

Signal Generator Using Direct Digital Synthesizer

Group D14

Group Members

Raghavendra Bhushan Karn (04D07017) rbhushan@ee.iitb.ac.in

Sharayu Arun Moharir (04D07018) sharayu@ee.iitb.ac.in

Namrata Bandekar (04D10023) namrata@ee.iitb.ac.in

Supervisor Prof. Girish Kumar

Course Instructor Prof. H. Narayanan

Abstract

This report discusses the design of a Direct Digital Synthesizer (DDS) based Function Generator which can generate sine waves of frequencies lower than 100 MHz with an accuracy of 1% of the output frequency. It describes a micro-controller based interface to the DDS Chip AD9953 for generation of user desired frequencies less than 100 MHz.

1 Introduction

Direct digital synthesis (DDS) is a method of producing an analog waveform—usually a sine wave—by generating a time-varying signal in digital form and then performing a digital-to-analog conversion. As the operations within a DDS device are primarily digital, therefore it can offer fast switching between output frequencies, fine frequency resolution, and operation over a broad spectrum of frequencies. With advances in design and process technology, DDS devices are very compact and draw little power. For example, the AD9953, a DDS-based programmable waveform generator, operating at 1.8 V with a 20 MHz clock, consumes a maximum power of 171 mW.

Many possibilities for frequency generation are open to a designer, ranging from phase-locked-loop (PLL)-based techniques for very high-frequency synthesis, to dynamic programming of digital-to-analog converter (DAC) outputs to generate arbitrary waveforms at lower frequencies. But the DDS technique is rapidly gaining acceptance for solving frequency- (or waveform) generation requirements in both communications and industrial applications because single-chip IC devices can generate programmable analog output waveforms simply and with high resolution and accuracy.

2 Design Approach

2.1 Hardware

Block Diagram

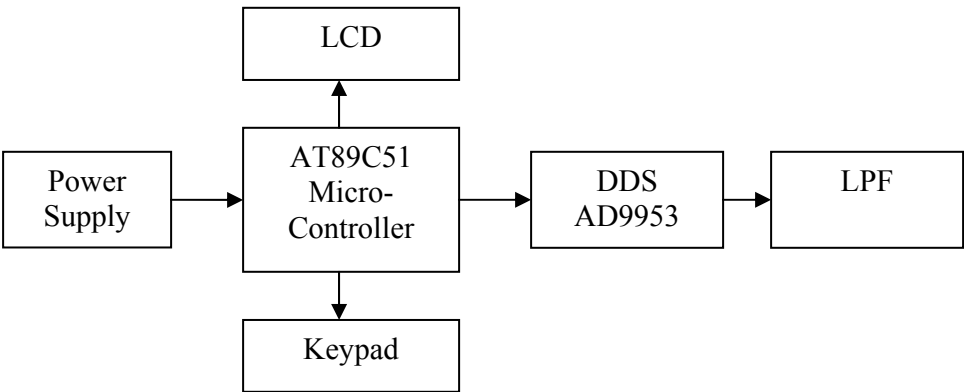


Figure 1: Basic Design Blocks

2.1.1 Power Supply

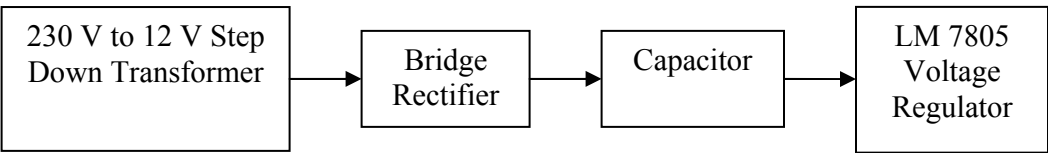


Figure 2: Power Supply Block Diagram

2.1.2 LCD

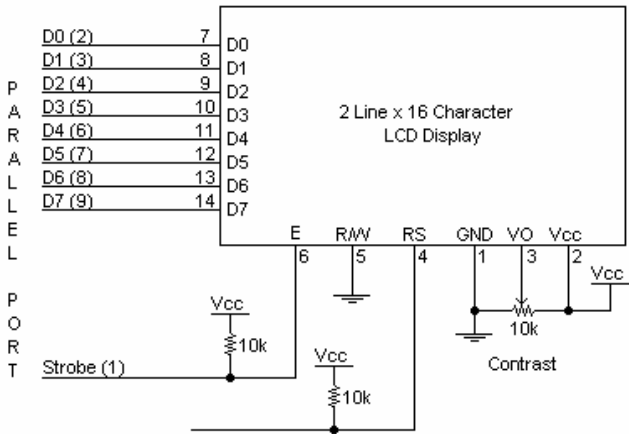


Figure 3: LCD

A 16x2 LCD is used to display the frequency and amplitude of the signal that is generated by the DDS.

2.1.3 Keypad

A 4x4 matrix keypad has been used for taking user input, for entering the desired frequency and amplitude of the signal. The LCD is used in conjunction with the keypad. Both the LCD and the keypad have been put exclusively for this purpose.

The keypad is attached to one of the ports of the microcontroller. The Micro-controller polls for a key press. Once a key press and key release is detected it calculates the key code from the port value to determine which key is pressed. The user can choose from three ranges of frequency: Hz, KHz, MHz.

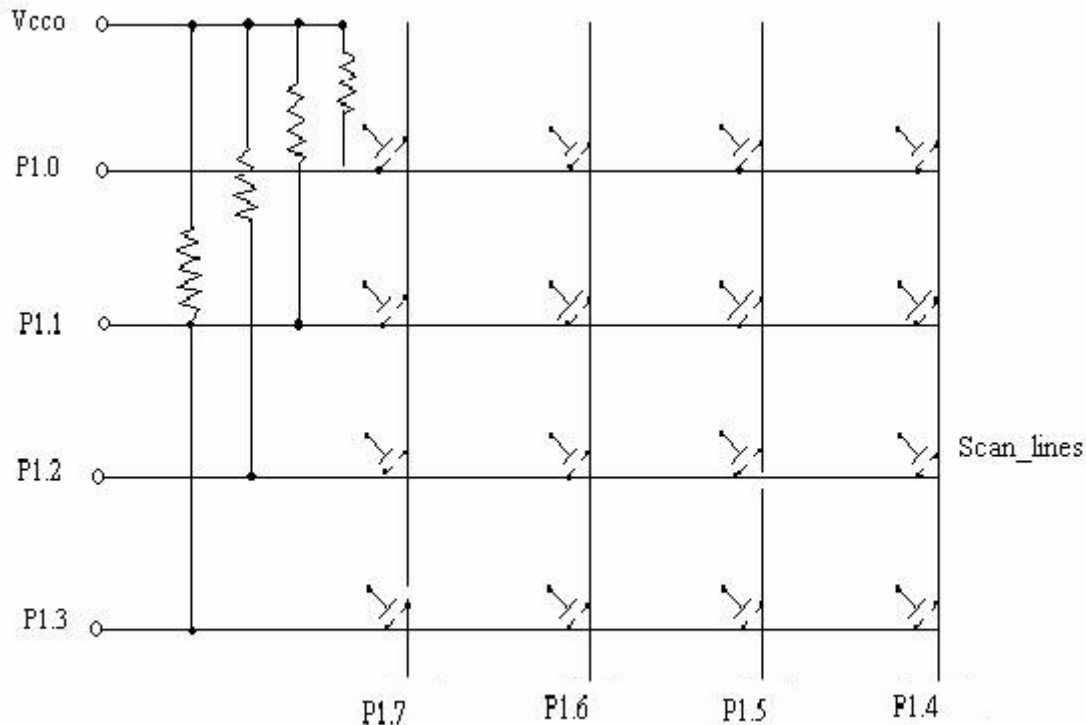


Figure 4: Keypad

2.1.4 DDS

The AD9953 is a direct digital synthesizer (DDS) featuring a 14-bit DAC operating up to 400 MSPS. The AD9953 uses advanced DDS technology, coupled with an internal high speed, high performance DAC to form a digitally programmable, complete high frequency synthesizer capable of generating a frequency-agile analog output sinusoidal waveform at up to 100 MHz. The AD9953 is designed to provide fast frequency hopping and fine tuning resolution (32-bit frequency tuning word). The frequency tuning and

control words are loaded into the AD9953 via a serial I/O port. The AD9953 is specified to operate over the extended industrial temperature range of -40°C to $+105^{\circ}\text{C}$.

2.1.5 Low Pass Filter (LPF)

The DDS chip generates high frequency noise due to the DAC. The output therefore needs to be passed through a low pass filter with band frequency of 200 MHz. Various filters were simulated using RFSim and a four pole Butterworth filter was found to have a desired frequency response.

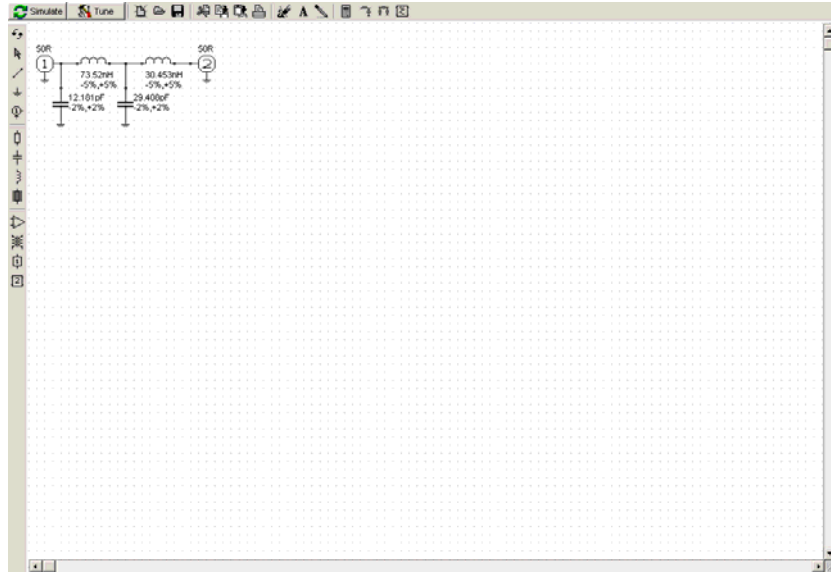


Figure 5: Filter circuit

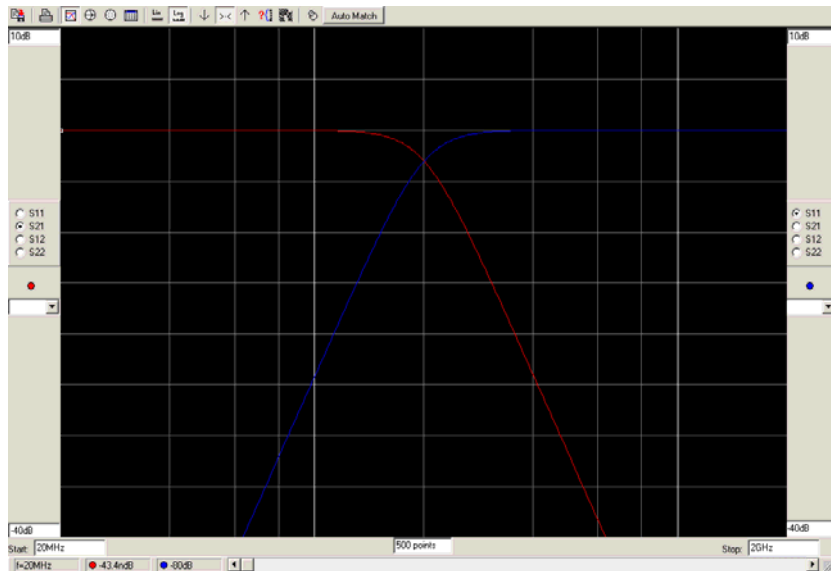


Figure 6: Filter response

2.1.6 PCB design for the DDS

The PCB needed a 50 ohms impedance for the high frequency output. A software called Line Gauge was used to determine the appropriate dimensions. According to the manufacturer's specifications, 1.50 mm was the appropriate width needed to create a 50 ohms track. Glass epoxy (FR4) was used as the substrate.

All the supply pins have been bypassed with a large capacitor (10 μ F) to provide a low impedance path for AC and 2 10 nF capacitors to avoid any RF signal to reach the voltage input pins of the DDS.

The circuit takes in 5V as the input voltage. Two TPS772218s have been used to obtain digital and analog voltages of 1.8V. The analog and digital grounds have been isolated using an inductor.

There are no low frequency lines of thickness less than twice the thickness of the board away from the line carrying the high frequency signal to avoid any kind of interference due to the high frequency signal.

An attenuator is used at the output of the DDS to protect the oscillator in the DDS from open circuit or short circuit at the output. Thus, in the case of open circuit or short circuit, the reflected waveform gets attenuated. The attenuator was designed using RFSim.

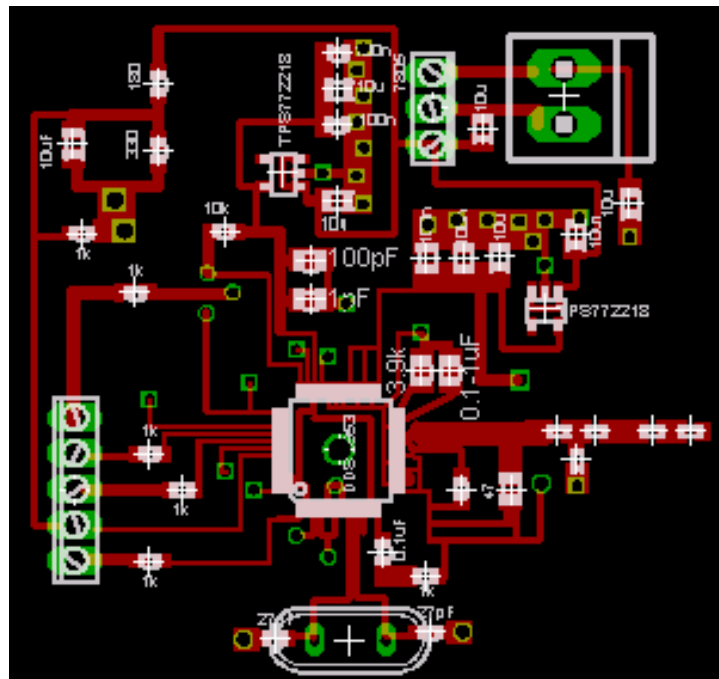


Figure 7: DDS PCB

Figure 9: Circuit Diagram for Microcontroller



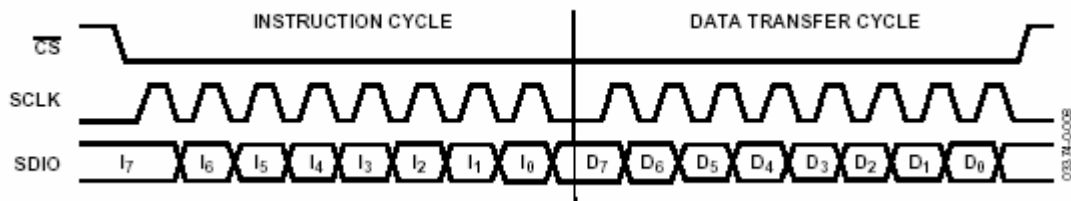
4 Software

4.1 DDS

There are two phases to a communication cycle with the AD9953. Phase 1 is the instruction cycle, which is the writing of an instruction byte into the AD9953, coincident with the first eight SCLK rising edges. The instruction byte provides the AD9953 serial port controller with information regarding the data transfer cycle, which is Phase 2 of the communication cycle.

The Phase 1 instruction byte defines whether the upcoming data transfer is read or write and the serial address of the register being accessed. The first eight SCLK rising edges of each communication cycle are used to write the instruction byte into the AD9953. The remaining SCLK edges are for Phase 2 of the communication cycle. Phase 2 is the actual data transfer between the AD9953 and the system controller. The number of bytes transferred during Phase 2 of the communication cycle is a function of the register being accessed.

After transferring all data bytes per the instruction, the communication cycle is completed. At the completion of any communication cycle, the AD9953 serial port controller expects the next eight rising SCLK edges to be the instruction byte of the next communication cycle. All data input to the AD9953 is registered on the rising edge of SCLK when the I/O update pin is high.



4 Port pins were used to communicate with the DDS. The four lines being SCLK, SDIO, Reset and I/O update.

Algorithm to send a bit to the DDS

1. Put data on SDIO.
2. Hold data on SDIO, make SCLK high.
3. Make SDIO and SCLK low.

Algorithm to write to a control register of the DDS

1. Send the instruction byte in the following format

INSTRUCTION BYTE

The instruction byte contains the following information:

Table 12.

MSB	D6	D5	D4	D3	D2	D1	LSB
R/Wb	X	X	A4	A3	A2	A1	A0

R/Wb—Bit 7 of the instruction byte determines whether a read or write data transfer will occur after the instruction byte write. Logic High indicates read operation. Logic 0 indicates a write operation.

X, X—Bits 6 and 5 of the instruction byte are Don't Care.

A4, A3, A2, A1, A0—Bits 4, 3, 2, 1, 0 of the instruction byte determine which register is accessed during the data transfer portion of the communication cycle.

2. Send the appropriate number of bytes of data.
3. Make I/O update high for this data to get transferred into the registers from the I/O buffers.

4.1.1 Initialization of the DDS

The DDS is initialized for the following specifications

- a) Cos wave output
- b) MSB first mode of data transfer
- c) Crystal Frequency (20 MHz) multiplied by a factor of 20 to get the system clock frequency (400 MHz)

Calculation for the Frequency Tuning Word of the DDS

$$f_o = (FTW)(f_s)/2^{32} \quad \text{with } 0 \leq FTW \leq 2^{31} \quad (1)$$

$$f_o = f_s \times \left(1 - (FTW/2^{32})\right) \quad \text{with } 2^{31} < FTW < 2^{32} - 1 \quad (2)$$

Where

f_o = Output frequency of the DDS

f_s = Frequency of the system clock (400 MHz in this case)

FTW = Frequency Tuning Word

4.2 Microcontroller code

```
#include <AT89X51.H>
```

```
#include <stdio.h>
```

```
double pow(a,b)
```

```
{
    int j=0;
    double result=1;
    if(b==0)
        return 1;
    else{
        for(j=0;j<b;j++)
```



```

        result=result*a;

    }
    return result;
}

//DDS functions
void byte_out(unsigned char byte);
void one();
void tuning_word_out(unsigned int tuning_word);
void zero();

/* output byte, MSB first */

/* output a '1' to DDS */
void one()
{
    P2=9;
    P2=0x0B;
    P2=8;

}

/* output a '0' to DDS */
void zero()
{
    P2=8;
    P2=0x0A;
    P2=8;

}

void byte_out(unsigned char byte)
{
    int i=0;
    for (i = 0; i < 8; i++)
    {
        if ((byte & 128) == 128)
            one();
        else
            zero();
        byte = byte << 1;
    }
}

```

```

//LCD functions
void delay (int a)small
{
    TR0=0;
    TF0=0;
    TMOD=0x01;
    while(a!=0)
    {
        TL0=0x00;
        TH0=0xFF;
        TR0=1;
        while(TF0==0)
        {
        }
        TR0=0;
        TF0=0;
        a=a-1;
    }
    return ;
}

void initdis (char a)small
{
    P1=a;
    P0=0x00;
    P0_2=1;
    P0_2=0;
    delay(10);
}

void dtdis (int a)small
{
    P1=a;
    P1_5=1;
    P1_4=1;
    P0=0x01;
    P0_2=1;
    P0_2=0;
    P0_0=0;
    delay(10);
    return ;
}

```

```
void chrdis (char a)small
```

```
{
    P1=a;
    P0=0x01;
    P0_2=1;
    P0_2=0;
    P0_0=0;
    delay(10);
    return ;
}
```

```
void strdis (char *str)small{
```

```
while (*str != '\0')
{
    chrdis(*str);
    ++str;
}
}
```

```
int test0(unsigned char *x) small
```

```
{
    *x=P3;
    if (P3==0xF0)
    return 0;
    return 1;
}
```

```
int ky (unsigned char y)small
```

```
{
    unsigned char
    k[]={0xBB,0xE7,0xEB,0xED,0xEE,0xD7,0xDB,0xDD,0xDE,0xB7,0xBD,0x7E,0x7D,0
    x7B,0x77,0xBE};
    int z;
    for( z=0;z<16;z++)
    {
        if(y==k[z])
        return z;
    }
}
```

```
void main(void)
```

```
{
    float unit=1;//default unit Hz
```

```

float freq=0;//default frequency (0 Hz)

//LCD variables
unsigned char a0,a1,a2,a3,a;
unsigned char* x0= &a0;
unsigned char* x1= &a1;
unsigned char* x2= &a2;
unsigned char* x3= &a3;
int number[6];
int state=0;
int p;
int count=0;
int key[]={0,1,2,3,4,5,6,7,8,9};

//DDS variables
unsigned int delta_phase1;
unsigned int delta_phase2;
unsigned int delta_phase3;
unsigned int delta_phase4;
int j=0;
float clock_in=400000000;
double delta_phase;
unsigned char b;

//LCD initialization
P0=0x00;
P1=0x00;
TR0=0,TF0=0;
TR1=0,TF1=0;
EA=0;
delay(200);
initdis(0x38);
initdis(0x0F);
initdis(0x01);
initdis(0x06);
P3=0x00;
P3=0xF0;
strdis("ENTER UNIT");

//DDS initialization
b=0x00;
/* set all pins low */

```

```

P2=0;
clock_in=400000000;

/* main program loop */

byte_out(0x00);
byte_out(0x02);
byte_out(0x00);
byte_out(0x02);
byte_out(0x42);
P2=0x0C;
delay(1);
P2=8;

byte_out(0x01);
byte_out(0x00);
byte_out(0x00);
byte_out(0xA0);
P2=0x0C;
delay(1);
P2=8;

while(1) {
    delay(100);
    if (state==0){

        if(test0(x0)==1){
            state=1;
            delay(100);
        }
        else{
            state=0;
            delay(100);
        }
    }

    if(state==1){

        if(test0(x1)==1){
            if(a0==a1){
                state=2;
                a=a1;
                a=a|0x0F;
                P3=a;
                a=P3;
            }
        }
    }
}

```

```

        P3=0xF0;
        delay(100);
    }
    else{
        state=0;
        delay (1);
    }
}
else{
    state=0;
    delay (1);
}
}

if (state==2){
    if(test0(x2)==1){
        if(a2==a1){
            state=2;
            delay (1);
        }
    }
    else{
        state=3;
        delay(1);
    }
}

if(state==3){
    if(test0(x3)==1){
        if(a2==a3){
            state=2;
            delay (1);
        }
    }
    else{
        p=ky(a);
        if(count!=6 && p<11){
            count++;
            number[count-1]=p;
        }

        if(p==0){
            dtdis('0');
            freq=freq*10;
        }
    }
}

```

```
if(p==1){  
  dtdis('1');  
  freq=freq*10+1;  
}
```

```
if(p==2){  
  dtdis('2');  
  freq=freq*10+2;  
}
```

```
if(p==3){  
  dtdis('3');  
  freq=freq*10+3;  
}
```

```
if(p==4){  
  dtdis('4');  
  freq=freq*10+4;}
```

```
if(p==5){  
  dtdis('5');  
  freq=freq*10+5;  
}
```

```
if(p==6){  
  dtdis('6');  
  freq=freq*10+6;  
}
```

```
if(p==7){  
  dtdis('7');  
  freq=freq*10+7;  
}
```

```
if(p==8){  
  dtdis('8');  
  freq=freq*10+8;  
}
```

```
if(p==9){  
  dtdis('9');  
  freq=freq*10+9;  
}
```

```
if(p==10)
```

```

chrdis('.');

if(p==11){
byte_out(0x00);
byte_out(0x00);
byte_out(0x00);
byte_out(0x02);
byte_out(0x42);

P2=0x0C;
delay(1);
P2=8;

byte_out(0x01);
byte_out(0x00);
byte_out(0x00);
byte_out(0xA0);

P2=0x0C;
delay(1);
P2=8;

byte_out(0x04);

freq=freq*unit;
if(freq>160000000){
freq=0;
}
delta_phase = (freq * pow(2, 32)) / clock_in;
delta_phase1 = delta_phase/pow(2,24);
b=(unsigned char)delta_phase1;
byte_out(b);
delta_phase2 = (delta_phase-
delta_phase1*pow(2,24))/pow(2,16);
b=(unsigned char)delta_phase2;
byte_out(b);
delta_phase3 = (delta_phase-delta_phase1*pow(2,24)-
delta_phase2*pow(2,16))/pow(2,8);
b=(unsigned char)delta_phase3;
byte_out(b);
delta_phase4 = delta_phase-delta_phase1*pow(2,24)-
delta_phase2*pow(2,16)-delta_phase3*pow(2,8);
b=(unsigned char)delta_phase4;
byte_out(b);

```



```
P2=0x0C;  
delay(1);  
P2=8;
```

```
freq=0;  
count=0;  
}
```

```
if(p==12){  
  initdis(0x01);//clr display  
  initdis(0x8E);  
  strdis("Hz");  
  initdis(0x80);  
  unit=1;  
}
```

```
if(p==13){  
  initdis(0x01);//clr display  
  initdis(0x8D);  
  strdis("KHz");  
  initdis(0x80);  
  unit=1000;  
}
```

```
if(p==14){  
  initdis(0x01);//clr display  
  initdis(0x8D);  
  strdis("MHz");  
  initdis(0x80);  
  unit=1000000;  
}  
if(p==15){  
  chrdis('a');  
}
```

```
state=0;  
delay(1);  
}
```

```
}
```

```
}
```

```
}
```

4.3 State diagram

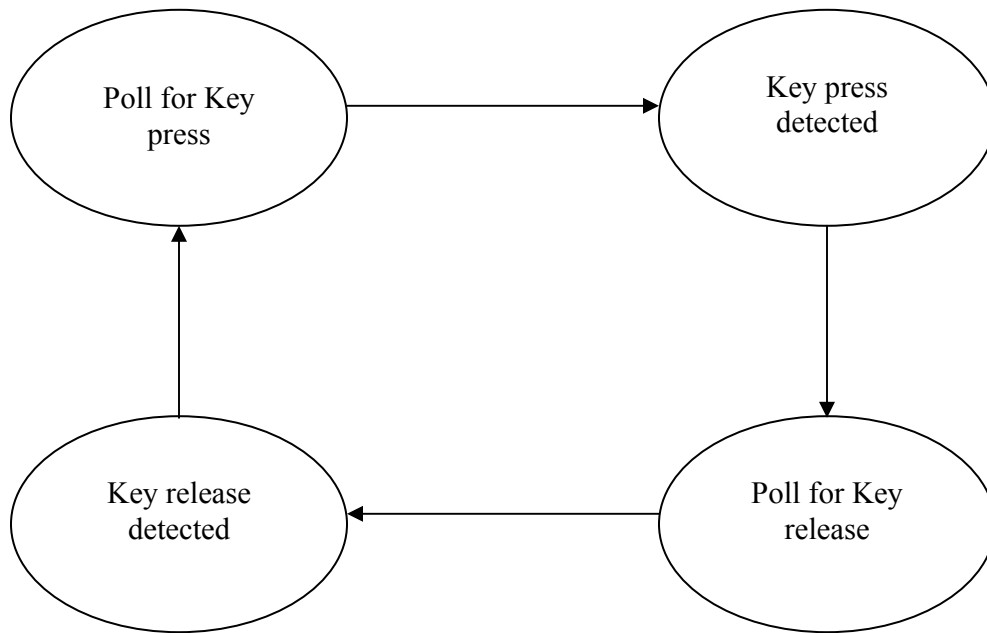


Figure 10: Keypad state diagram

5 Test Procedure and Results

The output of the DDS was observed on a Digital Storage Oscilloscope and a Spectrum analyzer. The results for frequencies from 10 Hz to 120 MHz are shown in Appendix I.

6 Discussion of the results

The output waveforms till 100 MHz were satisfactory. Distortion was observed for higher values of frequency. The manufacturers of the DDS chip suggested that up to 40% of the clock frequency can be generated. Observations of this circuit contradict this claim as all waveforms above 100 MHz were found to be distorted and this distortion could not be removed by adding a low pass filter to the circuit.

7 Suggestions for improvement

A frequency multiplier could have been attached to the output to get higher frequencies in the range of GHz.

8 Users Manual

Instructions

- 1) Connect plug to 230 Volts, 50 Hz supply.
- 2) Switch on the device using the red button in the front.

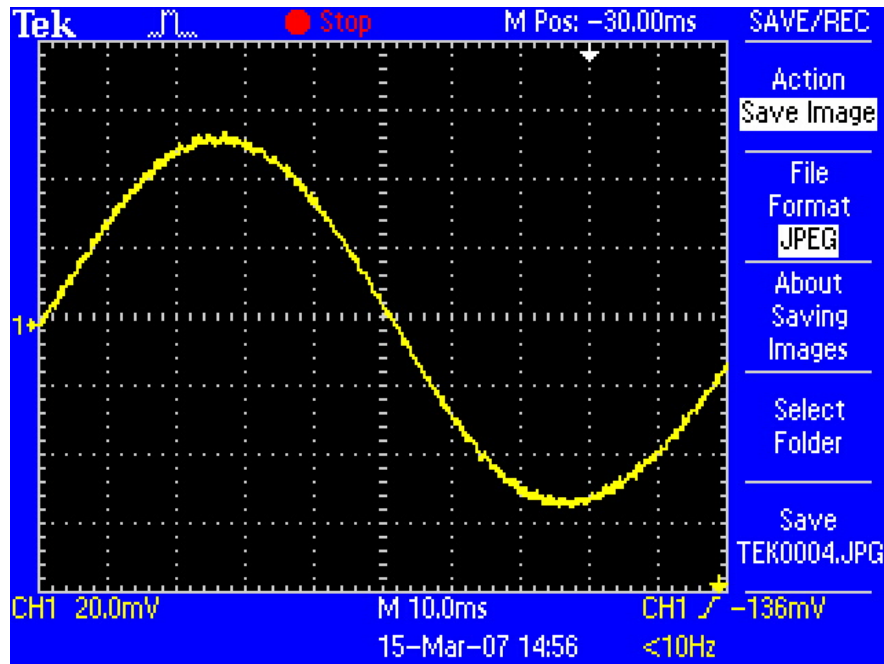
- 3) Enter the unit using keypad.
- 4) Enter the desired frequency. It should be less than 100 MHz.
- 5) Press 'Enter'.
- 6) Connect one end of a probe at the output.

9 References

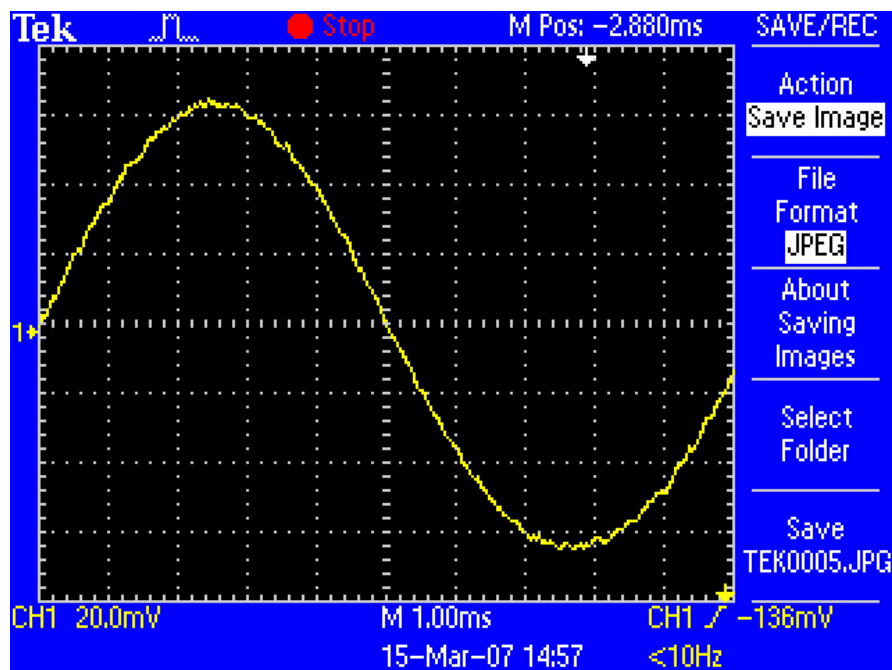
- 1) Analog Devices, AD9953 Datasheet, "400 MSPS, 14-Bit, 1.8 V CMOS, Direct Digital Synthesizer", 2007.
- 2) Atmel, AT89C51 Datasheet, "8-bit Microcontroller with 4K Bytes Flash", 2000.
- 3) Texas Instruments, LM 7805 Voltage regulator Datasheet.
- 4) Myke Predko, "LCD Interfacing Reference Page", <http://www.myke.com/lcd.htm>.
- 5) Analog devices, "A Technical Tutorial on Digital Signal Synthesis", 1999.

Appendix I

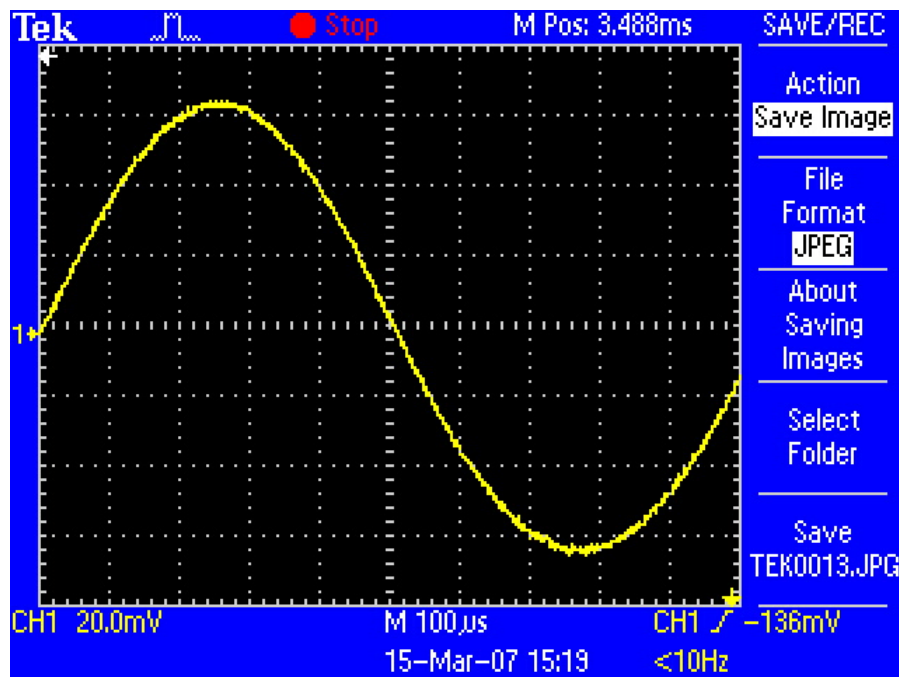
10 Hz



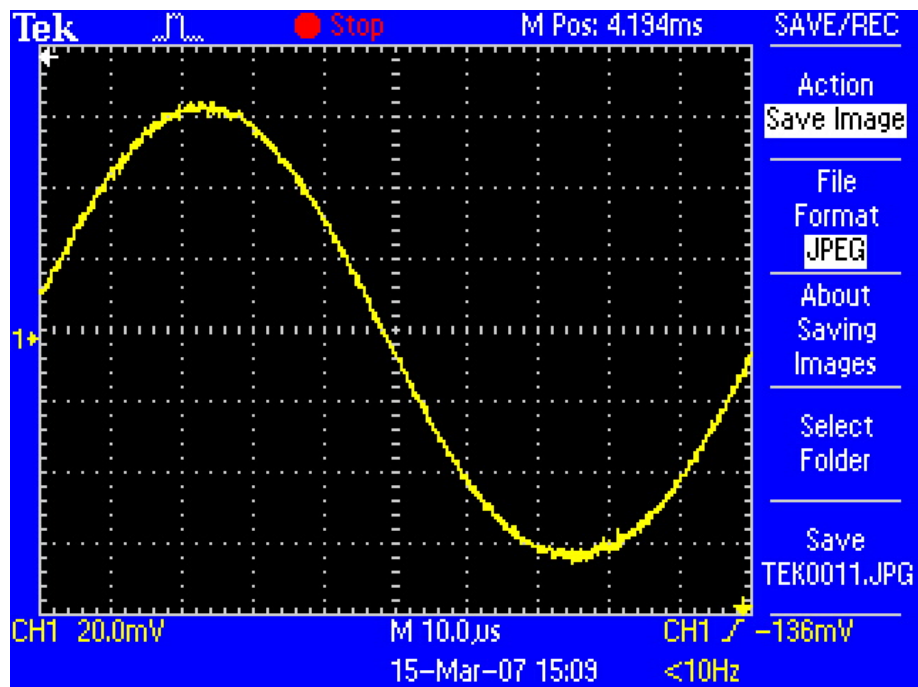
100 Hz



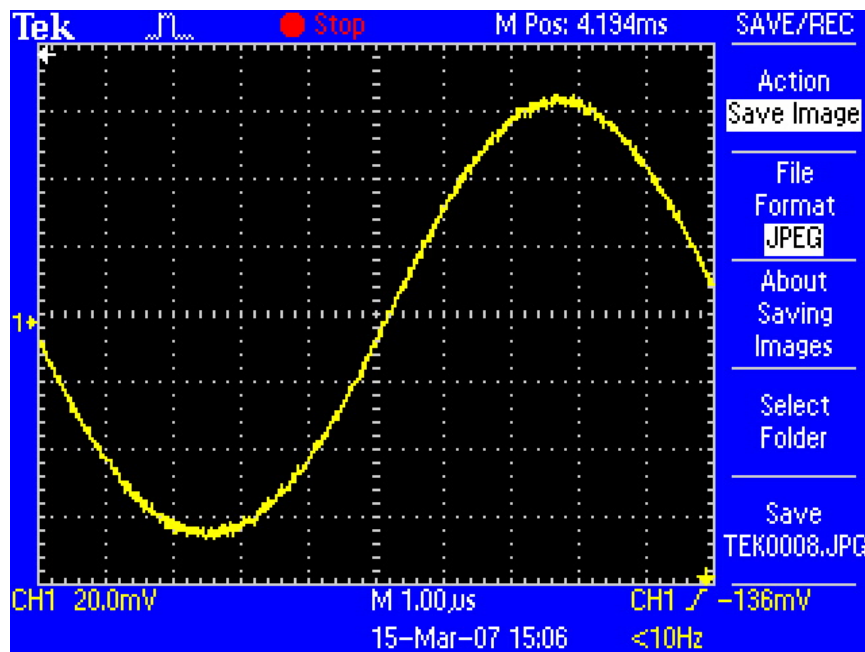
1 KHz



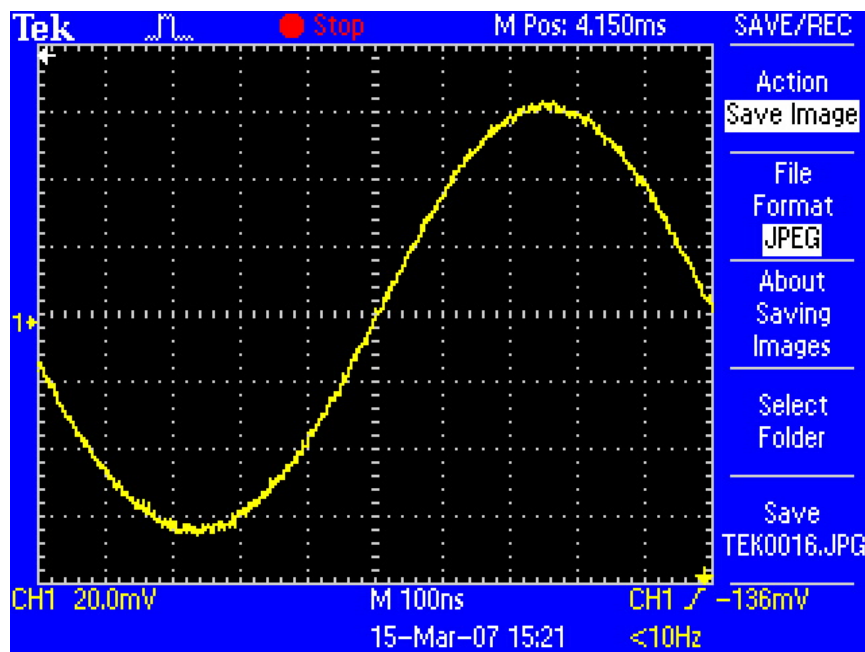
10 KHz



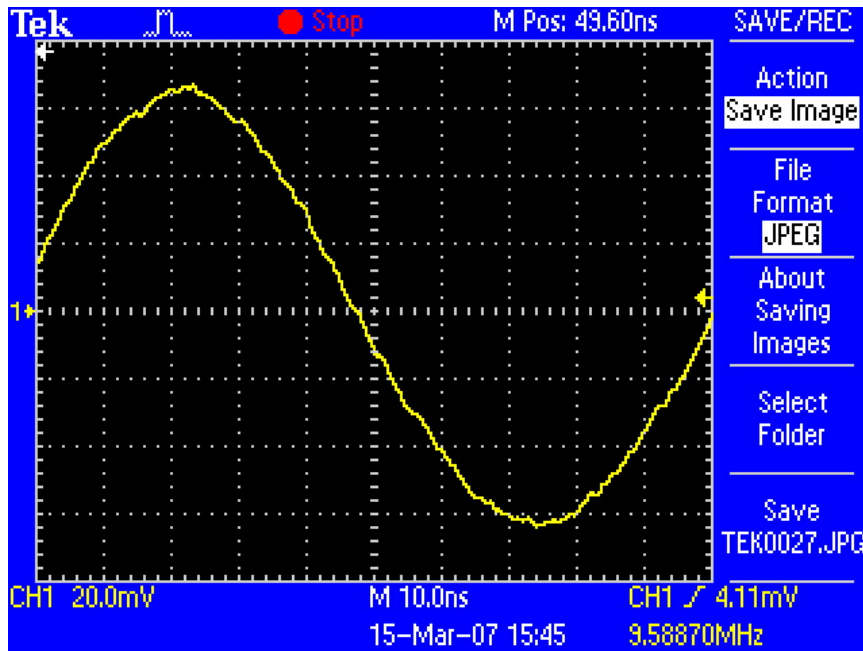
100 KHz



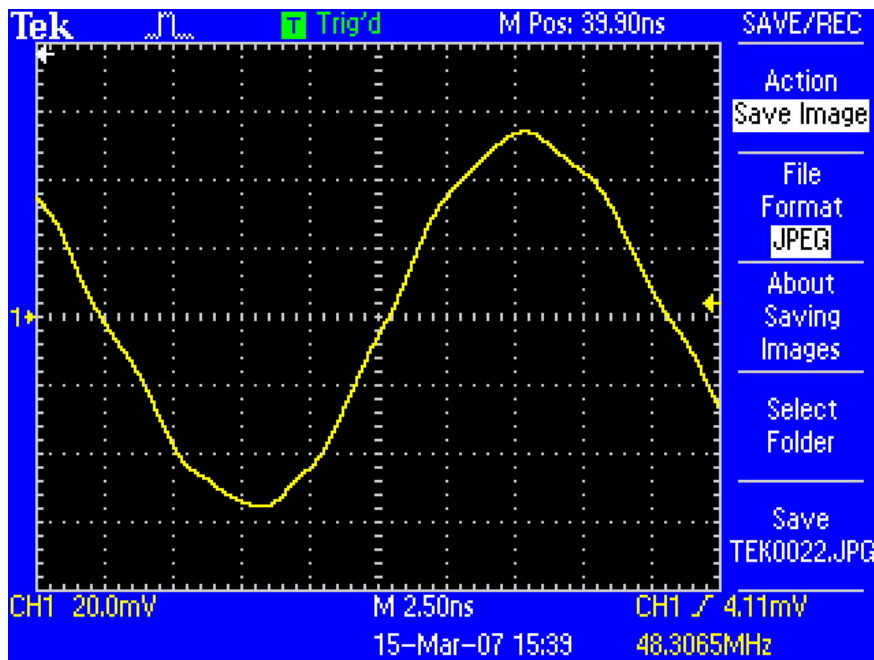
1 MHz



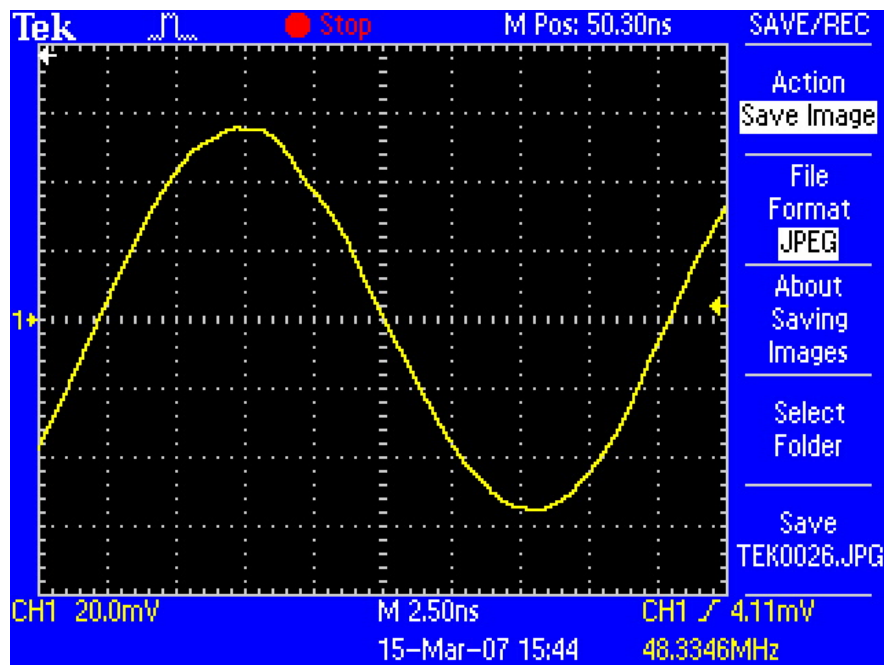
10 MHz



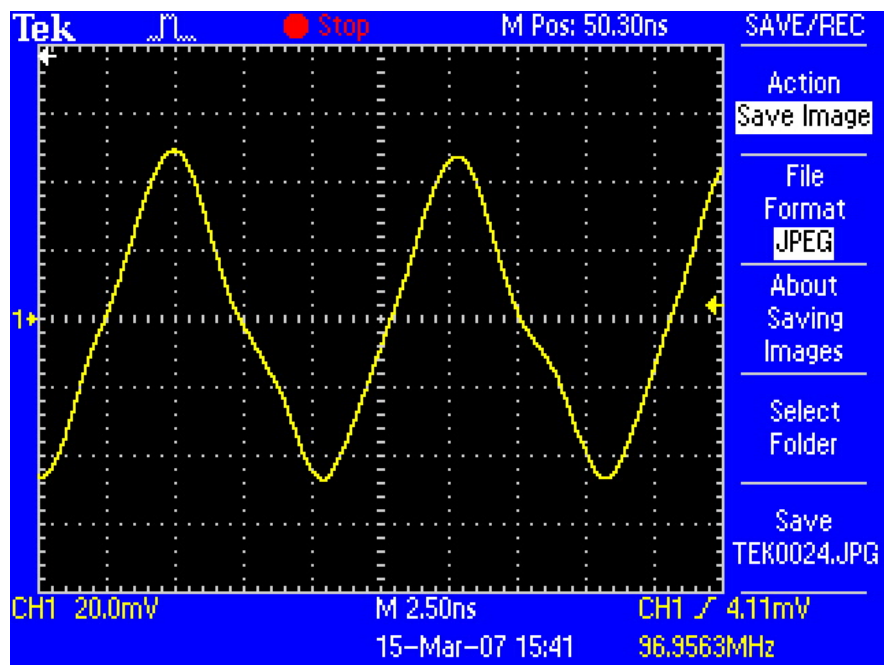
50 MHz (Without Filter)



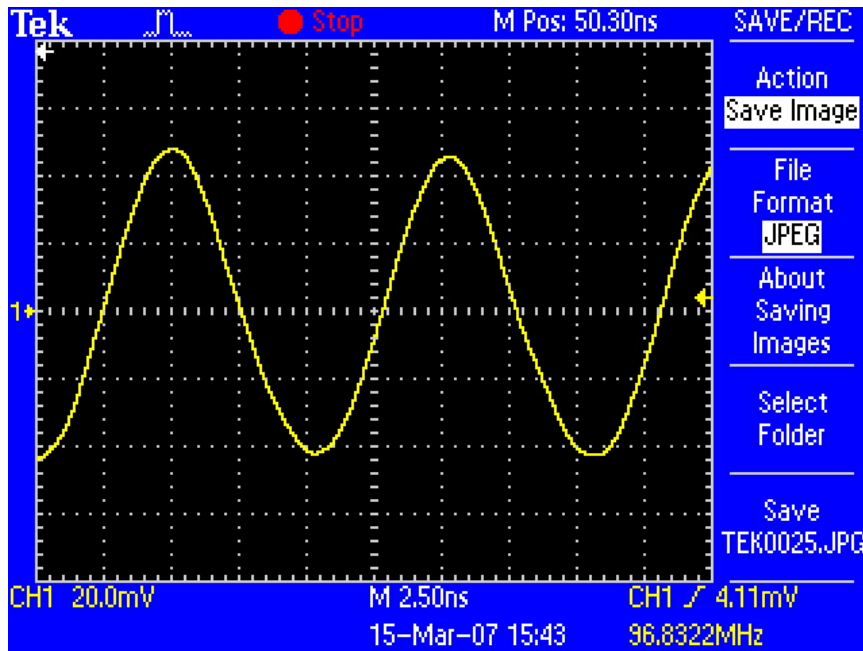
50 MHz (With Filter)



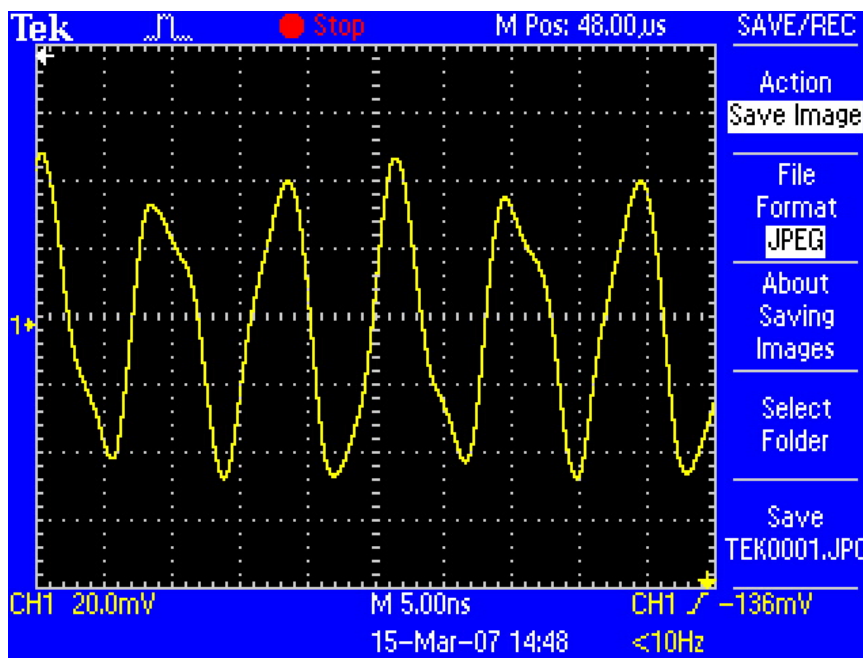
100 MHz (Without Filter)



100 MHz (With Filter)



120 MHz (Without Filter)



120 MHz (With Filter)

