LogiLAB : A Smart board for self-Education applications and Academic Laboratories Purposes

By :

Ali can Öçal (2010513046)

Fouad Asil (2012513301)

THESIS

Submitted in partial fulfillment of the requirements for Bachelor's degree in Electric and Electronic Engineering in the Faculty of Agricultural and Engineering of the University of Cukurova University

ADVISOR: Assist. Prof. Dr. Murat AKSOY

Adana – Turkey

2016

ABSTRACT

Designing a smart board for teaching the Logic circuits for all levels of student and non-professionals Using special made circuit connected via microcontroller to LABVIEW Interface , Based on EEE307 Course of Electrical and Electronic Engineering Department in Cukurova University .

Keywords : self-education , Labs panel , LABVIEW , Arduino .

Dedication

To all the pure hearts in this world who are standing with the right and truth . and pay their life or a part of it in that sake.

To all the Oppressed in the world .

To all the martyrs of the great Syrian revolution .

To the next generation who will fix all of this mess .

ACKNOWLEDGMENT

This project would not have been possible without the support of many people. Many thanks to our adviser Assist Prof Dr.Murat AKSOY who offered guidance and support , and to professor DR.Mustafa Gok who enhanced our ideas . Thanks are also to Department of Electric and Electronic Engineering and all the professors who touched us . And for all of our collogues.

And finally, thanks to our families and numerous friends who endured this long process with us, always offering support and love.

Table of Contents

CHAPTER 1: Introduction

Nowadays the world became a small village and the technologies are increasing everyday which give a new horizons for developing our educational methods .

In Engineering, the practical part is very important to understand and take the experience of what you are studying .

1.1 The idea of LogiLab

We have been inspired by our logic circuit course . we found it very interesting. but unfortunately there was no practical part for this course , we wanted so much to try this logic functions and circuits on a real part and take all the values we need . besides that, some necessary electronic abilities are needed to do this experience which could be hard for the amateurs .

So, we decided to design an easy-to-use panel , with interface to help everyone – even children- to know better about the logic circuits if they want .

We can call this a first step , we are looking forward to develop our project to be more useful .

Each sub circuit include pin for each possible input or output with instructions painted over the board .

The result could be seen on the LED’s or on the computer interface using the output pins .

1.2 Used programs

1.2.1 Proteus

PROTEUS 8 ... CAD Connected
Our most integrated product release ever!

The Proteus Design Suite is an Electronic Design Automation (EDA) tool including schematic capture, simulation and PCB Layout modules. It is developed in Yorkshire, England by Labcenter Electronics Ltd with offices in North America and several overseas sales channels.

The software runs on the Windows operating system and is available in English, French, Spanish and Chinese languages.

The Proteus Design Suite is a Windows application for schematic capture, simulation, and PCB layout design. It can be purchased in many configurations, depending on the size of designs being produced and the requirements for microcontroller simulation. All PCB Design products include an autorouter and basic mixed mode SPICE simulation capabilities.

Schematic capture in the Proteus Design Suite is used for both the simulation of designs and as the design phase of a PCB layout project. It is therefore a core component and is included with all product configurations.

The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic. It is then co-simulated along with any analog and digital electronics connected to it. This enables it's used in a broad spectrum of project prototyping in areas such as motor control, temperature control and user interface design. It also finds use in the general hobbyist community and, since no hardware is required, is convenient to use as a training or teaching tool. Support is available for co-simulation of:

- Microchip Technologies PIC10, PIC12, PIC16,PIC18,PIC24,dsPIC33 Microcontrollers.
- Atmel AVR (and Arduino), 8051 and ARM Cortex-M3 Microcontrollers
- NXP 8051, ARM7, ARM Cortex-M0 and ARM Cortex-M3 Microcontrollers.
- Texas Instruments MSP430, PICCOLO DSP and ARM Cortex-M3 Microcontrollers.
- Parallax Basic Stamp, Freescale HC11, 8086 Microcontrollers.

The PCB Layout module is automatically given connectivity information in the form of a netlist from the schematic capture module. It applies this information, together with the user specified design rules and various design automation tools, to assist with error free board design. Design Rule Checking does not include high speed design constraints.[12] PCB's of up to 16 copper layers can be produced with design size limited by product configuration.

The 3D Viewer module allows the board under development to be viewed in 3D together with a semi-transparent height plane that represents the boards enclosure. STEP output can then be used to transfer to mechanical CAD software such as Solidworks or Autodesk for accurate mounting and positioning of the board.

1.2.2 Arduino

1.2.2.1 what is Arduino

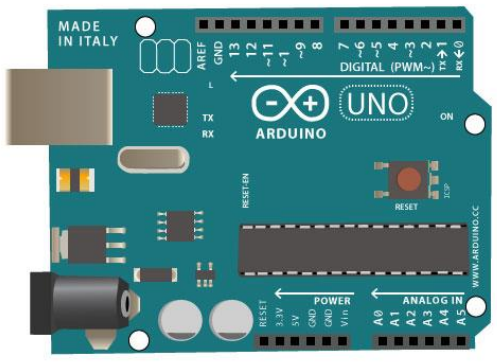
Arduino is a software company, project, and user community that designs and manufactures computer open-source hardware, open-source software, and microcontroller-based kits for building digital devices and interactive objects that can sense and control physical devices.

The project is based on microcontroller board designs, produced by several vendors, using various microcontrollers. These systems provide sets of digital and analog I/O pins that can interface to various expansion boards (termed shields) and other circuits. The boards feature serial communication interfaces, including Universal Serial Bus (USB) on some models, for loading programs from personal computers. For programming the microcontrollers, the Arduino project provides an integrated development environment (IDE) based on a programming language named Processing, which also supports the languages C and C.++

The first Arduino was introduced in 2005, aiming to provide a low cost, easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. Common examples of such devices intended for beginner hobbyists include simple robots, thermostats, and motion detectors.

Arduino boards are available commercially in preassembled form, or as do-it-yourself kits. The hardware design specifications are openly available, allowing the Arduino boards to be produced by anyone. Adafruit Industries estimated in mid-2011 that over 300,000 official Arduinos had been commercially produced, and in 2013 that 700,000 official boards were in users' hands.

1.2.2.2 Arduino UNO



The Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

"Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

1.2.2.3 Software
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.

This software can be used with any Arduino board.

1.2.3 LABVIEW
See chapter 4

1.3 Prospects for development

We consider our project as a first step of a full educational package . one of the most important improve is to connect *LogiLAB* to the moodle of the course using the department web suit , that will give a huge possibility to do a lot .

Developing the easy-to-use interface is important too , we are thinking of the possibility to add the lecture notes to the interface as a theoretical support .

We can increase the output pins by replacing UNO board with any other more improved board .

Finally, we believe that we can make another board for any other electric and electronic field . and we can connect all of this boards using software engineering .

Chapter 2: The Electronic Board

We designed the board to be the work space for all connections . it contains a pre-made sub circuits which based on the EEE307 Course .

2.1 Circuit elements :

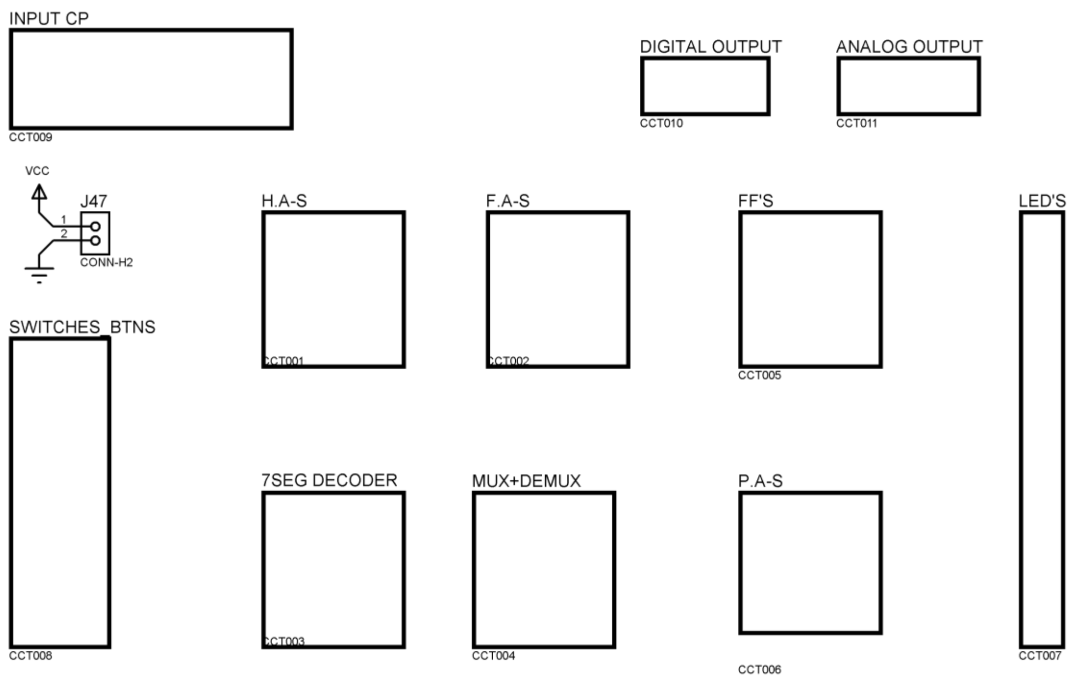
All the logic IC elements are from TTL family so we can use the same voltage level to supply all the circuit .

The datasheets are attached in the appendix .

- Resistors (Various values)
- Capacitors (Various values)
- Pins
- 7-segment display (Common Anode)
- 7404(NOT gate)
- 74185(Full Adder)
- 74386(XOR Gate)
- 7408(AND Gate)
- 7432(OR Gate)
- 7447(Decoder)
- 74111(J-K Flip Flop)
- 74153(MUX)
- 74155(DEMUX)
- 555
- Connecters
- Switches
- Jumpers
- LED's

2.2 Sub Circuits design:

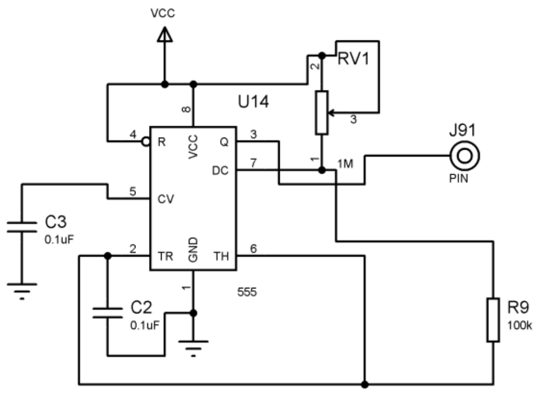
2.2.1 General View :



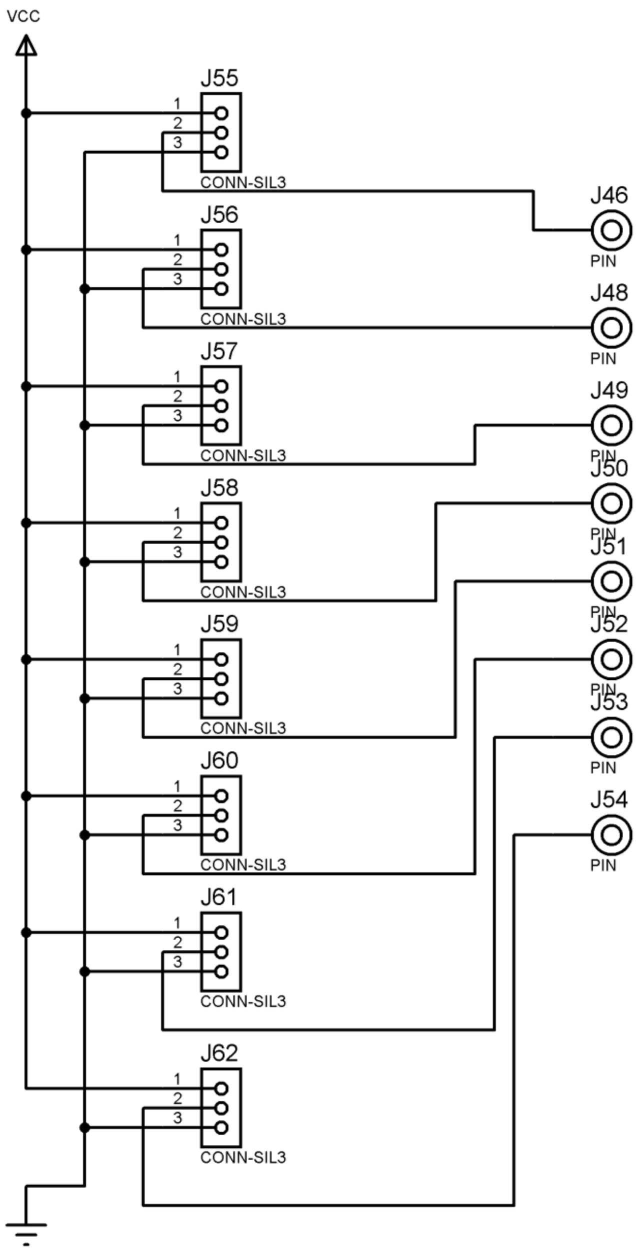
Each block above represent a sub circuit .

2.2.2 Input CP :

We used 555 in Astable mode with Variable resistance to adjust the frequency .

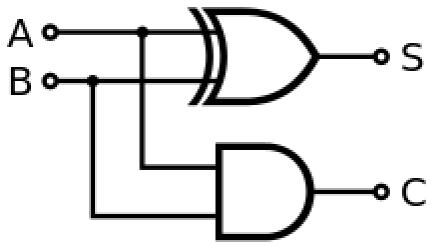


Switching Buttons as a Digital input either 1 or 0 (VCC or GND)



2.2.3 Half Adder – Subtractor

The **half adder** adds two single binary digits *A* and *B*. It has two outputs, sum (*S*) and carry (*C*). The carry signal represents an overflow into the next digit of a multi-digit addition. The value of the sum is 2*C* + *S*. The simplest half-adder design, pictured on the right, incorporates an XOR gate for *S* and an AND gate for *C*. With the addition of an OR gate to combine their carry outputs, two half adders can be combined to make a full adder.



The half adder adds two input bits and generates a carry and sum, which are the two outputs of a half adder. The input variables of a half adder are called the augend and addend bits. The output variables are the sum and carry. The truth table for the half adder is:

Inputs Outputs

A	***B***	***C***	***S***
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

The half subtractor is a combinational circuit which is used to perform subtraction of two bits. It has two inputs, the minuend and subtrahend and two outputs the difference and borrow out . The borrow out signal is set when the subtractor needs to borrow

from the next digit in a multi-digit subtraction. That is,   when   . Since and are bits, if and only if and . An important point worth mentioning is that the half

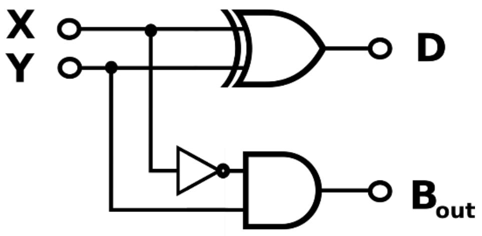
subtractor diagram aside implements and not since on the diagram is given by   .

This is an important distinction to make since subtraction itself is not commutative, but the difference bit s calculated using an XOR gate which is commutative.

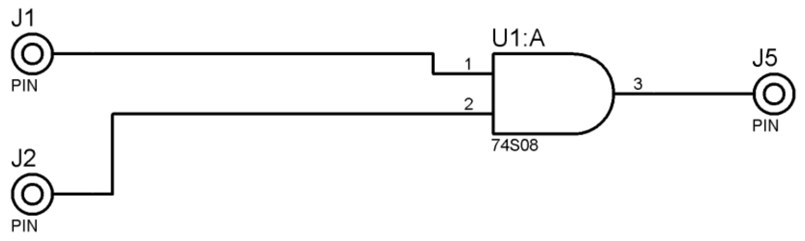
The truth table for the half subtractor is:

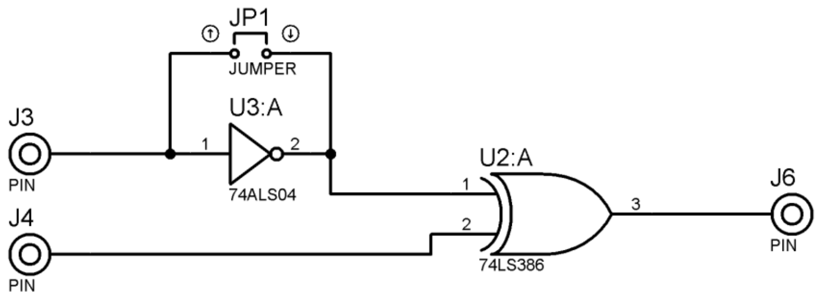
Inputs Outputs

X	Y	D	Bout
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



LogiLAB Circuit is giving the ability to do the both by using jumpers :





2.2.4 Full Adder - Subtractor

A **full adder** adds binary numbers and accounts for values carried in as well as out. A one-bit full adder adds three one-bit numbers, often written as *A*, *B*, and C_{in}; *A* and *B* are the operands, and C_{in} is a bit carried in from the previous less-significant stage.[2] The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. bit binary numbers. The circuit produces a two-bit output, output carry and sum

typically represented by the signals C_{out} and *S*, where .

A full adder can be implemented in many different ways such as with a custom transistor-level circuit or composed of other gates. One example implementation is

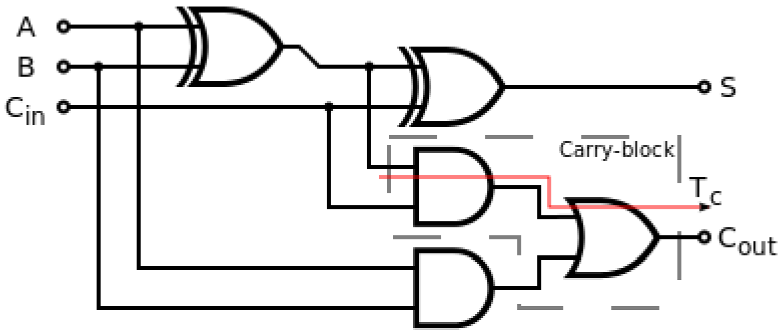
with and .

In this implementation, the final OR gate before the carry-out output may be replaced by an XOR gate without altering the resulting logic. Using only two types of gates is convenient if the circuit is being implemented using simple IC chips which contain only one gate type per chip.

A full adder can be constructed from two half adders by connecting *A* and *B* to the input of one half adder, connecting the sum from that to an input to the second adder, connecting C_i to the other input and OR the two carry outputs. The critical path of a

full adder runs through both XOR-gates and ends at the sum bit . Assumed that an XOR-gate takes 3 delays to complete, the delay imposed by the critical path of a full adder is equal to

The carry-block subcomponent consists of 2 gates and therefore has a delay of



The full subtractor is a combinational circuit which is used to perform subtraction of

three input bits: the minuend , subtrahend , and borrow in . The full

subtractor generates two output bits: the difference and borrow out . is

set when the previous digit borrowed from . Thus, is also subtracted from

as well as the subtrahend . Or in symbols: . Like the half subtractor, the full subtractor generates a borrow out when it needs to borrow from the next digit.

Since we are subtracting by and , a borrow out needs to be generated

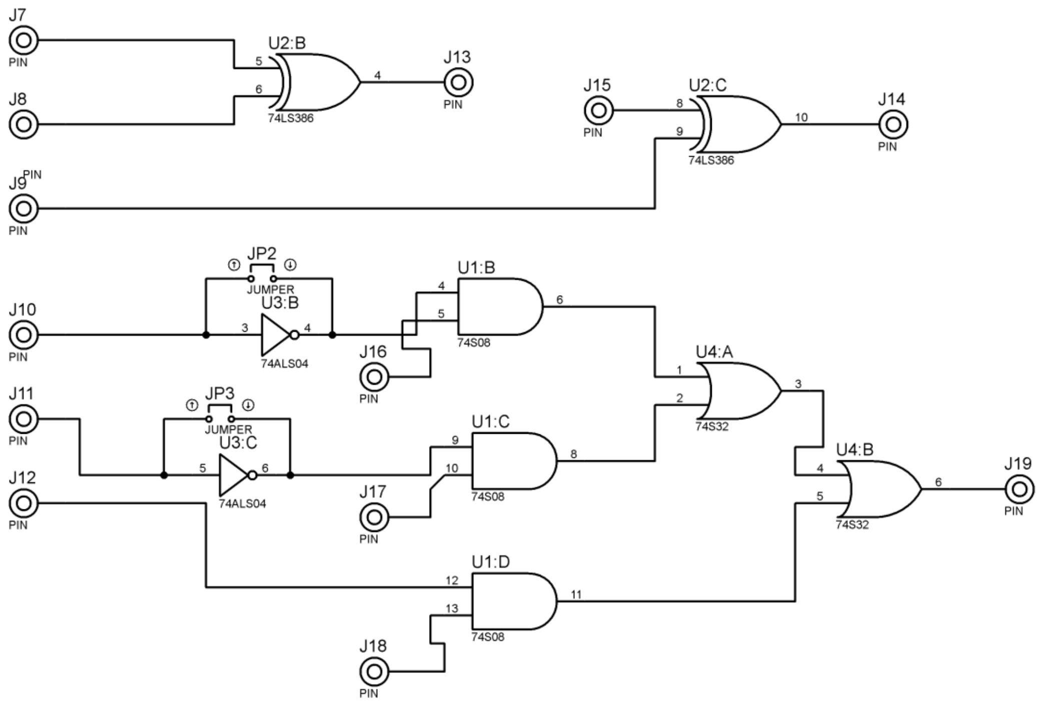
when . When a borrow out is generated, 2 is added in the current digit. (This is similar to the subtraction algorithm in decimal. Instead of adding 2, we add 10 when

we borrow.) Therefore, .

The truth table for the full subtractor is:

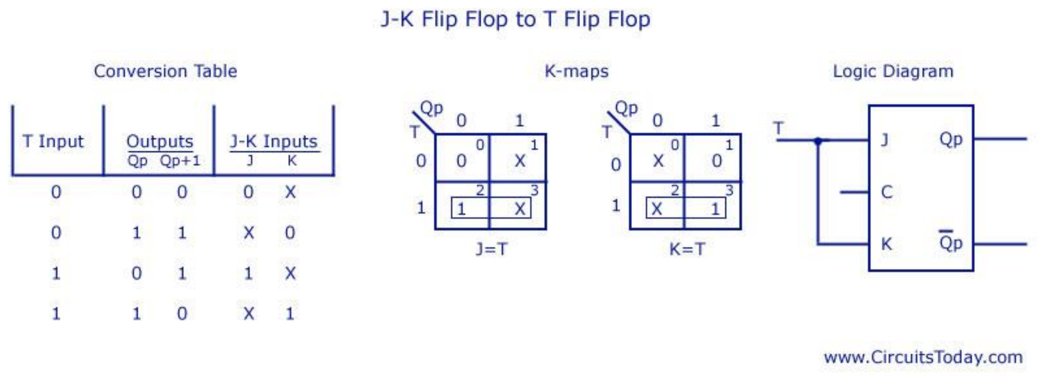
Inputs			**Outputs**	
X	***Y***	***B*in**	***D***	***B*out**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

LogiLAB Circuit :

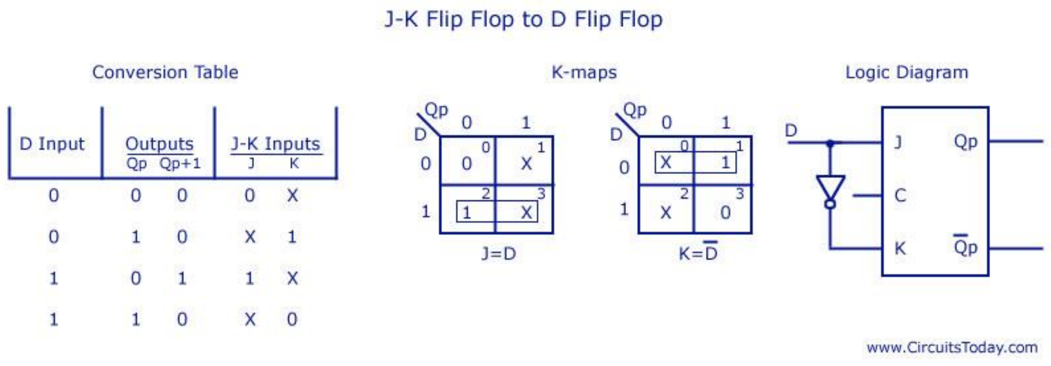


2.2.5 Flip Flops :

We used J-K Flip Flop because it’s easy to convert it to D or T FF's .

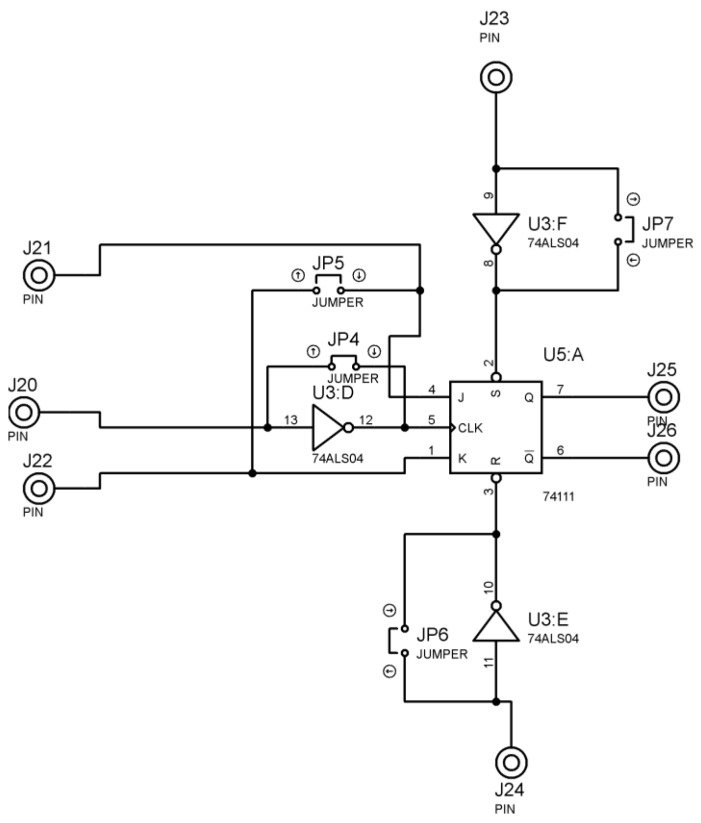


T Input	Outputs Qp	Outputs Qp+1	J-K Inputs J	J-K Inputs K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1



D Input	Outputs Qp	Outputs Qp+1	J-K Inputs J	J-K Inputs K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

LogiLAB Circuit :

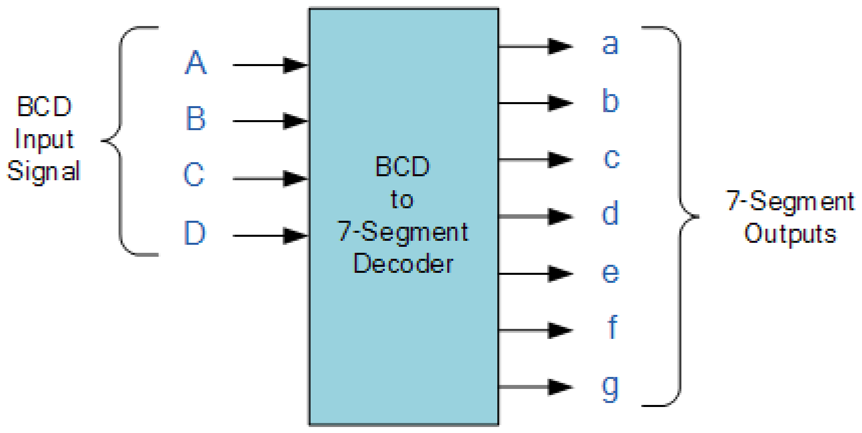


2.2.6 Decoder

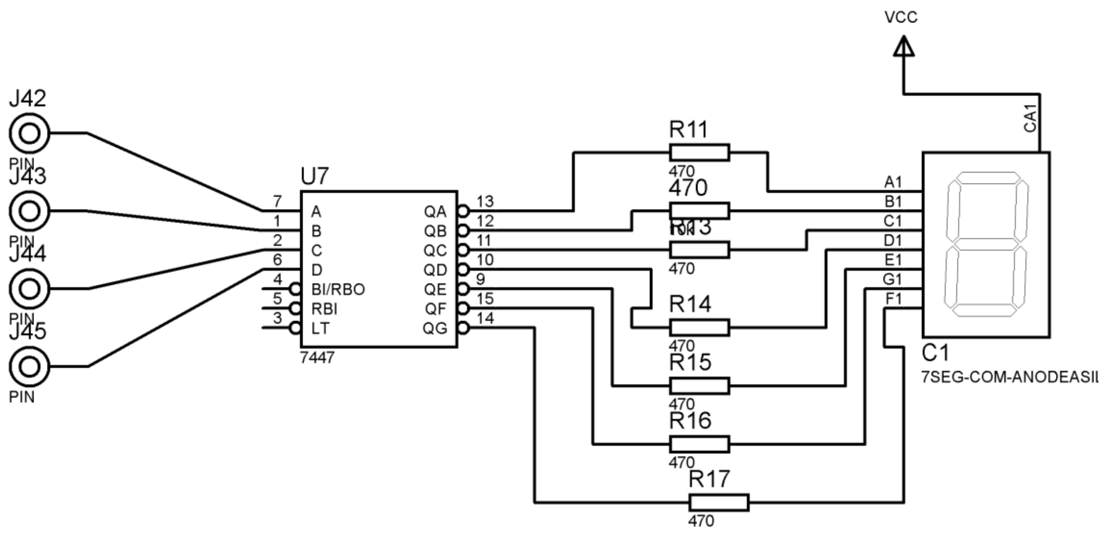
A decoder is a circuit that changes a code into a set of signals. It is called a decoder because it does the reverse of encoding, but we will begin our study of encoders and decoders with decoders because they are simpler to design.

BCD to 7-Segment Display Decoders

A binary coded decimal (BCD) to 7-segment display decoder such as the TTL 74LS47 or 74LS48, have 4 BCD inputs and 7 output lines, one for each LED segment. This allows a smaller 4-bit binary number (half a byte) to be used to display all the denary numbers from 0 to 9 and by adding two displays together, a full range of numbers from 00 to 99 can be displayed with just a single byte of 8 data bits.



LogiLAB Decoder circuit :



2.2.7 Multiplexer and Demultiplexer

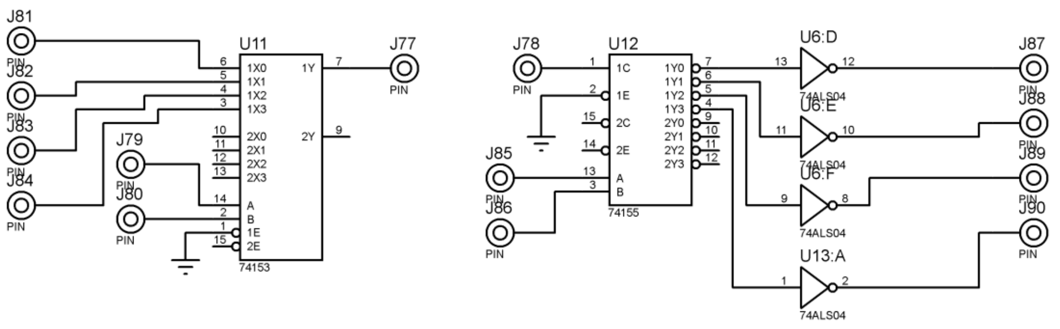
In electronics, a multiplexer (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line .A multiplexer of 2n inputs has n select lines, which are used to select which input line to send to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth . A multiplexer is also called a data selector. Multiplexers can also be used to implement Boolean functions of multiple variables.

An electronic multiplexer makes it possible for several signals to share one device or resource, for example one A/D converter or one communication line, instead of having one device per input signal.

Conversely, a demultiplexer (or demux) is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input. A multiplexer is often used with a complementary demultiplexer on the receiving endز

An electronic multiplexer can be considered as a multiple-input, single-output switch, and a demultiplexer as a single-input, multiple-output switch . The schematic symbol for a multiplexer is an isosceles trapezoid with the longer parallel side containing the input pins and the short parallel side containing the output pin. The schematic on the right shows a 2-to-1 multiplexer on the left and an equivalent switch on the right. The sel wire connects the desired input to the output.

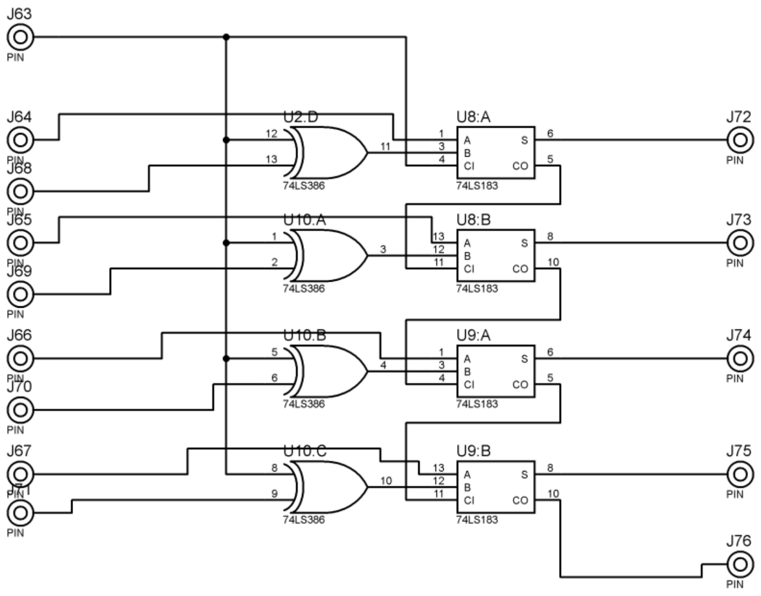
In LogiLAB There is a typical 2 select bits MUX and DEMUX . they could use individually or togather by connecting the both selector inputs .



2.2.8 Parallel Adder / Subtractor

The operations of both addition and subtraction can be performed by a one common binary adder. Such binary circuit can be designed by adding an Ex-OR gate with each full adder as shown in below figure. The figure below shows the 4 bit parallel binary adder/subtractor which has two 4 bit inputs as A3A2A1A0 and B3B2B1B0.

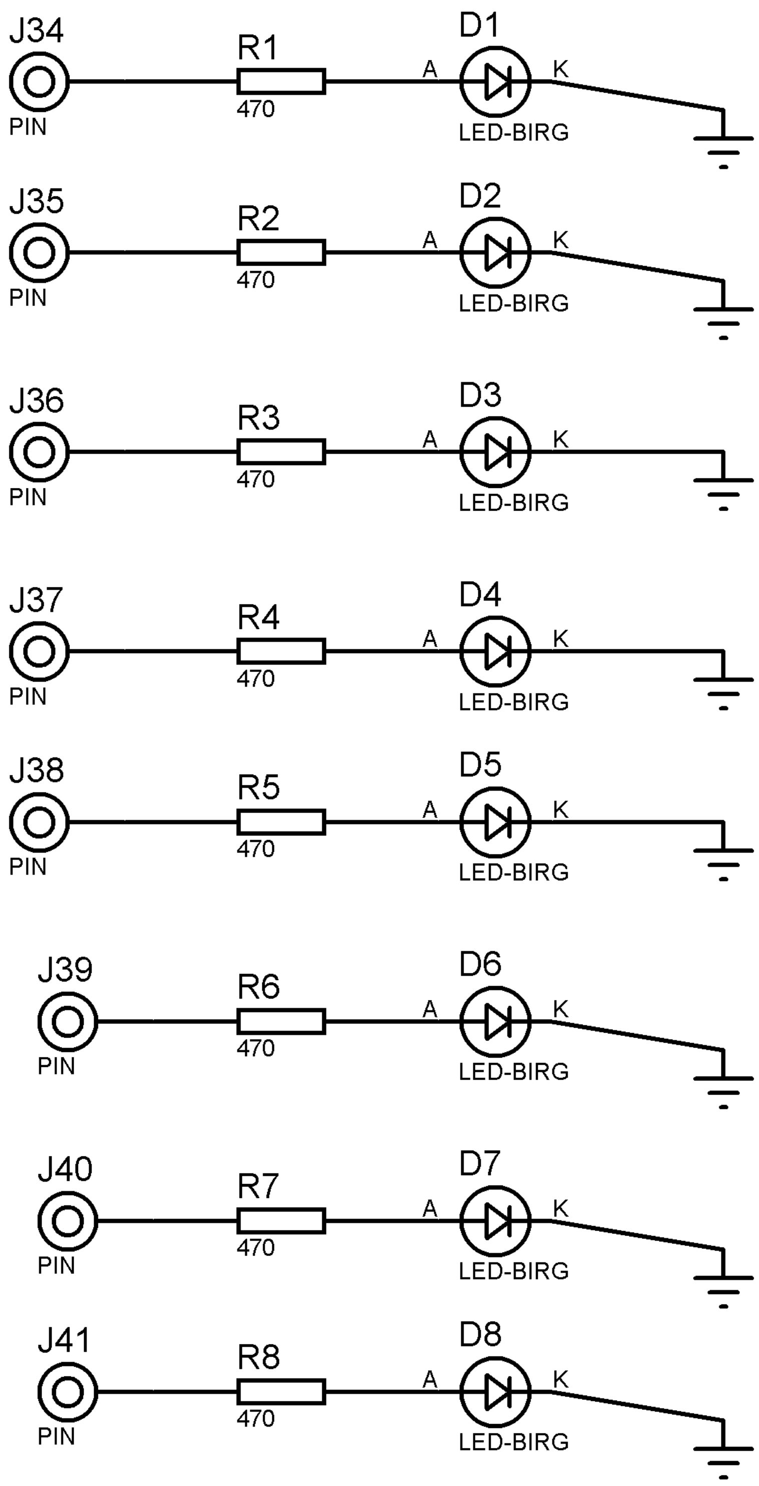
The mode input control line M is connected with carry input of the least significant bit of the full adder. This control line decides the type of operation, whether addition or subtraction.



PIN J63 is the control line .

2.2.9 Outputs

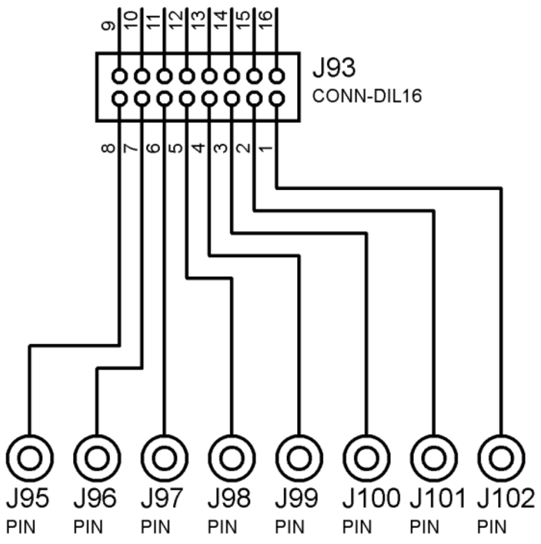
2.2.9.1 LEDS outputs for immediate Digital values :

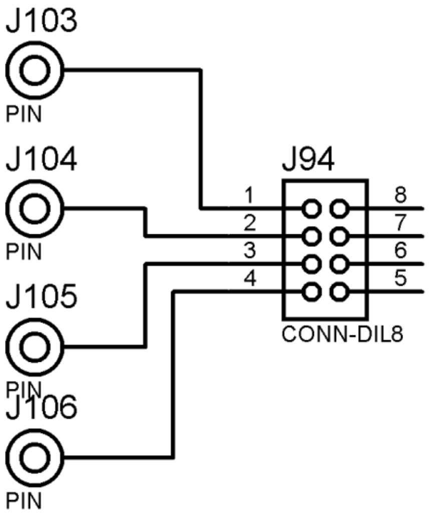


2.2.9.2 Arduino (Microcontroller) Outputs

We used Arduino UNO as a default microcontroller but we gave the ability to use any other Arduino board by the settings in the interface .

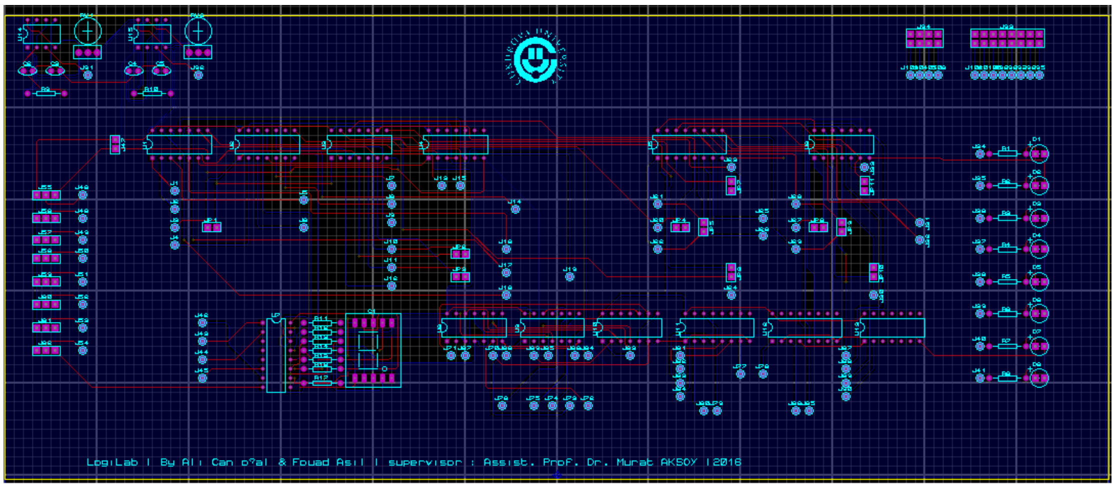
The UNO board contains 4 Analog pins and more than 12 digital pins , we used 4 Analog and 8 digital pins as the board output :



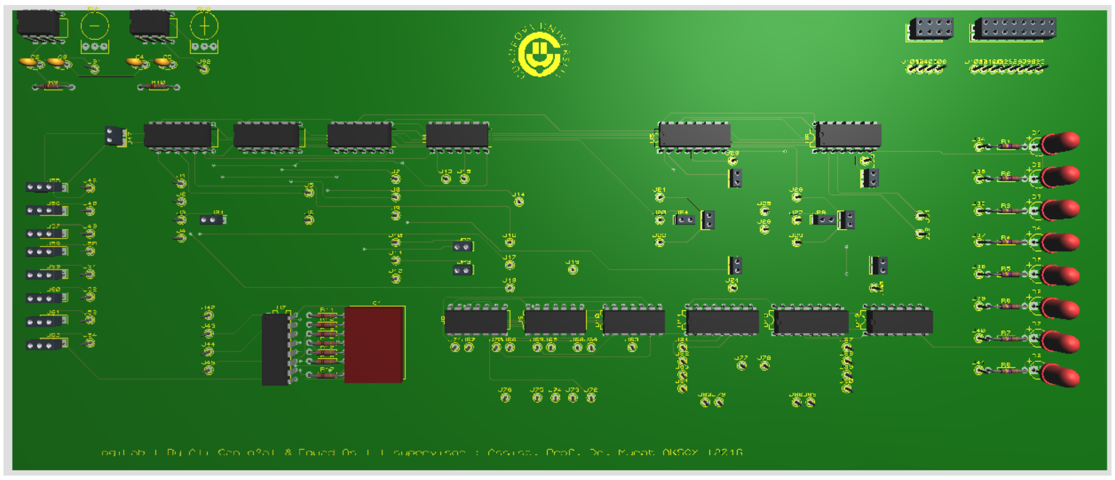


2.3 PCB

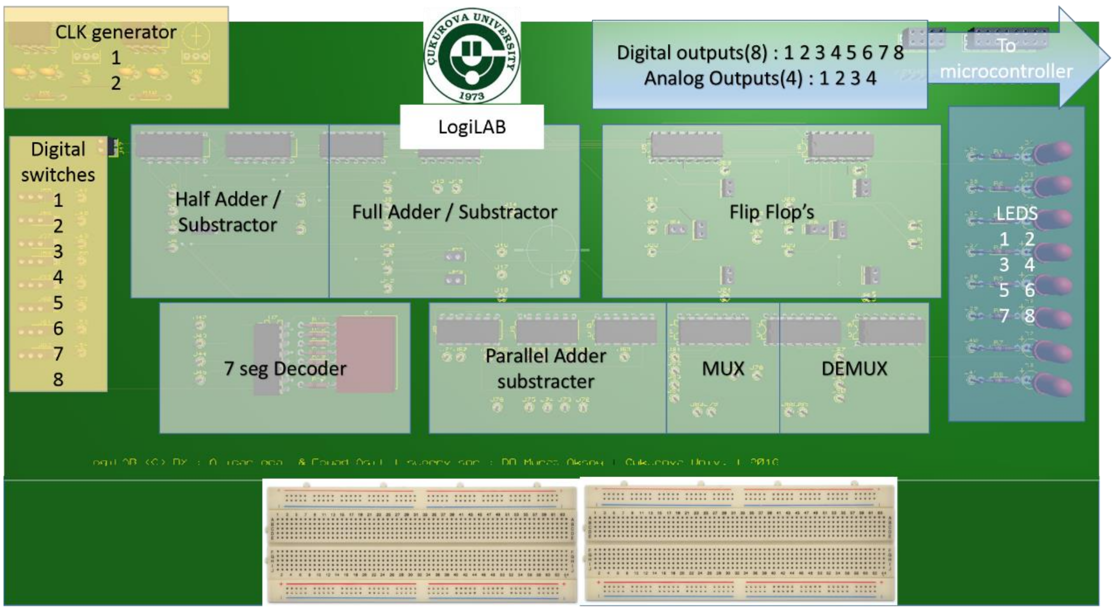
Layout design made by Proteus ARES , we designed the printed circuit according to the general design and we left places for instructions on the board .


LogiLab | By Ali Can o?al & Fouad Asil | supervisor : Assist. Prof. Dr. Murat AKSOY |2016

A 3D visualization without the instructions



This figure explains the relation between hardware and design



A bread boards are going to be added , so the student can use the elements of LogiLAB and can use the computers interface .

Chapter 3: Microcontroller (Arduino Board)

Since we used LabView we used **VI Package Manager (VIPM)** to connect the board with the labview interface .

Arduino code to send and receive Values form and toward the computer is base_lifa :

The main code :

```
**  LVFA_Firmware - Provides Basic Arduino Sketch For Interfacing With LabVIEW

 **  Written By:    Sam Kristoff - National Instruments

 **  Written On:    November 2010

 **  Last Updated:  Dec 2011 - Kevin Fort - National Instruments

 **  This File May Be Modified And Re-Distributed Freely. Original File Content

 **  Written By Sam Kristoff And Available At www.ni.com/arduino.

** Includes.

// Standard includes.  These should always be included.

#include <Wire.h>

#include <SPI.h>

#include <Servo.h>

#include "LabVIEWInterface.h"

 setup()

 **  Initialize the Arduino and setup serial communication.

 **

 **  Input:  None

 **  Output: None

*************************************************************************
******/

void setup()
```

```
{
 // Initialize Serial Port With The Default Baud Rate
 syncLV();
 // Place your custom setup code here
}
/*************************************************************************
******
 **  loop()
 **
 **  The main loop.  This loop runs continuously on the Arduino.  It
 **  receives and processes serial commands from LabVIEW.
 **  Input:  None
 **  Output: None

*************************************************************************
******/
void loop()
{
 // Check for commands from LabVIEW and process them.
 checkForCommand();
 // Place your custom loop code here (this may slow down communication with LabVIEW)
 if(acqMode==1)
 {
   sampleContinously();
 }
}
```

Required Files :

LabView Interface.h

```
/*************************************************************************
******

 **  LVFA_Firmware - Provides Functions For Interfacing With The Arduino Uno

**  Written By:    Sam Kristoff - National Instruments

 **  Written On:    November 2010

 **  Last Updated:  Dec 2011 - Kevin Fort - National Instruments

  **  This File May Be Modified And Re-Distributed Freely. Original File Content

 **  Written By Sam Kristoff And Available At www.ni.com/arduino.

**  Define Constants

**

**  Define directives providing meaningful names for constant values.

*************************************************************************
******/



#define FIRMWARE_MAJOR 02

#define FIRMWARE_MINOR 00

#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)

#define DEFAULTBAUDRATE 9600    // Defines The Default Serial Baud Rate (This must match
the baud rate specifid in LabVIEW)

#else

#define DEFAULTBAUDRATE 115200

#endif

#define MODE_DEFAULT 0           // Defines Arduino Modes (Currently Not Used)

#define COMMANDLENGTH 15         // Defines The Number Of Bytes In A Single LabVIEW
Command (This must match the packet size specifid in LabVIEW)

#define STEPPER_SUPPORT 1        // Defines Whether The Stepper Library Is Included -
Comment This Line To Exclude Stepper Support
```

```
// Declare Variables

unsigned char currentCommand[COMMANDLENGTH];    // The Current Command For The
Arduino To Process

//Globals for continuous aquisition

unsigned char acqMode;

unsigned char contAcqPin;

float contAcqSpeed;

float acquisitionPeriod;

float iterationsFlt;

int iterations;

float delayTime;

**  syncLV

**  Synchronizes with LabVIEW and sends info about the board and firmware
(Unimplemented)

**  Input:  None

**  Output: None

*****************************************************************************
******/

void syncLV();

**  setMode

**  Sets the mode of the Arduino (Reserved For Future Use)

**  Input:  Int - Mode

**  Output: None

void setMode(int mode);

**  checkForCommand

**
```

** Checks for new commands from LabVIEW and processes them if any exists.

** Input: None

** Output: 1 - Command received and processed

** 0 - No new command

int checkForCommand(void);

** processCommand

** Processes a given command

** Input: command of COMMANDLENGTH bytes

** Output: 1 - Command received and processed

** 0 - No new command

void processCommand(unsigned char command[]);

** writeDigitalPort

** Write values to DIO pins 0 - 13. Pins must first be configured as outputs.

** Input: Command containing digital port data

** Output: None

void writeDigitalPort(unsigned char command[]);

** analogReadPort

** Reads all 6 analog input ports, builds 8 byte packet, send via RS232.

** Input: None

** Output: None

void analogReadPort();

** sevenSegment_Config

** Configure digital I/O pins to use for seven segment display. Pins are stored in sevenSegmentPins array.

** Input: Pins to use for seven segment LED [A, B, C, D, E, F, G, DP]

** Output: None

void sevenSegment_Config(unsigned char command[]);

** sevenSegment_Write

** Write values to sevenSegment display. Must first use sevenSegment_Configure

** Input: Eight values to write to seven segment display

** Output: None

void sevenSegment_Write(unsigned char command[]);

** spi_setClockDivider

** Set the SPI Clock Divisor

** Input: SPI Clock Divider 2, 4, 8, 16, 32, 64, 128

** Output: None

void spi_setClockDivider(unsigned char divider);

** spi_sendReceive

** Sens / Receive SPI Data

** Input: Command Packet

** Output: None (This command sends one serail byte back to LV for each data byte.

void spi_sendReceive(unsigned char command[]);

** checksum_Compute

** Compute Packet Checksum

** Input: Command Packet

** Output: Char Checksum Value

unsigned char checksum_Compute(unsigned char command[]);

** checksum_Test

** Compute Packet Checksum And Test Against Included Checksum

** Input: Command Packet

```
**  Output: 0 If Checksums Are Equal, Else 1

int checksum_Test(unsigned char command[]);

**  AccelStepper_Write

**  Parse command packet and write speed, direction, and number of steps to travel

**  Input:  Command Packet

**  Output: None

void AccelStepper_Write(unsigned char command[]);

**  SampleContinuosly

**  Returns several analog input points at once.

**  Input:  void

**  Output: void

void sampleContinously(void);

**  finiteAcquisition

**  Returns the number of samples specified at the rate specified.

**  Input:  pin to sampe on, speed to sample at, number of samples

**  Output: void

void finiteAcquisition(int analogPin, float acquisitionSpeed, int numberOfSamples );

**  lcd_print

**  Prints Data to the LCD With The Given Base

**  Input:  Command Packet

**  Output: None

void lcd_print(unsigned char command[]);
```

Chapter 4: LabView Interface

4.1 LabVIEW

(short for Laboratory Virtual Instrument Engineering Workbench) is a system-design platform and development environment for a visual programming language from National Instruments.

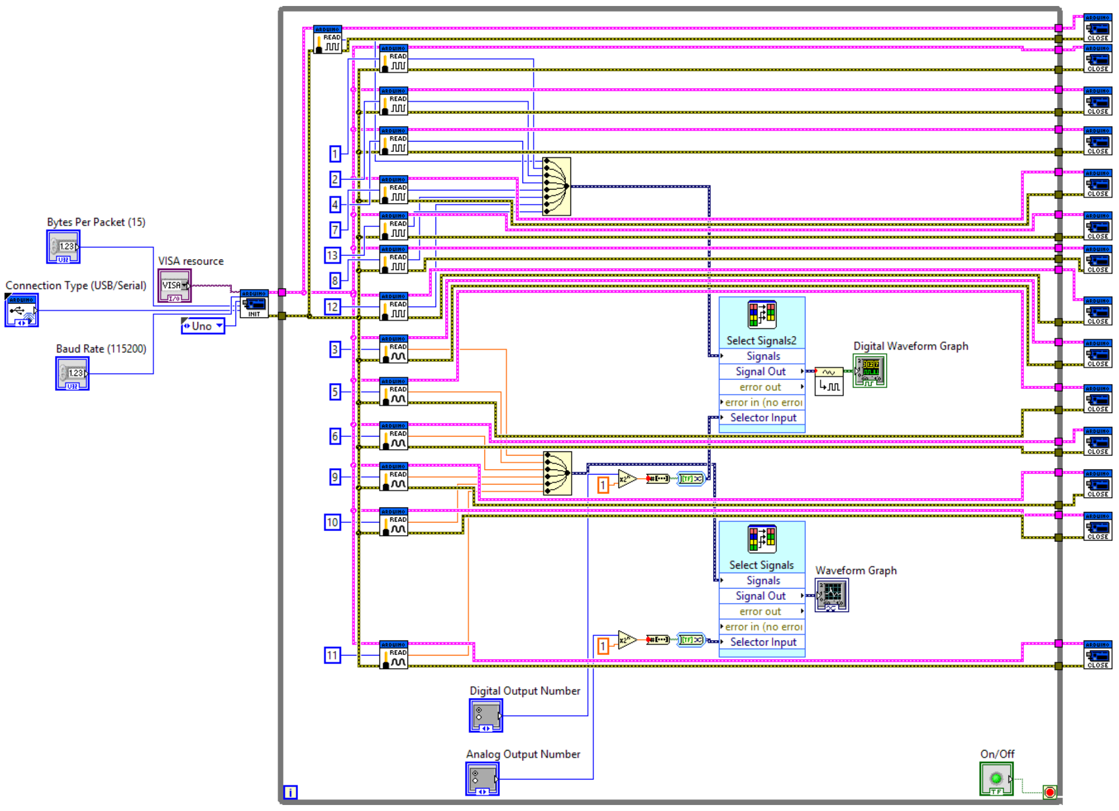
The graphical language is named "G" (not to be confused with G-code). Originally released for the Apple Macintosh in 1986, LabVIEW is commonly used for data acquisition, instrument control, and industrial automation on a variety of platforms including Microsoft Windows, various versions of UNIX, Linux, and OS X. The latest version of LabVIEW is LabVIEW 2015, released in August 2015.

LabVIEW ties the creation of user interfaces (called front panels) into the development cycle. LabVIEW programs/subroutines are called virtual instruments (VIs). Each VI has three components: a block diagram, a front panel and a connector panel. The last is used to represent the VI in the block diagrams of other, calling VIs. The front panel is built using controls and indicators. Controls are inputs – they allow a user to supply information to the VI. Indicators are outputs – they indicate, or display, the results based on the inputs given to the VI. The back panel, which is a block diagram, contains the graphical source code. All of the objects placed on the front panel will appear on the back panel as terminals. The back panel also contains structures and functions which perform operations on controls and supply data to indicators. The structures and functions are found on the Functions palette and can be placed on the back panel. Collectively controls, indicators, structures and functions will be referred to as nodes. Nodes are connected to one another using wires – e.g. two controls and an indicator can be wired to the addition function so that the indicator displays the sum of the two controls. Thus a virtual instrument can either be run as a program, with the front panel serving as a user interface, or, when dropped as a node onto the block diagram, the front panel defines the inputs and outputs for the node through the connector pane. This implies each VI can be easily tested before being embedded as a subroutine into a larger program.

The graphical approach also allows non-programmers to build programs by dragging and dropping virtual representations of lab equipment with which they are already familiar. The LabVIEW programming environment, with the included examples and documentation, makes it simple to create small applications. This is a benefit on one side, but there is also a certain danger of underestimating the expertise needed for high-quality G programming. For complex algorithms or large-scale code, it is important that the programmer possess an extensive knowledge of the special

LabVIEW syntax and the topology of its memory management. The most advanced LabVIEW development systems offer the possibility of building stand-alone applications. Furthermore, it is possible to create distributed applications, which communicate by a client/server scheme, and are therefore easier to implement due to the inherently parallel nature of G.

4.2 LogiLab interface Code :



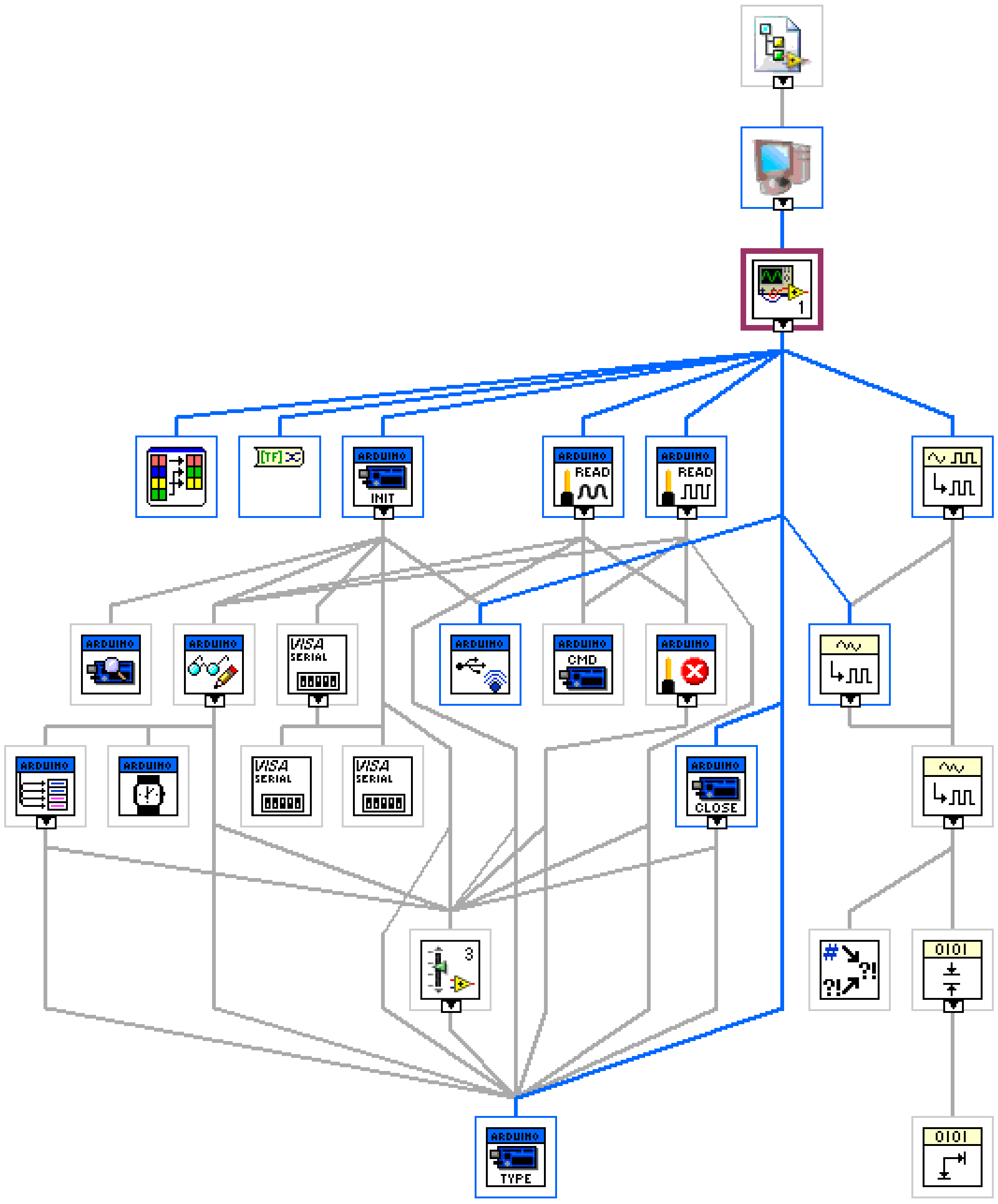
The outer part is for microcontroller settings , inside the "while Loop" which is controlled by stop button we can see the blocks representing Arduino's pins which are connected to the board .

The yellow blocks (one for digital signals and the other is for the analog signals) are for marge the signals together as a first step before select signals .

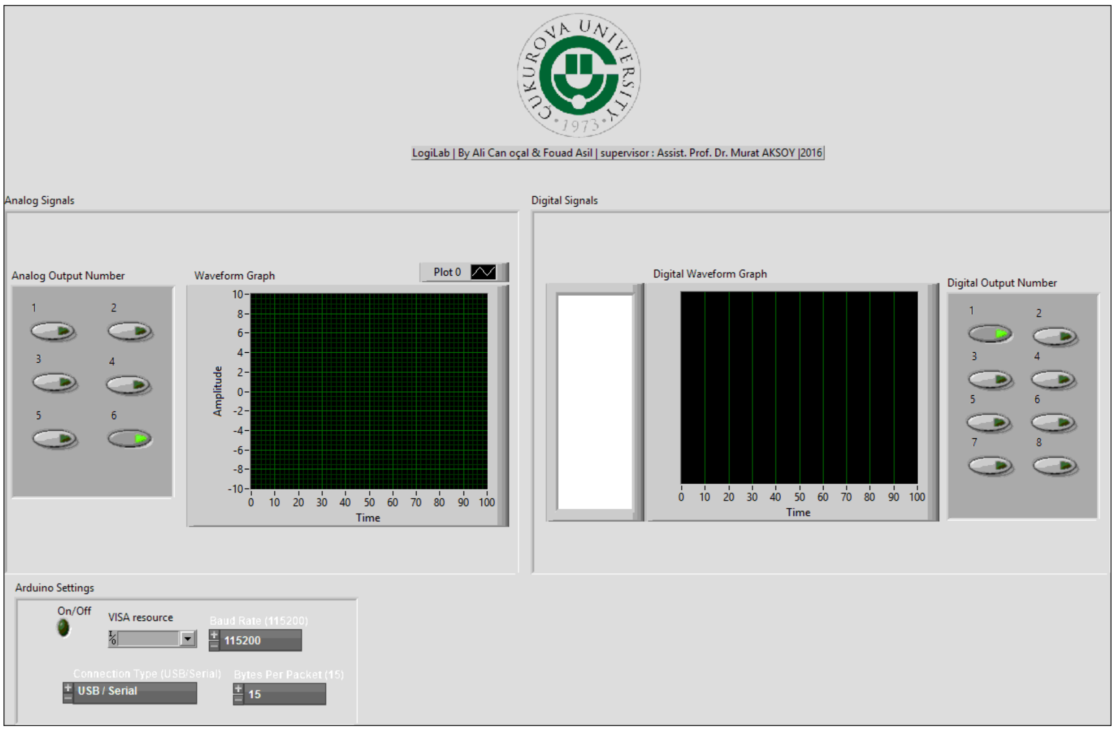
Radio buttons for selecting the channel or the output number are transformed to binary array to use as selector input .

The "select signals" block will work as a multiplexer to choose which output will be plotted in the graphs .

Position in Hierarchy



The interface :

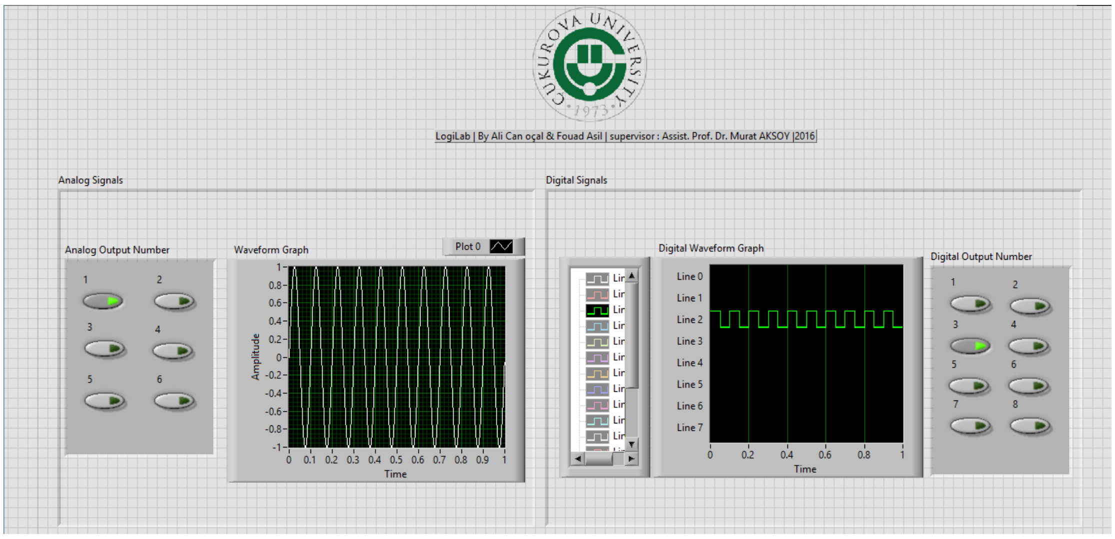


4.3 Interface simulation

Since we didn't have enough time to print the circuit and try it in the reality, we did a simulation for two signals generated from the PC to replace the real signals .

One of them is a digital , the other is an analog .

We connected this signals as an input for the LABVIEW program and we saw the signals on the graphs .



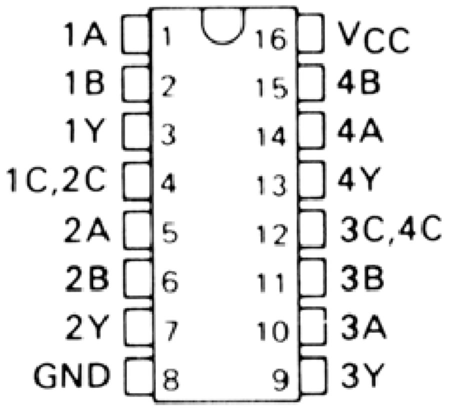
REFERENCES

AKSOY, Murat. EEE325 lecture notes.
GOK, Mustafa . EEE307 lecture notes.

- Neamen, Donald A. Microelectronics : circuit analysis and design
- https://www.arduino.cc/en/main/arduinoBoardUno
- https://en.wikipedia.org/wiki/Arduinoss
- https://en.wikipedia.org/wiki/Proteus_Design_Suite
- http://www.labcenter.com/index.cfm
- www.ni.com

APPENDIX A: Electronic Elements datasheets :

74135 : Exclusive-OR/NOR Gates



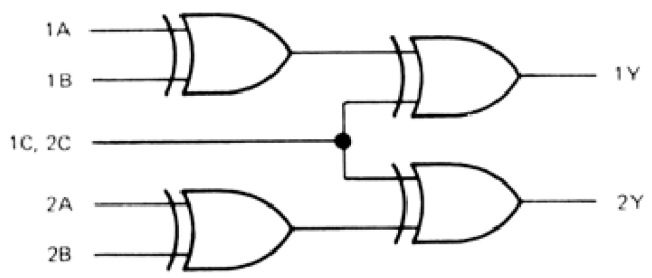
FUNCTION TABLE

INPUTS			OUTPUT
A	B	C	Y
L	L	L	L
L	H	L	H
H	L	L	H
H	H	L	L
L	L	H	H
L	H	H	L
H	L	H	L
H	H	H	H

H = high level, L = low level

- Fully Compatible with Most TTL and TTL MSI Circuits
- Fully Schottky Clamping Reduces Delay Times . . . 8 ns Typical
- Can Operate as Exclusive-OR Gate (C Input Low) or as Exclusive-NOR Gate (C Input High)

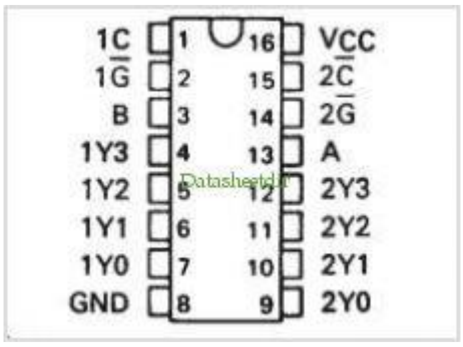
logic diagram (one half)



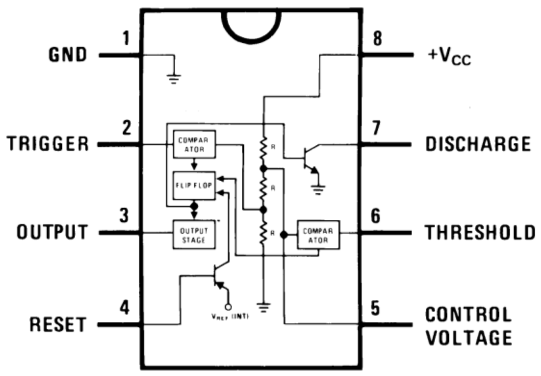
positive logic

$$Y = A \oplus B \oplus C = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

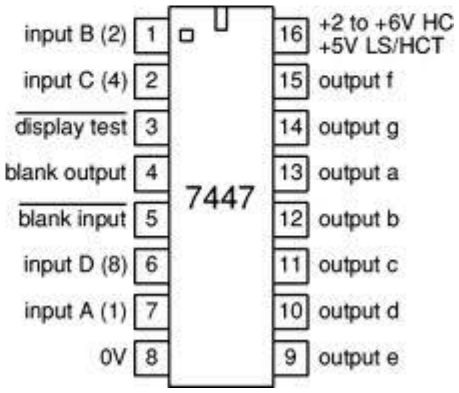
74155:



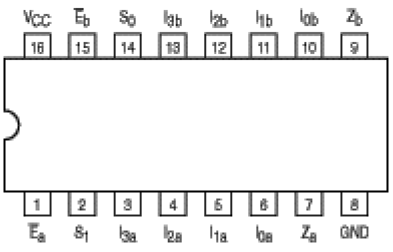
555 :



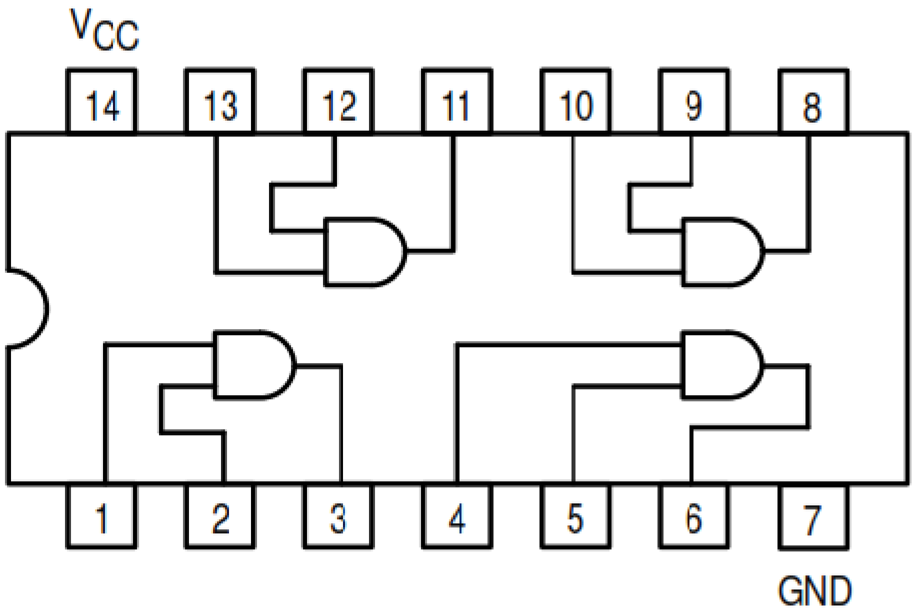
7447:



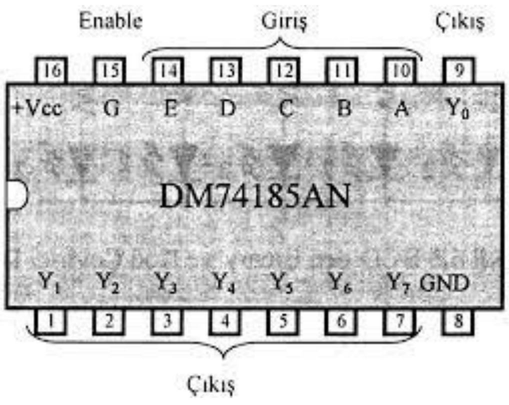
74153



7408

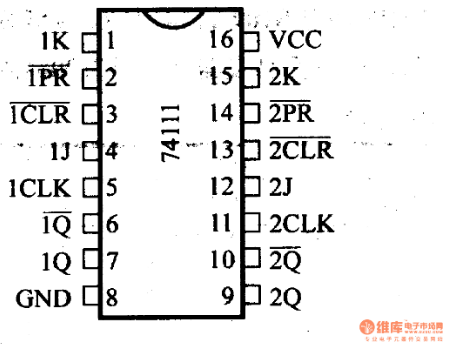


74185 :

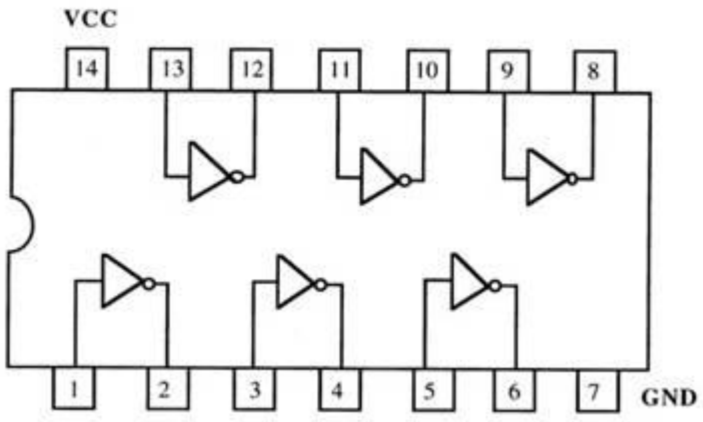


74111、74F111 双 J-K 触发器（带数据锁定）

功能表见 74110。

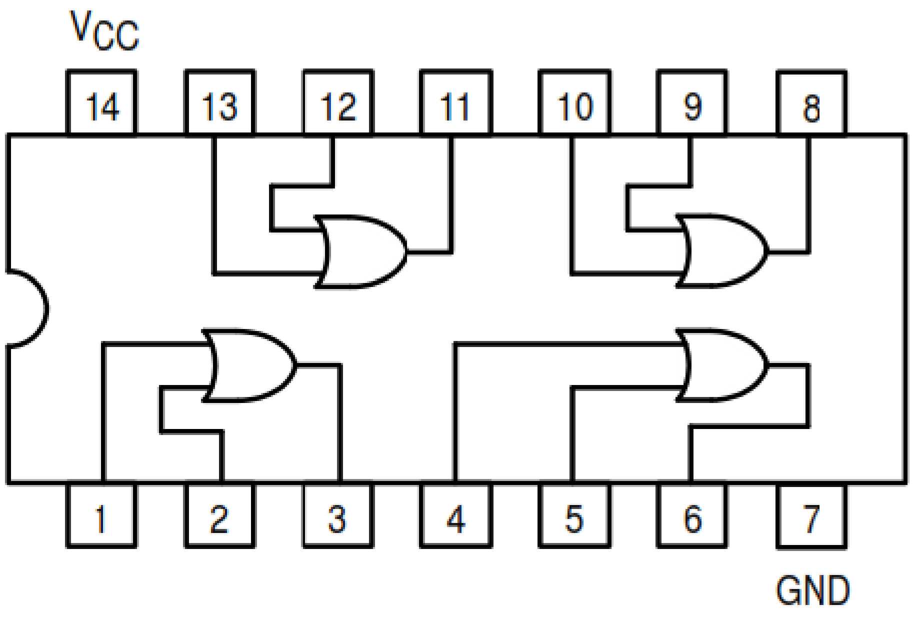


7404:



74LS04

7432


VCC
14
13
12
11
10
9
8
1
2
3
4
5
6
7
GND