University of Tehran
College of Engineering
School of Electrical and Computer Engineering

Real-time Digital Signal Processing Laboratory

Dr.Shah-Mansouri

Lab 6

Soroush Mesforush Mashhad

SN:810198472

Dey 01

Contents

Abstract

In this lab we firstly studied the difference between the floating point and fixed point representation via the online class, then we shall implement the fixed and floating point versions of the Cosine function.

1 Floating Point implementation of $\cos(\theta)$

We know we implement the floating point version of the Cosine function using its Maclaurin series as depicted below.

$$\cos(\theta) = 1 - \frac{1}{2!}\theta^2 + \frac{1}{4!}\theta^4 - \frac{1}{6!}\theta^5 + \dots$$

1.1 First Implementation

In this implementation we use the first code given in the lab manual named $fcos1$ which is depicted as follows.

```
float fcos1(float x)
{
  float fcoef[4] = { 1.0,-1 / 2.0,1.0 / (2.0 * 3.0 * 4.0),-1.0 /
    (2.0 * 3.0 * 4.0 * 5.0 * 6.0) };
  float out;
  out = fcoef[0];
  out += fcoef[1] * x * x;
  out += fcoef[2] * x * x * x * x;
  out += fcoef[3] * x * x * x * x * x * x;

  return out;
}
```

The output is depicted as below.

```
First Implementation (fcos1) :
cos(0.000000) = 1.000000
cos(0.100000) = 0.995004
cos(-0.800000) = 0.696703
cos(19.373156) = -67746.656250
cos(-0.523592) = 0.866029
cos(1.570796) = -0.000894
cos(3.141592) = -1.211352
```

Figure 1: First Implementation of Cosine floating point

1.2 Second Implementation

In this implementation we use the second code given in the lab manual named $fcos2$, in this code we use the powers of the created θs each time to reduce computational cost by reducing the number of multiplications. The code is depicted as follows.

```
float fcos2(float x)
{
  float fcoef[4] = { 1.0,-1 / 2.0,1.0 / (2.0 * 3.0 * 4.0),-1.0 /
    (2.0 * 3.0 * 4.0 * 5.0 * 6.0) };
  float out, x2;
  x2 = x * x;
  out = x2 * fcoef[3];
  out = (out + fcoef[2]) * x2;
  out = (out + fcoef[1]) * x2;
  out += fcoef[0];
  return out;
}
```

The output is depicted as below.

```
Second Implementation (fcos2):
cos(0.000000) = 1.000000
cos(0.100000) = 0.995004
cos(-0.800000) = 0.696703
cos(19.373156) = -67746.656250
cos(-0.523592) = 0.866029
cos(1.570796) = -0.000894
cos(3.141592) = -1.211353
```

Figure 2: Second Implementation of Cosine floating point

Query: Why shouldn't the argument of $\cos(\theta)$ be large?

Due to the fact that the Maclaurin series which we have used is valid around $\theta = 0$, if we give it points which are far from this point, the estimation is

larger and therefore the error shall be greater, in other words this functions works good for angles for which their remainder after division to 2π is small.

2 Fixed Point implementation of $\cos(\theta)$

Here we shall simulate $\cos(\theta)$ in a fixed point manner, to do so we utilize the following code.

```
short icos(short x)
{
  short iCoef[4] = { (short)(UNITQ15),(short)(-(UNITQ15 / 2.0)),
  (short)(UNITQ15 / (2.0 * 3.0 * 4.0)),
  (short)(-(UNITQ15 / (2.0 * 3.0 * 4.0 * 5.0 * 6.0))) };
  long cosine, z;
  short x2;
  z = (long)(x * x);
  x2 = (short)(z >> 15);
  cosine = (long)iCoef[3] * x2;
  cosine = cosine >> 13;
  cosine = (cosine + (long)iCoef[2]) * x2;
  cosine = cosine >> 13;
  cosine = (cosine + (long)iCoef[1]) * x2;
  cosine = cosine >> 13;
  cosine = cosine + (long)iCoef[0];
  return((short)cosine);
}
```

The output is depicted below.

```
iFixed Point (icos) :
cos(0.000000) = 0.999969
cos(0.100000) = 0.995026
cos(-0.800000) = 0.696716
cos(19.373156) = 0.809845
cos(-0.523592) = 0.866028
cos(1.570796) = -0.000824
cos(3.141592) = 0.653534
```

Figure 3: Fixed Point implementation of Cosine

Query: Why has the variable *cosine* been shifted 13 bits to the right?

First of all, we must imply that the input to the **icos** function is in the **Q14** form, when it is multiplied by itself, it takes the **Q28** form.

We know that the format of our output should be **Q15** because as we know $-1 \leq \cos(\theta) \leq 1$, the x_2 variable after being shifted 15 units to the right takes the **Q13** format, multiplying this in something in the **Q15** format yields to the **Q28** format, for this reason we shift the *cosine* variable 13 units to the right to reach the **Q15** format.

Helper functions

It is important to note that to perform the needed conversions to the necessary formats I utilized the following functions accordingly.

Function 1

```
float FloatMaker(short input, int frac_bit) {
  return ((float)input / (float)(1 << frac_bit));
}
```

Function 2

```
short FixMaker(float input, int frac_bit) {
  return (short)(input * (1 << frac_bit));
}
```

Next we shall go on to fill out the needed tables.

θ	$\cos(\theta)$	$f\cos 1(\theta)$	$f\cos 2(\theta)$	$i\cos(\theta)$
0	1	1	1	0.999969
0.1	0.995	0.995004	0.995004	0.995025
-0.8	0.6967	0.696703	0.696703	0.696716
19.373155	0.866	-67746.656250	-67746.656250	0.809845
-0.523592	0.866	0.866029	0.866029	0.866028
1.5707963	0	-0.000894	-0.000894	-0.000824
3.141592	-1	-1.211352	-1.211352	0.653534

Here this lab is concluded.

References

[1] Vahid Shah-Mansouri, *Real-time Digital Signal Processing Laboratory lab notes, Fall 01*

[2] Mohammad Ali Akhaee, *Digital Signal Processing lecture notes, Spring 01*