|  |  |
| --- | --- |
|  | School of Engineering |

|  |  |  |
| --- | --- | --- |
| **ELC 411-02** | **Embedded Systems** | **Fall 2019** |

**LAB REPORT**

**Design Assignment 3: Interrupts & Timers**



**Date: November 15th, 2019**

**Submitted by: Jordan Sinoway, Condor Gao**

**Instructor: Dr. Larry Pearlstein**

**Numerical Sine Wave Oscillator Plots**

**Explanation:**

**What is the range of values? Is this what you would expect? What is the minimum number of integer bits that could be used, and would the total number of bits for fixed point be in that case?**

N = 15: The range of the values is between 1 and -1, which is expected because we are right shifting by 215 bits. The minimum number of integers is 1, because you only need to represent numbers between 1 and -1, and the total number of bits for fixed point in this case is 17 (15 + 1 + 1 sign bit).

N = 8: The range of the values is between 1 and -1, which is expected because we are right shifting by 28 bits. The minimum number of integers is 1, because you only need to represent numbers between 1 and -1, and the total number of bits for fixed point in this case is 10 (8 + 1 + 1 sign bit).

N = 6: The range of the values is between 1 and -1, which is expected because we are right shifting by 26 bits. The minimum number of integers is 1, because you only need to represent numbers between 1 and -1, and the total number of bits for fixed point in this case is 8 (6 + 1 + 1 sign bit).

N = 3: The range of the values is between 1 and -1, which is expected because we are right shifting by 23 bits. The minimum number of integers is 1, because you only need to represent numbers between 1 and -1, and the total number of bits for fixed point in this case is 5 (3 + 1 + 1 sign bit).

**Code:**

#include "project.h"

#include <stdio.h>

//#define FLOATING\_VER

#define FIXED\_N (15)

#define FIXED\_M (32-FIXED\_N-1)

#define FIXED\_ONE (1<<FIXED\_N)

#define FIXED\_ONE\_HALF (FIXED\_ONE >> 1)

#define SQR(x) ((x) \* (x))

// Define w0 as PI/100

// cos(w0) ~ 1.0

// sin(w0) ~ w0, small angle approximation

#define COS\_0 (FIXED\_ONE)

#define SIN\_0 ( (int) (FIXED\_ONE \* 3.1415926535897 / 100 + 0.5))

#define DCOS\_0 (1.0)

#define DSIN\_0 (2 \* 3.1415926535897 / 100)

#ifdef FLOATING\_VER

double dx\_real = 1.0;

double dx\_imag = 0.0;

double drsq;

double disq;

#else

int32\_t x\_real=FIXED\_ONE; // Initial value of x (complex number) is 1+j0, in fixed point

int32\_t x\_imag=0; // Initial value of x (complex number) is 1+j0, in fixed point

int32\_t rsq; // Holds square of real component

int32\_t isq; // Holds square of imag component

#endif

int32\_t sgn\_real; // +/-1, based on x\_real > 0 or < 0

int32\_t sgn\_imag; // +/-1, based on x\_imag > 0 or < 0

int main(void)

{

int i;

char msg\_str[1024];

CyGlobalIntEnable; /\* Enable global interrupts. \*/

UART\_Start();

for(i = 0; i < 400; ++i)

{

#ifdef FLOATING\_VER

dx\_real = dx\_real\*DCOS\_0 - dx\_imag\*DSIN\_0;

dx\_imag = dx\_real\*DSIN\_0 + dx\_imag\*DCOS\_0;

sgn\_real = dx\_real >= 0 ? 1 : -1; // signum of x\_real

sgn\_imag = dx\_imag >= 0 ? 1 : -1; // signum of x\_imag

drsq = SQR(dx\_real);

disq = SQR(dx\_imag);

if (drsq+disq > 1.0)

{

dx\_real -= 0.001 \* sgn\_real;

dx\_imag -= 0.001 \* sgn\_imag;

}

else

{

dx\_real += 0.001 \* sgn\_real;

dx\_imag += 0.001 \* sgn\_imag;

}

sprintf( msg\_str, "%5d\t%5d\n\r", (int) (dx\_real\*1000), (int) (dx\_imag\*1000) );

UART\_PutString(msg\_str);

#else

x\_real = x\_real\*COS\_0 - x\_imag\*SIN\_0;//x\_real << FIXED\_N; // QM.N \* QM.N -> Q2M.2N

x\_real = (x\_real + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

x\_imag = x\_real\*SIN\_0 + x\_imag\*COS\_0; // QM.N \* QM.N -> Q2M.2N

x\_imag = (x\_imag + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

sgn\_real = x\_real >= 0 ? 1 : -1; // signum of x\_real

sgn\_imag = x\_imag >= 0 ? 1 : -1; // signum of x\_imag

rsq = SQR(x\_real); // QM.N \* QM.N -> Q2M.2N

rsq = (rsq + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

isq = SQR(x\_imag); // QM.N \* QM.N -> Q2M.2N

isq = (isq + FIXED\_ONE\_HALF) >> FIXED\_N; // convert back to QM.N, with rounding

// The goal is for 'x' to represent a value on the unit circle, i.e. complex magnitue

// should be unity. Because of quantization error (which will even occur with double

// precision, need some way of avoiding collapsing to zero, or growing to infinity

//

// My quick and dirty solution is to push both real and imaginary part down toward

// zero a smidgen if the magnitude is greater than one, and away from zero otherwise

if (rsq+isq > FIXED\_ONE)

{

x\_real -= sgn\_real;

x\_imag -= sgn\_imag;

}

else

{

x\_real += sgn\_real;

x\_imag += sgn\_imag;

}

sprintf( msg\_str, "%5d\t%5d\n\r", (int) (x\_real), (int) (x\_imag) );

UART\_PutString(msg\_str);

#endif

}

}