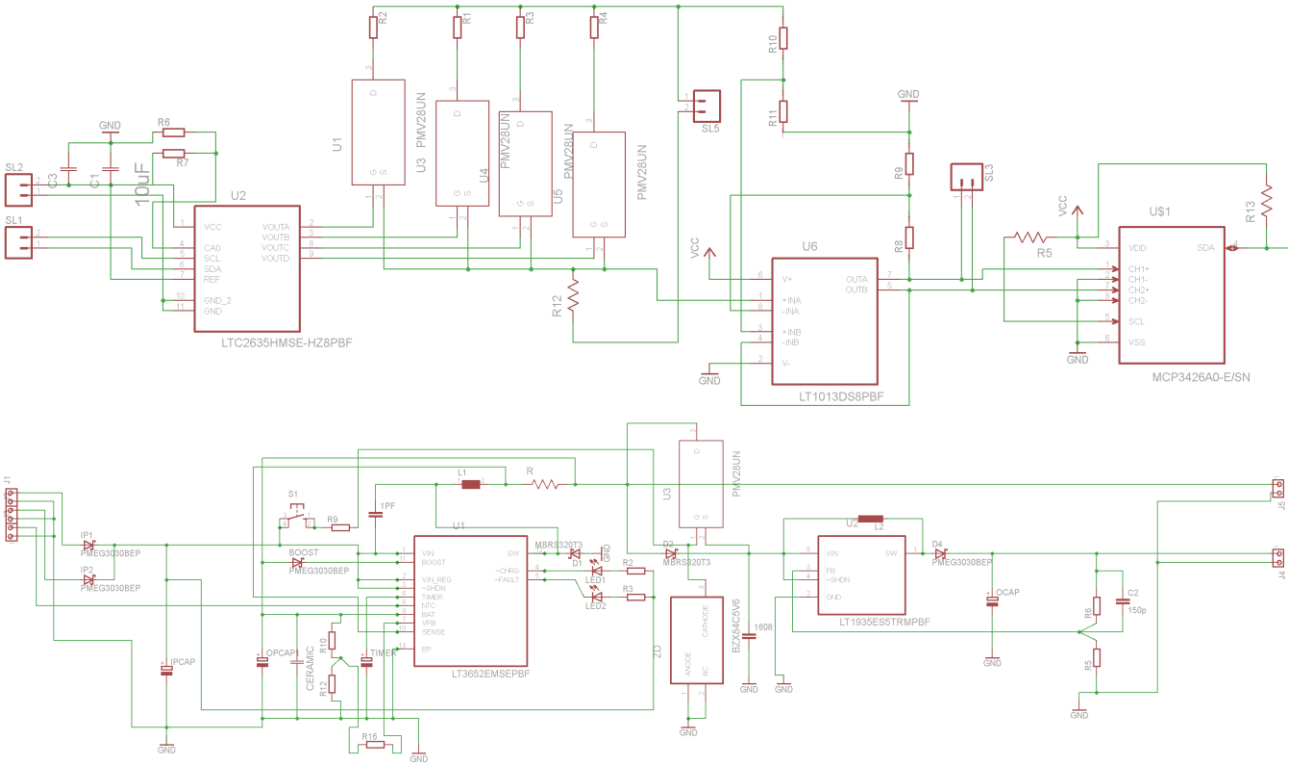
Table A7.3. Input/ Output Specification of I-V curve tracing Board

Pin Name	Input/Output	Electrical Name / Spec
Pin 1	Input	5 V
Pin 2	-	Ground
Pin 3-4	Com Bus	I2C Communication Bus
Pin 5	Output	Voltage Measure Port, 1.8 V
Pin 6	Output	Current Measure Port, 1.8 V
Pin 7	-	Ground

Pin 7	Input	Solar Cell Power In,3.92 V with (voltage division is such that Vout = .45 Vin), 5.5 mA with gain across sense resistance = 181

Appendix 8: Complete Hardware Design




R2
R1
R3
R4
R10
U1
U3 PMV28UN
U4
U5
PMV28UN
SL5
R11
GND
R9
SL3
R6
R7
SL2
SL1
C3
C1
10UF
U2
LTC2635HMSE-HZ8PBF
R12
VCC
U6
LT1013DS8PBF
R8
R5
U$1
MCP3426A0-E/SN
R13

J1
S1
L1
R
U3
PMV28UN
IP1
IP2
PMEG3030BEP
R9
1PF
U1
LT3652EMSEPBF
MBRS320T3
D1
LED1
LED2
R2
R3
IPCAP
OPCAP
CERAMIC
R10
R12
TIMER
R16
ZD
BZX84C5V6
U2
LT1935ES5TRMPBF
D4
OCAP
C2
150p
R6
R5
J5
J4

Appendix 9: Snap Shot of Final Device