FUNCTION GENERATOR USING ARDUINO DUE

Project report submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

Arunabho Mukherjee(121012)

Anirudh Singh (121015)

Praveen Bagriya(121022)

Aman Aggarwal(121086)

UNDER THE GUIDANCE OF

Dr. RAJEEV KUMAR

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

May , 2016

CONTENTS

DECLARATION

We hereby declare that the work reported in the B-Tech thesis entitled **"Function Generator Using Arduino Due"** submitted at **Jaypee University of Information Technology, Waknaghat India,** is an authentic record of our work carried out under the supervision of **Dr. Rajeev Kumar**. We have not submitted this work elsewhere for any other degree or diploma.

Arunabho Mukherjee (121012)

Anirudh Singh (121015)

Praveen Bagriya(121022)

Aman Aggarwal(121086)

Department of Electronics and Communication Engineering

Jaypee University of Information Technology, Waknaghat , India

Date: 24th May,2016

CERTIFICATE

This is to certify that the work reported in the B-Tech. thesis entitled "**Function Generator Using Arduino Due**", submitted by **Arunabho Mukherjee (121012) , Anirudh Singh (121015) , Praveen Bagriya (121022) and Aman Aggarwal (121086)** at **Jaypee University of Information Technology, Waknaghat , India,** is a bonafide record of their original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

(Signature of Supervisor)

Dr. Rajeev Kumar

Dept. of ECE, JUIT

24th May,2016.

ACKNOWLEDGEMENT

We would like to thank Prof. (Dr.) Sunil Bhooshan, Head of Dept. Electronics and Communication, for providing us the opportunity of working on a project.

We are highly indebted to Dr. Rajeev Kumar for motivating and enlightening us for our project work. We thank you for being a constant support throughout, without whose valuable guidance and insights, this project would not be a complete one. We offer you our sincere gratitude to you for instructing and directing us through thick and thin.

We would also like to acknowledge Mr. Mohan Sharma for helping us in the labs. Thank you for being there and helping us in and out.

LIST OF ACROYNMS

UART	Universal Asynchronous Receiver/Transmitter
PWM	Pulse Width Modulation
USB-OTG	Universal Serial Bus-On The Go
DAC	Digital to Analog Converter
TWI	Two Wire Interface
SPI	Serial Peripheral Interface
JTAG	Joint Test Action Group
AWG	Arbitrary Waveform Generators
DDS	Direct Digital Synthesis
ARM	Advanced RISC Machine
SDA	Serial Data
SCL	Serial Clock
AREF	Analog Reference
IOREF	Input Output Reference
AVR	Advanced Virtual RISC
SRAM	Static Random Access Memory
RISC	Reduced Instruction Set Computer

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In a time where everything is getting simplified, using an Arduino simplifies the amount of hardware and software development you need to do in order to get a system running. Our current study includes development of a function generator using arduino due and interfacing keypad and lcd to generate waveforms of different frequencies and amplitudes. In the future this would be used to test debug complex circuits also made in Arduino.

CHAPTER 1
LITERATURE SURVEY

1.1 Introduction

1.1.1 Function Generator

A **function generator** is usually a piece of electronic test equipment or software used to generate different types of electrical waveforms over a wide range of frequencies. Some of the most common waveforms produced by the function generator are the sine, square, triangular and sawtooth shapes.

These waveforms can be either repetitive or single-shot (which requires an internal or external trigger source). Integrated circuits used to generate waveforms may also be described as function generator ICs.

Although function generators cover both audio and RF frequencies, they are usually not suitable for applications that need low distortion or stable frequency signals. When those traits are required, other signal generators would be more appropriate.

Some function generators can be phase-locked to an external signal source (which may be a frequency reference) or another function generator.

Function generators are used in the development, test and repair of electronic equipment. For example, they may be used as a signal source to test amplifiers or to introduce an error signal into a control loop.

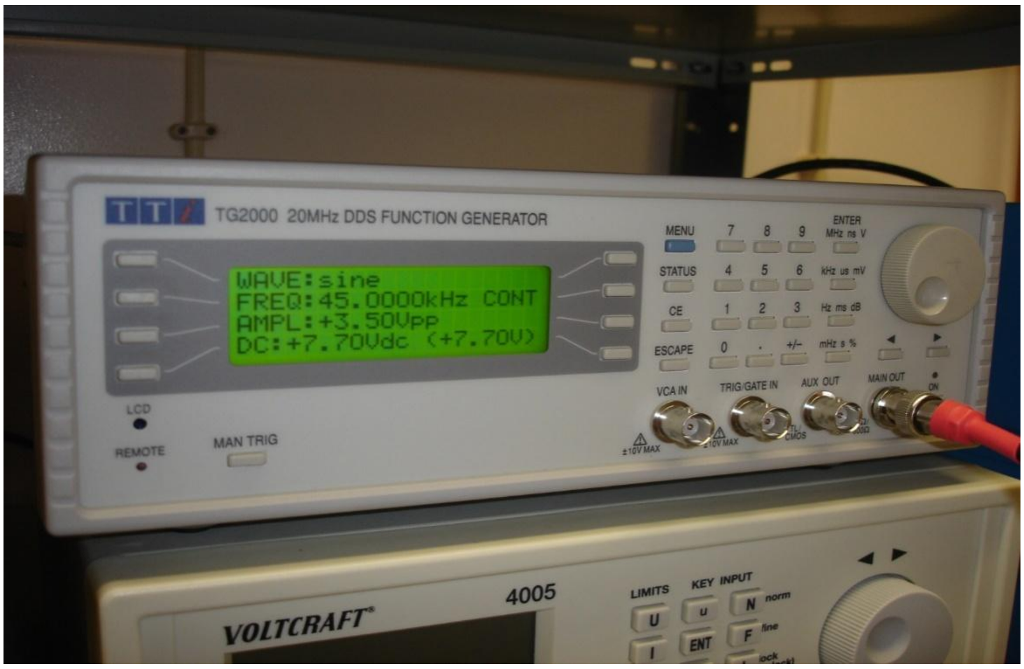


Figure 1.1 : Function Generator

Working

Simple function generators usually generate triangular waveform whose frequency can be controlled smoothly as well as in steps. This triangular wave is used as the basis for all of its other outputs. The triangular wave is generated by repeatedly charging and discharging a capacitor from a constant current source. This produces a linearly ascending or descending voltage ramp. As the output voltage reaches upper and lower limits, the charging and discharging is reversed using a comparator, producing the linear triangle wave. By varying the current and the size of the capacitor, different frequencies may be obtained. Sawtooth waves can be produced by charging the capacitor slowly, using a current, but using a diode over the current source to discharge quickly - the polarity of the diode changes the polarity of the resulting sawtooth, i.e. slow rise and fast fall, or fast rise and slow fall.

A 50% duty cycle square wave is easily obtained by noting whether the capacitor is being charged or discharged, which is reflected in the current switching comparator output. Other duty cycles (theoretically from 0% to 100%) can be obtained by using a comparator and the sawtooth or triangle signal. Most function generators also contain a non-linear diode shaping circuit that can convert the triangle wave into a reasonably accurate sine wave by rounding off the corners of the triangle wave in a process similar to clipping in audio systems.

A typical function generator can provide frequencies up to 20 MHz. RF generators for higher frequencies are not function generators in the strict sense since they typically produce pure or modulated sine signals only.

Function generators, like most signal generators, may also contain an attenuator, various means of modulating the output waveform, and often the ability to automatically and repetitively "sweep" the frequency of the output waveform (by means of a voltage controlled oscillator) between two operator-determined limits. This capability makes it very easy to evaluate the frequency response of a given electronic circuit.

Some function generators can also generate white or pink noise.

More advanced function generators are called arbitrary waveform generators (AWG). They use direct digital synthesis (DDS) techniques to generate any waveform that can be described by a table of amplitudes.

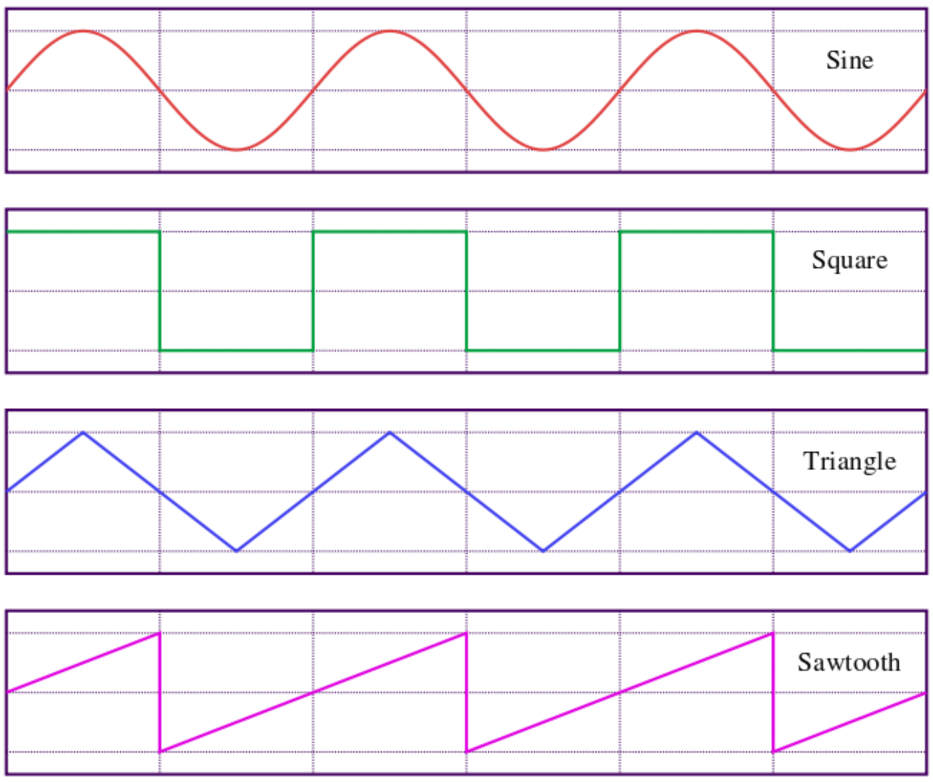


Figure 1.2 : Waveforms

A function generator requires extensive hardware, which is pre-programmed by the manufacturing company and is not flexible or portable. An arduino based function generator is a multi-purpose application of arduino due and is easy and fast to program, adaptable to changes and only requires an arduino due board.

1.1.2 The Arduino Due:

The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button.

Unlike most Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Applying voltages higher than 3.3V to any I/O pin could damage the board.

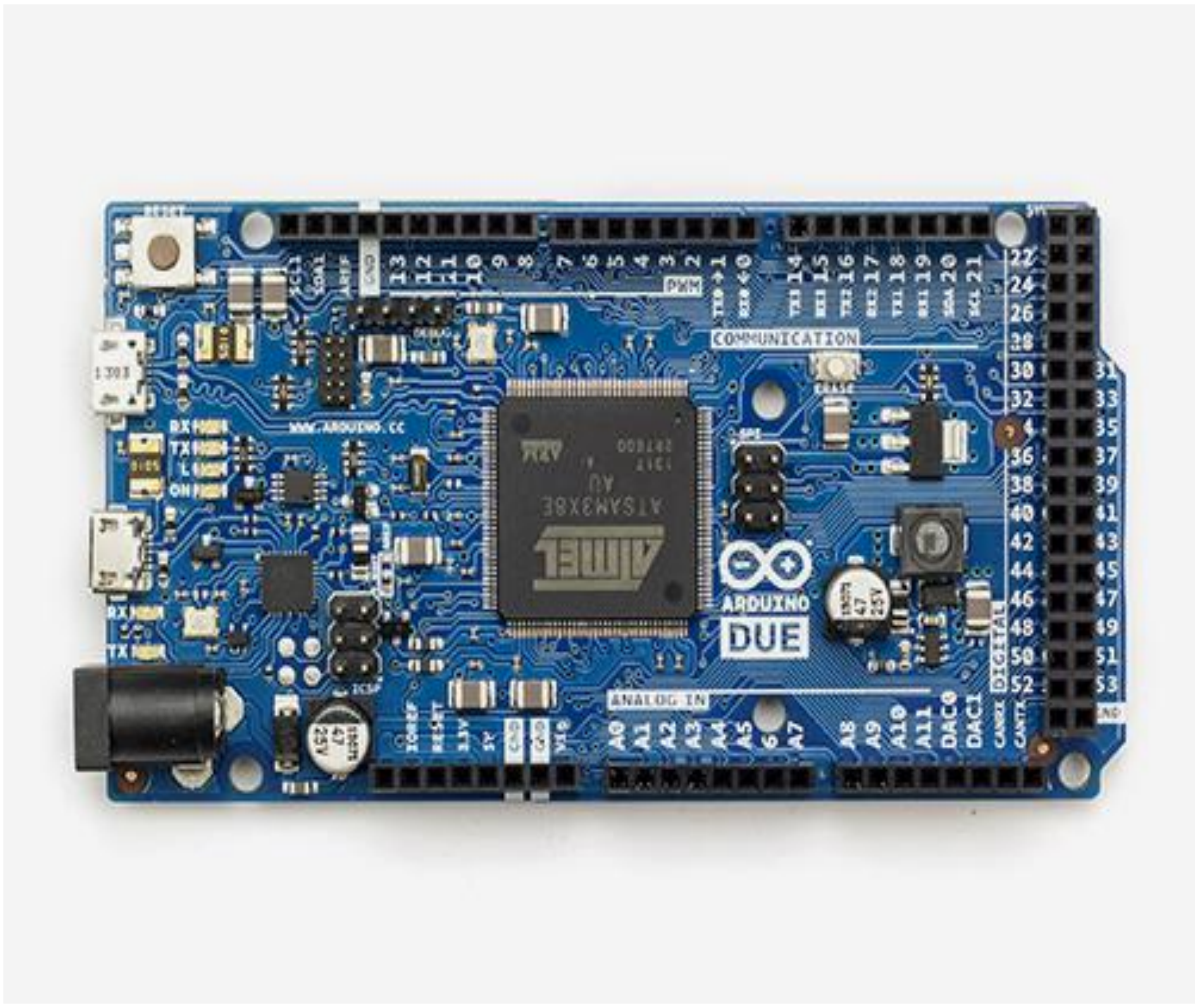


Figure 1.3 Arduino Due Board

The board contains everything needed to support the microcontroller; simply connect it to a computer with a micro-USB cable or power it with a AC-to-DC adapter or battery to get started. The Due is compatible with all Arduino shields that work at 3.3V and are compliant with the 1.0 Arduino pinout.

The Due follows the 1.0 pinout which consists of:

- TWI which has SDA and SCL pins that are near to the AREF pin.
- IOREF that allows an attached shield with the proper configuration to adapt to the voltage provided by the board. This enables shield compatibility with a 3.3V board like the Due and AVR-based boards which operate at 5V.
- An unconnected pin, reserved for future use.

Table1: Technical Specifications of Arduino Due

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

ARM Core Benefits

- A 32-bit core, that allows operations on 4 bytes wide data within a single CPU clock.
- CPU Clock at 84Mhz.
- 96 KBytes of SRAM.
- 512 KBytes of Flash memory for code.
- A DMA controller, that can relieve the CPU from doing memory intensive tasks.

Power Supply:

The Arduino Due can be powered via the USB connector or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

Table 2: Power Pins and their functions:

Power Pins	Functions
Vin	The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or if supplying voltage via the power jack, access it through this pin.
5V	This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board.
3.3V	A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 800 mA. This regulator also provides the power supply to the SAM3X microcontroller.
IOREF	This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly

	configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.
GND	Ground pins.

Memory:

The SAM3X has 512 KB (2 blocks of 256 KB) of flash memory for storing code. The bootloader is pre-burned in factory from Atmel and is stored in a dedicated ROM memory. The available SRAM is 96 KB in two contiguous bank of 64 KB and 32 KB. All the available memory (Flash, RAM and ROM) can be accessed directly as a flat addressing space.

It is possible to erase the Flash memory of the SAM3X with the onboard erase button. This will remove the currently loaded sketch from the MCU. To erase, press and hold the Erase button for a few seconds while the board is powered.

Input and Output pins:

- Digital I/O: pins from 0 to 53
- Each of the 54 digital pins on the Due can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 3.3 volts. Each pin can provide (source) a current of 3 mA or 15 mA, depending on the pin, or receive (sink) a current of 6 mA or 9 mA, depending on the pin. They also have an internal pull-up resistor (disconnected by default) of 100 KOhm. In addition, some pins have specialized functions:
- Serial: 0 (RX) and 1 (TX)
- Serial 1: 19 (RX) and 18 (TX)
- Serial 2: 17 (RX) and 16 (TX)
- Serial 3: 15 (RX) and 14 (TX)

  Used to receive (RX) and transmit (TX) TTL serial data (with 3.3 V level). Pins 0 and 1 are connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- PWM: Pins 2 to 13

  Provide 8-bit PWM output with the analogWrite() function. the resolution of the PWM can be changed with the analogWriteResolution() function.
- SPI: SPI header (ICSP header on other Arduino boards)

  These pins support SPI communication using the SPI library. The SPI pins are broken out on the central 6-pin header, which is physically compatible with the Uno, Leonardo and Mega2560. The SPI header can be used only to communicate with other SPI devices, not for programming the SAM3X with the In-Circuit-Serial-Programming technique. The SPI of the Due has also advanced features that can be used with the Extended SPI methods for Due.

- CAN: CANRX and CANTX
  These pins support the CAN communication protocol but are not not yet supported by Arduino APIs.
- "L" LED: 13
  There is a built-in LED connected to digital pin 13. When the pin is HIGH, the LED is on, when the pin is LOW, it's off. It is also possible to dim the LED because the digital pin 13 is also a PWM output.
- TWI 1: 20 (SDA) and 21 (SCL)
- TWI 2: SDA1 and SCL1.
  Support TWI communication using the Wire library. SDA1 and SCL1 can be controlled using the Wire1 class provided by the Wire library. While SDA and SCL have internal pullup resistors, SDA1 and SCL1 have not. Adding two pullup resistor on SDA1 and SCL1 lines is required for using Wire1.
- Analog Inputs: pins from A0 to A11
  The Due has 12 analog inputs, each of which can provide 12 bits of resolution (i.e. 4096 different values). By default, the resolution of the readings is set at 10 bits, for compatibility with other Arduino boards. It is possible to change the resolution of the ADC with analogReadResolution(). The Due's analog inputs pins measure from ground to a maximum value of 3.3V. Applying more than 3.3V on the Due's pins will damage the SAM3X chip. The analogReference() function is ignored on the Due.

  The AREF pin is connected to the SAM3X analog reference pin through a resistor bridge. To use the AREF pin, resistor BR1 must be desoldered from the PCB.

- DAC1 and DAC2
  These pins provides true analog outputs with 12-bits resolution (4096 levels) with the analogWrite() function. These pins can be used to create an audio output using the Audio library.

Other pins on the board:

- AREF :
  Reference voltage for the analog inputs. Used with analogReference().
- Reset
  Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

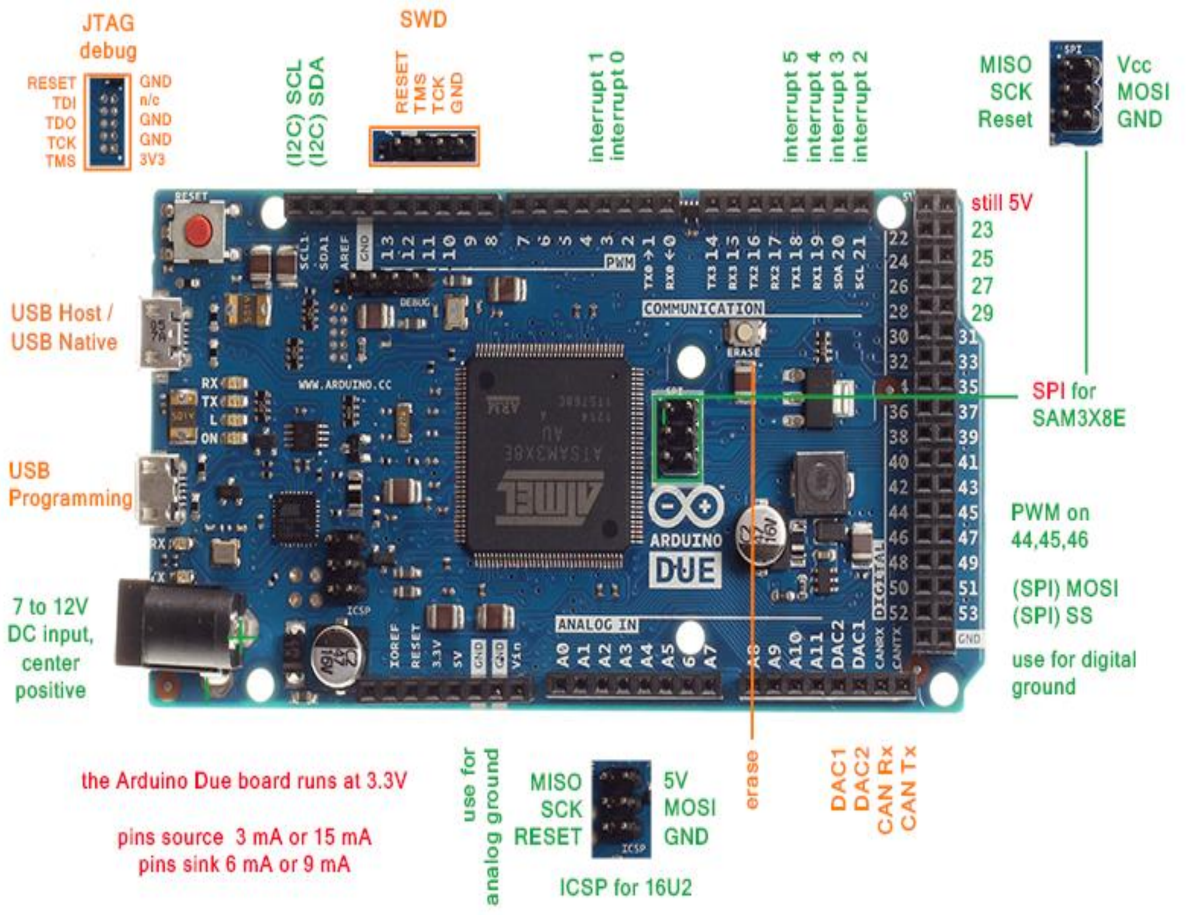


Figure 1.4 :Arduino Due pinout diagram

Communication:

The Arduino Due has a number of facilities for communicating with a computer, another Arduino or other microcontrollers, and different devices like phones, tablets, cameras and so on. The SAM3X provides one hardware UART and three hardware USARTs for TTL (3.3V) serial communication.

The Programming port is connected to an ATMega16U2, which provides a virtual COM port to software on a connected computer (To recognize the device, Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically). The 16U2 is also connected to the SAM3X hardware UART. Serial on pins RX0 and TX0 provides Serial-to-USB communication for programming the board through the ATmega16U2 microcontroller. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

The Native USB port is connected to the SAM3X. It allows for serial (CDC) communication over USB. This provides a serial connection to the Serial Monitor or other applications on your computer. It also enables the Due to emulate a USB mouse or keyboard to an attached computer. To use these features, see the Mouse and Keyboard library reference pages.

The Native USB port can also act as a USB host for connected peripherals such as mice, keyboards, and smartphones.

USB Overcurrent Protection

The Arduino Due has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

1.1.3 Atmel SAM3X

The Atmel SAM3X/A series is a member of a family of Flash microcontrollers based on the high performance 32-bit ARM Cortex -M3 RISC processor. It operates at a maximum speed of 84 MHz and features up to 512 Kbytes of Flash and up to 100 Kbytes of SRAM. The peripheral set includes a High Speed USB Host and Device port with embedded transceiver, an Ethernet MAC, 2 CANs, a High Speed MCI for SDIO/SD/MMC, an External Bus Interface with NAND Flash Controller (NFC), 5 UARTs, 2 TWIs, 4 SPIs, as well as a PWM timer, three 3-channel general-purpose 32-bit timers, a low-power RTC, a lowpower RTT, 256-bit General Purpose Backup Registers, a 12-bit ADC and a 12-bit DAC. The SAM3X/A devices have three software-selectable low-power modes: Sleep, Wait and Backup. In Sleep mode, the processor is stopped while all other functions can be kept running. In Wait mode, all clocks and functions are stopped but some peripherals can be configured to wake up the system based on predefined conditions. In Backup mode, only the RTC, RTT, and wake-up logic are running. The SAM3X/A series is ready for capacitive touch thanks to the QTouch library, offering an easy way to implement buttons, wheels and sliders. The SAM3X/A architecture is specifically designed to sustain high-speed data transfers. It includes a multi-layer bus matrix as well as multiple SRAM banks, PDC and DMA channels that enable it to run tasks in parallel and maximize data throughput. The device operates from 1.62V to 3.6V and is available in 100 and 144-lead LQFP, 100-ball TFBGA and 144-ball LFBGA packages.

Figure 1.5: ATMEL SAM3X

Key Features:

- **Optimized for Connectivity** — With its architecture and peripherals including Ethernet, dual CAN and HS USB MiniHost and device with on-chip PHY, the SAM3X is optimized for applications requiring high levels of connectivity.
- **Atmel QTouch Capacitive Touch Support** — The SAM3X is touch-ready, offering native support for Atmel QTouch® technology for easy implementation of buttons, sliders and wheels functionality in your applications.
- **Enhanced Safety and Security** — A variety of features integrated into the SAM3X series provide safety and security for your system. For example, dual-bank Flash enables safe in-system firmware upgrades. On-the-fly external memory scrambling on the 16-bit external bus interface enhances the protection of your external memory content, without impacting system performance. In addition, the SAM3X offers clock failure detection and a true random number generator.
- **Low Power Consumption** — All SAM3 MCUs feature a sophisticated, flexible power management scheme that minimizes power consumption under all usage conditions. You can put the devices in back-up mode with the core and peripherals powered down, reducing power down to 2.5uA for the SAM3X series. A high-speed on-chip RC oscillator accelerates wake-up from back-up mode, further reducing average power consumption.

Application of Atmel Sam3X:

The SAM3X/A devices are particularly well suited for networking applications: industrial and home/building automation, gateways.

1.1.4 Physical Characteristics

The maximum length and width of the Arduino Due PCB are 4 and 2.1 inches respectively, with the USB connectors and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

Table 3: Advantages of Arduino Due

Advantage	Explanation
Ready to Use	One doesn't have to think about programmer connections for programming or any other interface.
Examples of codes	Library of examples present inside the software of Arduino.
Effortless functions	Its automatic unit conversion capability.
Flexibility	Efficient shift of technology and resources.
Mobility	Can be used anytime anywhere with just the help of pre-programmed Arduino due

CHAPTER 2

Softwares Used for Implementation

We first start with the software implementation of function generator .A completely different approach to function generation is to use software instructions to generate a waveform, with provision for output. For example, a general-purpose digital computer can be used to generate the waveform; if frequency range and amplitude are acceptable, the sound card fitted to most computers can be used to output the generated wave. The software's that will be used in implementation are discussed below:

2.1 **Arduino Software(IDE)**

The Due can be programmed with the Arduino Software (IDE). 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.

2.1.1 Variables :

- Float : Datatype for floating-point numbers, a number that has a decimal point. Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers. Floating-point numbers can be as large as 3.4028235E+38 and as low as -3.4028235E+38. They are stored as 32 bits (4 bytes) of information.
- String : Text strings can be represented in two ways. You can use the String data type, which is part of the core as of version 0019, or you can make a string out of an array of type char and null-terminate it.
- Byte : A byte stores an 8-bit unsigned number, from 0 to 255.
- Char : A data type that takes up 1 byte of memory that stores a character value. Character literals are written in single quotes, like this: 'A' (for multiple characters - strings - use double quotes: "ABC").
- Int : A data type that takes up 1 byte of memory that stores a character value. Character literals are written in single quotes, like this: 'A' (for multiple characters - strings - use double quotes: "ABC"). For the Arduino Due, it stores a 32 bit value.
- Boolean : A **boolean** holds one of two values, true or false. (Each boolean variable occupies one byte of memory.)

2.1.2 Structure

- Serial.begin(9600) : opens serial port, sets data rate to 9600 bps
- Continue : The continue statement skips the rest of the current iteration of a loop (**do**, **for**, or **while**). It continues by checking the conditional expression of the loop, and proceeding with any subsequent iterations.
- Setup () : Function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.
- Loop () : After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

2.1.3 Functions

- analogWriteResolution() : sets the resolution of the analogWrite() function. It defaults to 8 bits (values between 0-255) for backward compatibility with AVR based boards.
- analogReadResolution() : Sets the size (in bits) of the value returned by analogRead(). It defaults to 10 bits (returns values between 0-1023) for backward compatibility with AVR based boards. The **Due and the Zero have 12-bit ADC capabilities** that can be accessed by changing the resolution to 12. This will return values from analogRead() between 0 and 4095.
- *lcd.*begin(cols, rows) : lcd: a variable of type LiquidCrystal

  cols: the number of columns that the display has

  rows: the number of rows that the display has

- getkey () : Reports the ASCII value of a key being pressed or released on an attached USB keyboard.
- lcd.setCursor () : To reposition the cursor to a particular location.
- lcd.print () : Prints text to the LCD.
- lcd.noBlink () : Turns off the blinking LCD cursor.
- toInt () : Converts a valid String to an integer. The input string should start with an integer number.
- makeKeymap () : Initializes the internal keymap to be equal to userKeymap
- analogWrite () : Writes an analog value to a pin.
- delalyMicroseconds () : Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.
- lcd.Blink () : Display the blinking LCD cursor.
- lcd.Clear () : Clears the LCD screen and positions the cursor in the upper-left corner.

Programming: The SAM3X also supports TWI and SPI communication. The Arduino software includes a Wire library to simplify use of the TWI bus; see the documentation for details. For SPI communication, use the SPI library.

Loading sketches to the SAM3X is different than the AVR microcontrollers found in other Arduino boards because the flash memory needs to be erased before being re-programmed. Upload to the chip is managed by ROM on the SAM3X, which is run only when the chip's flash memory is empty.

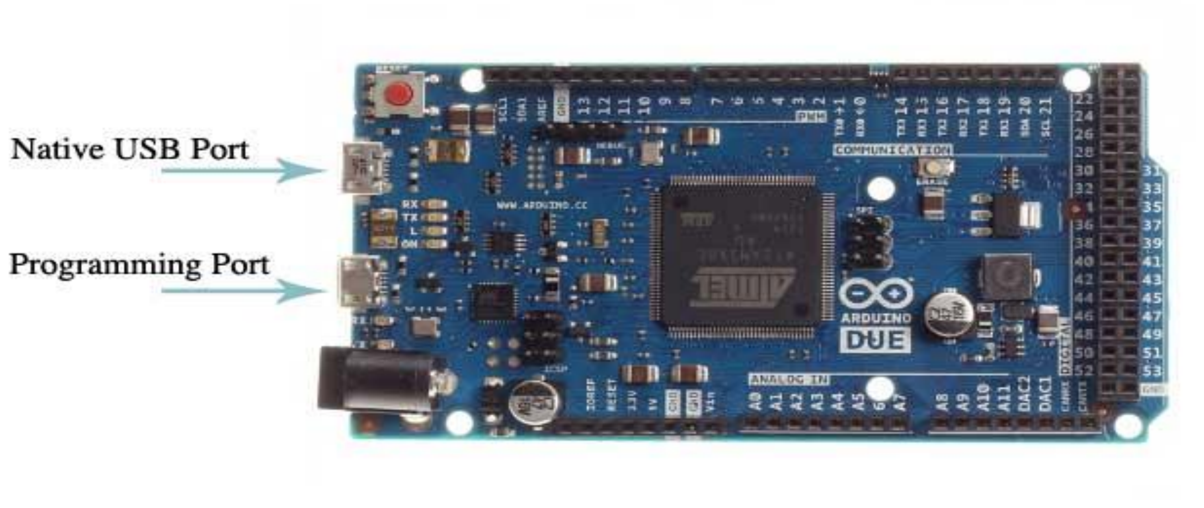


Figure 2.1: Arduino Due board showing the Native USB and the Programming Port

Either of the USB ports can be used for programming the board, though it is recommended to use the Programming port due to the way the erasing of the chip is handled :

- Programming port: To use this port, select "Arduino Due (Programming Port)" as your board in the Arduino IDE. Connect the Due's programming port (the one closest to the DC power jack) to your computer. The programming port uses the 16U2 as a USB-to-serial chip connected to the first UART of the SAM3X (RX0 and TX0). The 16U2 has two pins connected to the Reset and Erase pins of the SAM3X. Opening and closing the Programming port connected at 1200bps triggers a “hard erase” procedure of the SAM3X chip, activating the Erase and Reset pins on the SAM3X before communicating with the UART. This is the recommended port for programming the Due. It is more reliable than the "soft erase" that occurs on the Native port, and it should work even if the main MCU has crashed.
- Native port: To use this port, select "Arduino Due (Native USB Port)" as your board in the Arduino IDE. The Native USB port is connected directly to the SAM3X. Connect the Due's Native USB port (the one closest to the reset button) to your computer. Opening and closing the Native port at 1200bps triggers a 'soft erase' procedure: the flash memory is erased and the board is restarted with the bootloader. If the MCU crashed for some reason it is likely that the soft erase procedure won't work as this procedure happens entirely in software on the SAM3X. Opening and closing the native port at a different baudrate will not reset the SAM3X.

Unlike other Arduino boards which use avrdude for uploading, the Due relies on bossac.

The ATmega16U2 firmware source code is available in the Arduino repository. You can use the ISP header with an external programmer (overwriting the DFU bootloader).

2.2 Interfacing an LCD

The LiquidCrystal library allows you to control LCD displays that are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually tell them by the 16-pin interface.

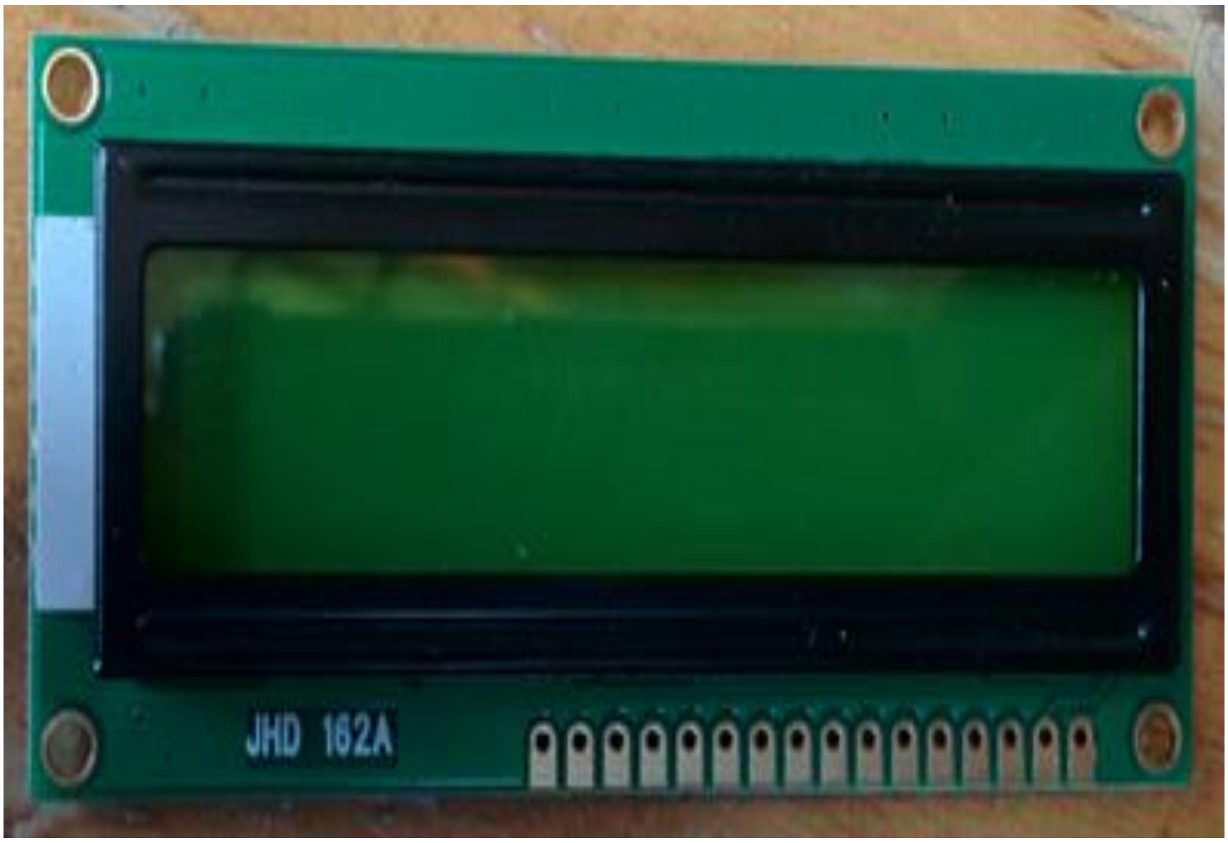


Figure 2.2:A 16x2 LCD

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

A **register select (RS) pin** that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

A **Read/Write (R/W) pin** that selects reading mode or writing mode

An **Enable pin** that enables writing to the registers

8 **data pins (D0 -D7)**. The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

There's also a **display contrast pin (Vo)**, **power supply pins (+5V and Gnd)** and **LED Backlight (Bklt+ and** BKlt**-)** pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The **LiquidCrystal Library** simplifies this for you so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

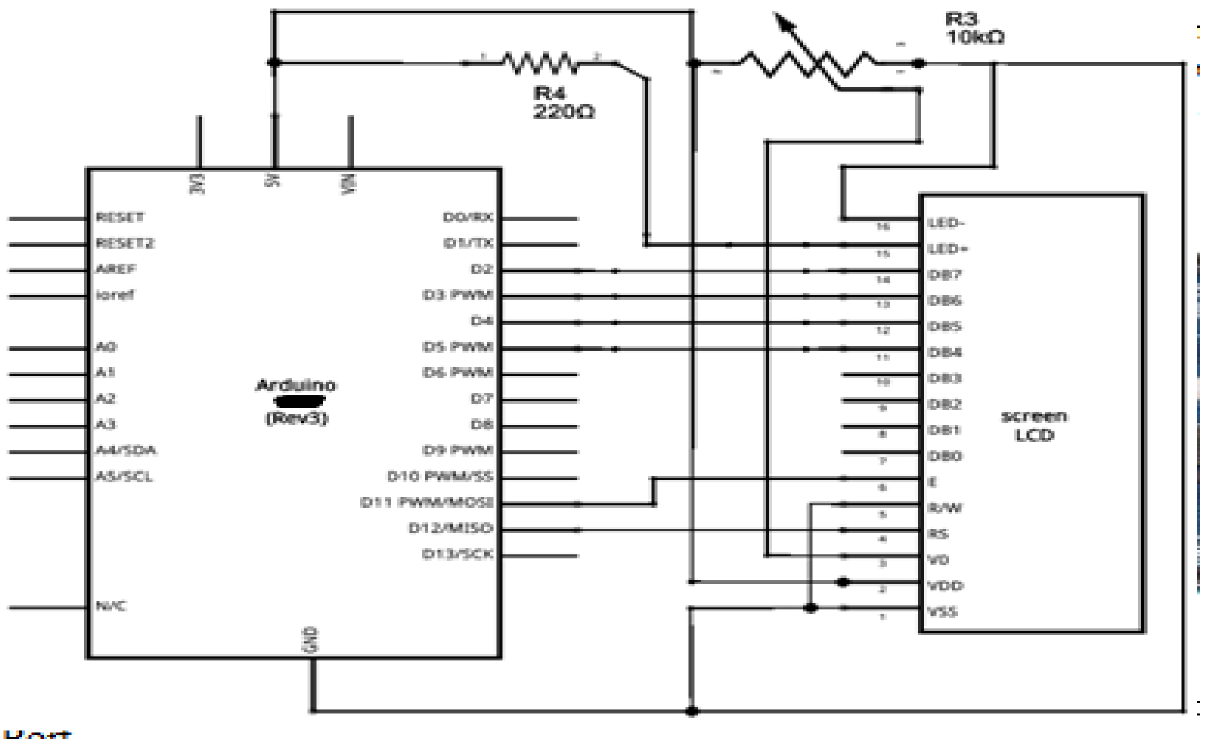


Figure 2.4 : Schematic diagram of LCD Interfacing with Arduino

2.3 Interfacing the Keypad

The Keypad library allows your Arduino to read a matrix type keypad. You can scavenge these keypads from old telephones or you can get them from almost any electronics parts store for less than $5 USD. They come in 3x4, 4x4 and various other configurations with words, letters and numbers written on the keys. This library is capable of supporting all of those.

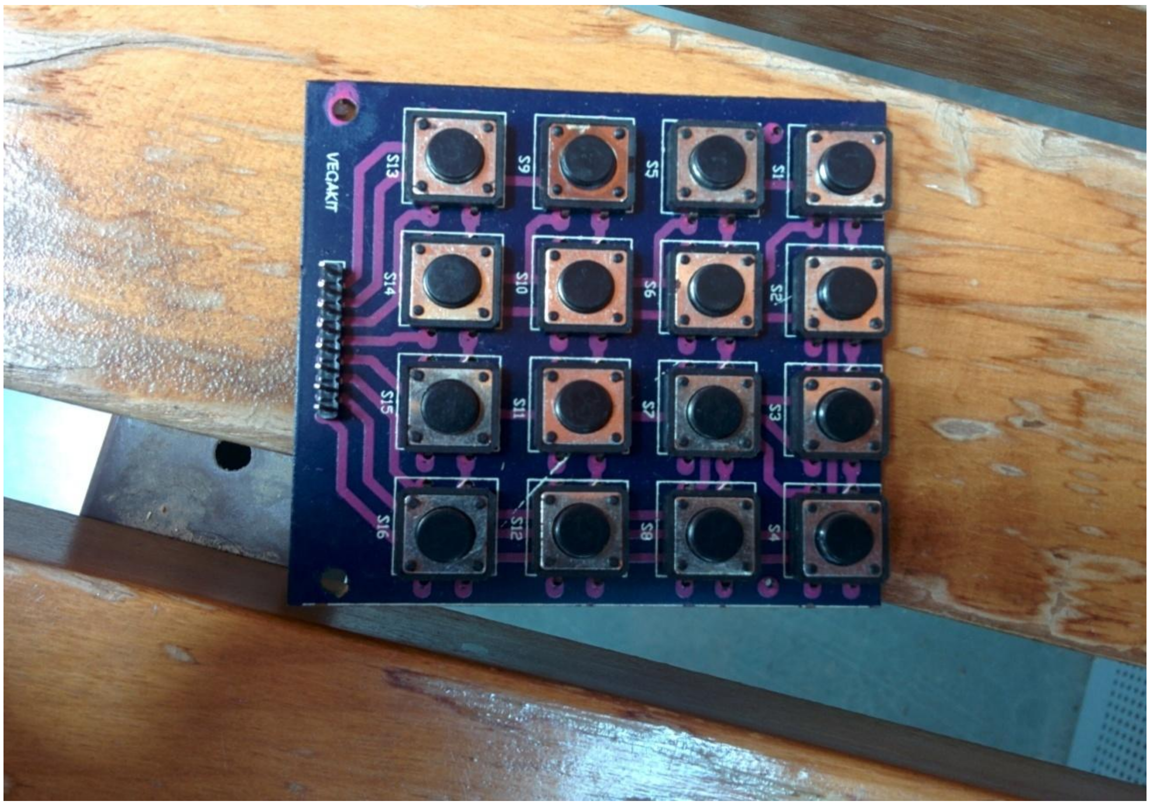
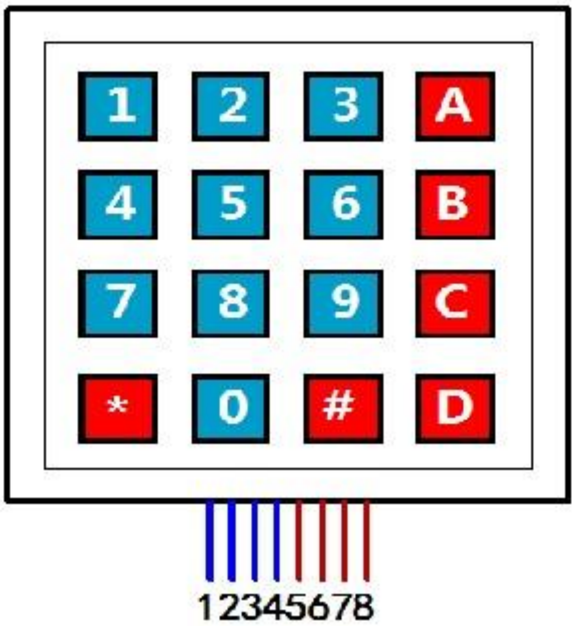
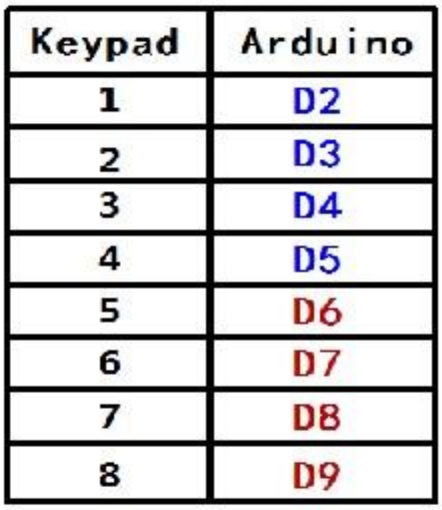


Figure2.4: 4x4 Keypad

2.3.1Procedure:





Keypad	Arduino
1	D2
2	D3
3	D4
4	D5
5	D6
6	D7
7	D8
8	D9

Figure 2.5 : Interfacing the 4X4 Keypad with the corresponding pins of Arduino Due

1.Connect your Ohm meter leads to pins 1 and 2.
2.Press all the buttons until the meter indicates a closure.
3.Write down the pin numbers next to the column and row for the key you just found. Example: Your meter is connected to pins 1 and 5. When you pressed the number 7 your meter reacted. Write 1 under COL0 and 5 next to ROW2.
4.If the meter didn't react then move the meter lead from pin 2 to pin 3 and repeat steps 2 and 3 above.
5.Now, keep moving the lead to the next pin and repeat steps 2 and 3 for each pin.
6.Once you have reached the end move the first meter lead from pin 1 to pin 2 and repeat steps 2 and 3 while connecting the second meter lead to pins 3 through the highest pin.
7.Once you have completely identified all the pins on the diagram then you can safely ignore any unused keypad pins. You are now ready to wire the keypad to your Arduino.

2.3.2 Troubleshooting

1. You can pretty much connect your keypad to any pins you would like. Be careful not to use the serial pins (0 and 1) if you are using them for communication.

2. If key presses seem to take a long time to show up then you are probably using long delay()'s in your code. The same thing can happen if you use too many small delay()s like delay(10).

3. Make sure you understand the pin mappings and have the keypad wired up to match. If you wired the pins incorrectly (and already soldered them in) then you may be able to fix it by redefining the pins and/or keymap to make your keypad work.

CHAPTER 3

PROPOSED WORK

In this chapter we will discuss the code for implementing a function generator using Arduino Due and the steps involved.

3.1 Code

```
#include <Keypad.h>

#include <LiquidCrystal.h>

 //Set the desired duty cycle in percentage

float dc,t,a,d;

float sample_delay1;

LiquidCrystal lcd(27, 26, 25, 24, 23, 22);

float sample_delay;

String num1,num2;

int wave1,duty_cycle,k;

const byte ROWS = 4; //four rows

const byte COLS = 4; //four columns

//define the cymbols on the buttons of the keypads

char hexaKeys[ROWS][COLS] = {

  {'1','2','3','A'},

  {'4','5','6','B'},#include "Waveforms.h"


  {'7','8','9','C'},

  {'.','0','E','D'}

};

byte rowPins[ROWS] = {5, 4, 3, 2}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {9, 8, 7, 6}; //connect to the column pinouts of the keypad

int i = 0;

int sample;

//initialize an instance of class NewKeypad

Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS,
COLS);

boolean in_setup = true;

void setup()

{

  Serial.begin(9600);

  analogWriteResolution(12);  // set the analog output resolution to 12 bit (4096 levels)

  analogReadResolution(12);

  lcd.begin(16,2);

  char key = customKeypad.getKey();

  lcd.setCursor(0, 0);

  lcd.print("freq= ");

  while(in_setup == true)

  {

        char key = customKeypad.getKey();

        if(key == 'A')

        {

          wave1 = 0;

          lcd.setCursor(0,1);

          lcd.print("sine_wave");

        }

        if(key == 'B')

        {
```

```
  wave1 = 1;
  lcd.setCursor(0,1);
  lcd.print("triangular");
}
if(key == 'C')
{
  wave1 = 2;
  lcd.setCursor(0,1);
  lcd.print("sawtooth");
}
if(key == 'D')
{
  wave1 = 3;
  lcd.setCursor(0,1);
  lcd.print("square");
}
if(key == 'E')
{
  lcd.noBlink();
  k = num1.toInt();
  sample_delay=1000000/(k*120);
  sample_delay1 = sample_delay - 9;
  in_setup = false;
  if(wave1 == 3)
  {
    lcd.setCursor(0,1);
    lcd.print("d.cycle ");
```

```
      while(true)
      {
        key = customKeypad.getKey();
if(key!=NO_KEY&&(key=='1'||key=='2'||key=='3'||key=='4'||key=='5'||key=='6'||key=='
7'||key=='8'||key=='9'||key=='0'||key=='.'))
        {
          num2 = num2 + key;
    //int numLength = num1.length();
          lcd.setCursor(8,1);  //to adjust one whitespace for operator
          lcd.print(num2);
        }
        if(key == 'E')
        {
          duty_cycle = num2.toInt();
          dc=duty_cycle*0.01;
          t=1/k;
          d=t*dc; //d=total time*duty cycle(on time)
          a=t-d;  //a=off time
          //calculation of delays
          break;
        }
      }
      while(true)
      {
        analogWrite(DAC1, 4095);
        analogWrite(DAC0, 0);
        delayMicroseconds(1000000*d);
        analogWrite(DAC1, 0);
```

```
              analogWrite(DAC0, 4095);

               delayMicroseconds(1000000*a);

              //square output

            }

          }

        }
if(key!=NO_KEY&&(key=='1'||key=='2'||key=='3'||key=='4'||key=='5'||key=='6'||key=='
7'||key=='8'||key=='9'||key=='0'))

        {

          num1 = num1 + key;

          //int numLength = num1.length();

          //to adjust one whitespace for operator

          lcd.setCursor(5,0);

          lcd.print(num1);

          lcd.blink();

        }

        if(key == '.')

        {

          num1 = "";

          num2 = "";

          k = 1;

          duty_cycle = 50;

          i=0;

          lcd.clear();

          continue;

        }

  }

}
```

```
void loop(){
  analogWrite(DAC1, waveformsTable[wave1][i] );
  i++;
  if(i == maxSamplesNum)  // Reset the counter to repeat the wave
    i = 0;
 delayMicroseconds(sample_delay1);
}
```

3.2 Explanation for the code developed

1. Setup the Arduino board with data rate as 9600bps.

2. Interface the keypad with the board.

· Key A is defined for the Sine wave.

· Key B is defined for the Triangular wave.

· Key C is defined for the Sawtooth wave.

· Key D is defined for the Square wave.

· Key E is for entering the frequency of the wave selected.

· Keys 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and . are used for specifying the numeric inputs.

3. Configure the Liquid Crystal Display (LCD) at 12 bit resolution.
4. Input is taken from the user.
5. Required wave shape is generated with the frequency specified by the user.

CHAPTER 4
CONCLUSION

4.1 Conclusion

The Arduino Due based Function Generator thus implemented has a lot of advantages over a regular Function Generator. The regular Function Generator that costs over $150, whereas the entire setup for the Arduino Based version can be purchased for $35 (approx.). The size of a regular Function Generator is much more as compared to that of the Arduino based one. The Arduino based Function Generator can fit inside a geometry box also, hence making it a lot more portable as compared to the regular one.

There are vast implementations of the Arduino based Function Generator. It could be used for on-site debugging and software implementations in the communication industry very well as the device is more portable. Also, it could be easily taken to remote locations.

A function generator is the basic electronic equipment required by students to understand Physics better. The Arduino based version could well be used in educational institutes that lack resources for the procurement of equipment. Since the Arduino itself is an open – source hardware, they could use the same board for other applications too. To gain further insights into the process of wave generation, students could themselves program the Arduino board as a Function Generator. An amplifier could also be interfaced with the Arduino board and students could gain insights into how each of these waves sound like.

References

[1]http://www.instructables.com/id/Arduino-Waveform-Generator/
[2]https://www.arduino.cc/en/Tutorial/DueSimpleWaveformGenerator
[3]http://www.academia.edu/2295290/pulse_and_square_wave_generator_funct
ion_generator
[4]http://courses.washington.edu/phys431/scope_ex/scope_ex.pdf
[5]https://blog.arduino.cc/category/research/
[6]http://www.engpaper.net/arduino.htm
[7]https://www.arduino.cc/en/Main/ArduinoBoardDue
[8]http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf
[9] https://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems
[10] http://in.mathworks.com/products/matlab/?requestedDomain=www.mathworks.com
[11] http://www.atmel.com/products/microcontrollers/arm/sam3x.aspx
[12] http://www.atmel.com/products/microcontrollers/arm/sam3x.aspx?tab=documents
[13] http://www.cvarc.org/new-wp/download/technical/IntroToarduino.pdf
[14] http://scholar.utc.edu/cgi/viewcontent.cgi?article=1509&context=theses
[15] http://www.xilinx.com/products/design-tools/ise-design-suite.html
[16] http://www.engpaper.net/arduino/atmel/sam3x.htm