

Real - Time Digital Signal Processing
Prof. Rathna G N
Department of Electrical Engineering
Indian Institute of Science - Bengaluru

Lecture - 12
Lab: Sine Generation

Welcome back to second lab of real time digital signal processing, as I was mentioning in the first lab, so anybody is ready to dirty their hand so you can start doing it. So, we will start with first in this lab sine generation.

(Refer Slide Time: 00:41)

Recap

- We discussed, how to use code composer studio for dot products and sine generation.
- In this lab, we will see different ways of sine generation using MATLAB and how to create a basic music with this

2

Rathna G N

So what is it, as a recap, we discussed how to use code composer studio and for dot products and then sine generator, what we did. Today, we will see that only 1 sine wave generation last class we have seen, so we have multiple ways of representing sine generation. So, we will be using both MATLAB and then code composer studio today, and then how to create a basic music. So, people who are music lovers, you can create your own music and then hear how it is going to come.

(Refer Slide Time: 01:15)

Direct Digital Synthesizer Technique



- Variable initialization section (lines 2–6). Remembering that the sine and cosine functions are constrained to the range ± 1 requires an amplitude scale factor, A , or the DAC's output will only use a tiny portion of the full range of $+32767$ to -32768 .
- For a constant output frequency, the calculation of the phase increment (line 9) need only be accomplished once. The calculated value of the phase increment must be $\leq \pi$ or signal aliasing will occur.
- The actual algorithm to generate the sinusoidal signal requires only three lines of code (lines 13–15), inside a "for" loop that simulates the execution of an ISR in C that is called each time a new sample arrives. These lines of code accomplish the following three tasks each time the "ISR" is called:
 - line 13: add the phase increment's value to the phase accumulator.
 - line 14: perform a modulus 2π operation to keep the phase accumulator in the range 0 to 2π .
 - line 15: calculate the system's output value by scaling the sine of the phase accumulator's value by A
- % Simulation inputs
- 2 $A = 32000$; % signal's amplitude
- $f = 1000$; % signal's frequency
- 4 phaseAccumulator = 0; % signal's initial phase
- $F_s = 48000$; % system's sample frequency
- 6 numberOfTerms = 50;
- % calculate this number of terms
- 8 % Calculated and output terms
- phaseIncrement = $2\pi \cdot f / F_s$; % calculate the phase increment
- 10
- for i = 1: numberOfTerms
- 12 % ISR's algorithm begins here
- phaseAccumulator = phaseAccumulator + phaseIncrement;
- 14 phaseAccumulator = mod (phaseAccumulator, 2π); output = $A \cdot \sin$ (phaseAccumulator)
- 16 % ISR's algorithm ends here
- end

So, the first sine generation, what we will say it as direct digital synthesizer technique. So, in this case, we will have variable initialization section. That is, we say lines 2 to 6 here. So, we will be initializing our amplitude as 32,000 signal amplitude what we call it. So, compared to this detail, on the left hand side, the code is given on the right hand side for MATLAB implementation.

And some course function are constrained lead to the range between ± 1 , which requires an amplitude scale factor of A what we call it, or the DAC output, whenever we want to output it on to DAC of our board, then we had to enhance the amplitude of it for that we have taken some amplitude equal to 32,000 in this case. So, we use the tiny portion of the full range of what we call it as $+32767$ to -32768 .

I think there should be an alarm ringing in your thing so we are using the 16 bit signed representation in this case. So, the number of bit for the fractional representation, what we will be using is 15 bits 1 for sine bit. So, for a constant output frequency, the calculation of the phase increment that is what is shown in line here. Phase enhancement, which is given by the equation $2 \times \pi \times \text{frequency component}$ what it is chosen F as 1000.

That is 1 kilohertz here, divided by the sampling frequency. In this case, sampling frequency F_s is as assumed as 48 kilohertz. So, we will be calculating the phase increment based on this unit and for a constant output frequency, the calculation of phase increment as we said, in line 9. The calculated value of the phase increment must be usually we restrict between $-\pi$ to π here, that is what it will be $\leq \pi$ or signal aliasing is going to happen.

So, for that reason, so you will be seeing the actual algorithm to generate our sinusoidal signal requires only 3 lines of code that what you are seeing line 13 to 15 basically is the code to generate our sine wave in this case. The rest of them are initialization functions. So, inside a for loop, and then simulate the execution of interrupt service routine in C if we are writing it will consider this writing of interrupt service routine in the next lab class, that is called each time a new sample is going to arrive.

So, these lines of code accomplish the following 3 tasks each time an interrupt service routine is going to be called. What is the first one? That is line 13 basically, so you are adding the phase increments value to the phase accumulator and then in the line 14, you will be calculating, perform a modulus basically, that is a 2π operation to keep the phase accumulator in the range 0 to 2π .

Either you can have $-\pi$ to π or 0 to 2π is the range what we are going to take it here in this case, we will be seeing that if it is going to cross 2π , then output is going to be modulus value what we will be taking it. In the line 15, we will calculate the system's output value by scaling the sine of the phase accumulator value by A. So, that is what we are going to do in the thing, just we will see that how this is going to run.

(Video Starts: 05:51)

So, we will run the MATLAB code in this case. So, we have you will be seeing the code sine generation dot m using the DDS method. So, you will be seeing that your magnitude A in this case I can assume 1 or there it was 32,000 we can vary we will see it in a while and then my frequency component what I want is 1000, 1 kilohertz frequency what I want to generate this is a variable so, you can play around with it.

And then you will be initializing your accumulator that is phase accumulator as 0 and sampling frequency 48 kilohertz. So, this also you can vary depending on your requirement and then your this thing number of terms what in this case we are generating 50 this also if you want more waves basically you can increase that, that is what it says is calculate this number of terms basically.

And then calculated and then output terms are going to be your phase increment this is the increment what we are providing it. So, the phase increment and this is the 1 loop what it is

going to be implemented. So, for $i = 1$ to number of terms, because we need 50 terms. So, we are accomplishing this one. So, instead of putting it on that DAC, here, what I am doing is I am going to store the output in a loop, basically array, which is given by this equation.

And then if you have to have the interrupt service routine algorithm for the sine loop, we will be providing it here. And then this output will be going through the interrupt service routine to our DAC output in the real time scenarios. So, here at the end of our routine, we can plot and see whether I am able to generate using this function. So, what MATLAB does is any of the MATLAB version, you can take it in this case I am using MATLAB 2020 B version.

So, you can use any one of them. So, when I run this code, so, you will be seeing that so, my sine generation is running. So, the 50 sample is the output what I have given, so, you will be seeing that you are getting in complete 1 period of 50 samples here. So, by varying this value, this is one way of generating our sine wave. So, if I increase my magnitude to whatever was given in that, I can do it as 32,000 so I can rerun my code.

So, you will be seeing that the magnitude which was 1 in the earlier case, it has gone up to 10^4 in this case. So, if you want to change your sampling frequency and the frequency part of it, so you can vary here, F can be I can give it as here now for 2000 hertz, basically, and sampling frequency I can keep it at 8000 hertz also because I have been meeting my whatever the sampling theorem says twice that of the maximum frequency so F_n is 2000 which is greater than or equal to twice that effect. So, I am a little more than the thing.

So, I can rerun this code as you will be seeing it so you are able to get 2000 hertz whatever. So, how do you know that you have got the 2000 hertz signal, So, you have to wait for FFT class or you can run if you are comfortable with MATLAB you can put the FFT magnitude if you calculate the thing you will be getting the as we can see that so, we can rerun the thing. So, you will be seeing that my FFT function has, what is it error in the thing.

So, you can go back and then correct it and then see that you will be getting your magnitude function. So, what is the frequency you have represented so, coming back to my slide what is the other method of running or generating our sine wave. So, this was the first one that is direct digital synthesizer what we have taken.

(Video Ends: 11:00)

(Refer Slide Time: 11:06)




Table Lookup Technique

- Variable initialization section (lines 2-4). These lines of code establish the variable signal that stores the required values of the output signal and the integer variable index that is used to access the different storage locations of signal.
- Period determination (line 7). This line of code determines the period of the signal based on the length of the variable signal.
- The actual algorithm to generate the sinusoidal signal requires only five lines of code (lines 11-15). These lines of code accomplish the following three tasks each time the ISR is called:
 - line 11-13: performs a modulus N operation to keep index in the range 1 to N. Remember that, unlike C/C++, MATLAB array indices start at 1 instead of 0.
 - line 14: calculates the system's output value by selecting the appropriate index of signal.
 - line 15: increments the integer variable index

- 1. % Simulation inputs
- signal = [32000 0 -32000 0] ; % cosine signal values (Fs/4 case)
- 3. index = 1; % used to lookup the signal value
- numberOfTerms = 20; % calculate this number of terms
- 5.
- % Calculated and output terms
- 7. N = length (signal) ; % signal period
- 9. for i = 1: numberOfTerms
- % ISR's algorithm begins here
- 11. if (index >= (N + 1))
- index = 1;
- 13. end
- output = signal (index)
- 15. index = index + 1;
- 1 % ISR's algorithm ends here
- 17. end

4

Rathna G N

Next one can be using a table lookup technique. So, how do you generate your table so, as you can see in the right hand side here, so, you have the signal represented as 32,000 0 - 32,000 and then 0 that is we call it as cosine signal values. So, what you have chosen $F_s/4$ case is what we have taken the thing so, coming with the thing it is a variable initialization section that is what lines 2 to 4 on the right hand side what it is going to be.

So, what it will take and then we will establish the code and then variable signal that stores the required values of the output signal and then you will be reusing them. So, what is that we will be doing the period determination from the line 7 as you can see it is n is equal to length of signal what you have taken the thing. So, we will be period determination is done by this code and then actual algorithm to generate our continuous sinusoidal signal is again lies from 11 to 15. So, you are checking whether index has increased to $n + 1$.

So, what the range we want is 1 to n. So, length of the signal so, if it is exceeded $n + 1$, then you will be starting index again from 1. So, this will be if statement what we will be using it and then my for loop is going to end here and then output is going to be signal of index and then index you will be incrementing it by 1. So, your interrupt service routine can be written here, and then we will be ending the code of for loop basically.

(Video Starts: 13:22)


So, we will go back and then see again in MATLAB whether it is running correctly or not. So, I have chosen this so you will be seeing that signal is represented by these values and index is 1 and number of terms also have chosen in this case 50 and then n is going to be length of

signal whatever you have chosen and then for $i = 1$ to number of terms. So, this code is going to be repeated.

If n is greater than or index is $> n$ for the $n + 1$ index will be reset to 1 and output again putting it in an array so that we can plot it and then see whether we are getting it correctly so we will run the code. So, you will be seeing that it has generated our sine wave as you can see in this case, so this is how second method what you can run with a lookup table. So, it is going to help us that you have pre calculated and then kept the thing.

(Video Ends: 14:40)

(Refer Slide Time: 14:46)



Musical Note (Sa re ga ma pa da ni Sa)

- clc; clear all; close all;
- Fs = 8000;
- t=1:Fs;
- sa(t)=sin(2*pi*440*t/Fs);
- y=sin(2*pi*440*t/Fs);
- re(t)=sin(2*pi*494*t/Fs);
- y2 = re(t);
- ga(t)=sin(2*pi*554*t/Fs);
- y3= ga(t);
- ma(t)=sin(2*pi*587*t/Fs);
- y4 = ma(t);

- pa(t)=sin(2*pi*660*t/Fs);
- y5=pa(t);
- da(t)=sin(2*pi*698*t/Fs);
- y6=da(t);
- ni(t)=sin(2*pi*784*t/Fs);
- y7 = ni(t);
- sahi(t)=sin(2*pi*880*t/Fs);
- y8=sahi(t);
- SatoNi = horzcat(y2,y3,y4,y5,y6,y7,y8);
- sound(SatoNi);

5
Rathna G N

So, it depends on how much memory what you need it. So, just as I was telling, so to generate using the sine concept, can we generate a musical note? In this case basic of music what I have taken it is Carnatic music whatever you call it, it is sa re ga ma pa da ni sahi, what will be generating it. So, what is the thing is going to happen, we will set the sampling frequency as 8000 hertz, and then t will be varying between 1 and then F_s and my sa, note, what we have chosen, the basic note in this case is 440 hertz.

So, that is what we put here, and then we will be generating the notes. The next note is as you can see is 494, either you can use the equation whatever the musical literature gives and then do it or if you have pre calculated I can select the frequency part of it for all the notes. So, in this case ga is going with 554 and then ma will be going with 587 and then pa with 660 and da with 698 and then ni will be 784.

And then the last note, as you know, it will be twice that of the first whatever base note what you have taken the thing, which is going to be 880 hertz. So, and then what we do is, we will be concatenating them, that is, all of them will be horizontal concatenation is going to happen. So, we have the function, horizontal cat actually, all these notes, and then we will be playing them, so just hold on a while.

(Video Starts: 16:35)


So, we will see how our music is going to come out of it. So, this is the code written. So, if you want you can treat it nicely and then say that with the comments what you can put the thing. So, here is the basic thing what I wanted to show in this lecture. So, go to the editor, so you will be running this. So, you will be hearing it and then you have to comment whether you have heard the sa re ga ma pa da ni sahi correctly or not.

Want to hear it once again? I will play the thing. So, if you have your notes, basically, some of the students played their own happy birthday song, and then our national anthem so which we can play when everybody is around. So, you can go and then write your own musical compositions and then generate it and hear that you are getting it correctly.

(Video Ends: 17:51)

(Refer Slide Time: 18:04)

Polynomial Approximation (GNU math library)



- This is the approach used by the [GNU scientific library](#) which stores the first 11 coefficients of the approximation.
- Horner's form is used to minimize the number of operations required
 - $a_{10}x^{10} + a_9x^9 + a_8x^8 + \dots + a_0$
 - $= (\dots((a_{10}x + a_9)x + a_8)x \dots) + a_0$
- To generate a sinusoid of frequency f_0 , need only repeat a few steps for each sample
- Calculate $\cos(\theta)$ via the approximation above and send the value to the DAC
- Increment θ by $\omega_0 = 2\pi f_0 / f_s$
- Check if θ exceeds 2π . If it does, subtract 2π to prevent overflow or loss of precision.
- This method produces a very accurate signal, but requires roughly 20 multiplications and 20 additions per sample in addition to the storage of the coefficients.
- It is also extremely flexible, since it allows us to change the frequency as we please by simply changing the amount at which we increment θ .

6
Rathna G N

So, the other way of doing it is whether I can do sine generation using my polynomial approximation, we call it as GNU math library here. So, this is the approach using the scientific library, which stores the first 11 coefficients of the approximation. So, what it calls is horner's form is used, what is that horner's form we will see it once now, the number of operations required in this case is $A^{10}10 + A^9 + A^8 + \dots$.

So, this will be A^0 so, you will be seeing that how the horner's form is going to be you will be doing A^{10} into your this thing, what is it $0x$, x is the input value what you have to give it plus $A^9x + A^8x$. So, so on what you will be putting the thing so that your power calculation is going to be reduced plus A^0 what you will be doing. So, to generate sinusoidal frequency f_0 , we need only few steps for each sample.

So, you will be calculating the $\cos \theta$ via the approximation above and then send the value to the DAC and then increment θ by $\omega_0 = 2\pi f_0/f_s$ and then check if θ is going to exceed 2π basically, then 2π if it does subtract 2π to prevent the overflow or loss precision. I want you to write the code for this in MATLAB and see whether you have got the sine wave generated correctly using this polynomial approximation.

So, what this says is this method produces a very accurate signal, but requires roughly 20 multiplications and 20 additions, as you can see in the bracket there per sample in addition to the storage of the coefficients, so, you have to store all the coefficients and then run them. And then it is also extremely flexible since it allows us to change the frequency as we please by simply changing the amount at which we increment θ .

So, by doing the θ increment varying, so you will be able to get whatever frequency you want to generate.

(Refer Slide Time: 20:56)



So, coming to the next one, some of the lab thing component, we will be seeing that how to generate real time sine wave generation and FIR filters. Hope you enjoyed the sine wave

generation and then music generation in MATLAB. So, the same thing what we will see in the next lab class on DSP processor, so you can play with it with whatever music you love. Thank you. So, see you in the next lab session.