References

[1] Lopez M, Gerstlauer A, Avila A, & Martinez-Chapa S. (2011). A programmable and configurable multi port System on Chip for stimulating electrokinetically drive microfluidic devices (8361-8364). IEE Conference publications.

[2] Lopez M, Hernandez M. Gonzales H. Martinez S. (2013, March). An electric stimulation system for electrokinetic particle manipulation in microfluidic devices. Review of Scientific Instruments.

[3] Jupiter, XT. (2006). Synopsys, Top-Down Hierarchical Flow, User Guide, Version Y-2006.06.

[4] Gascoyne P, & Vykonkal J. (Jan 2004). Dielectrophoresis-based sample handling in general-purpose programmable diagnostic instrument (Vol 92, pp 22-42). IEEE Proceedings.

[5] Manaresi N, Romani A, Medoro G, Altomare L, Leonardi A, Tartagni M, & R Guerrieri. (2003). A CMOS Chip for Individual Cell Manipulation and Detection. (Vol. 38, pp. 2297-2305). IEEE Journal of Solid-State Circuits

[6] Tierney J, Rader C & Gold B. (1971). A digital frequency synthesizer. (Vol 19, pp 48-57). IEEE Transactions on Audio and Electroacustics.

[7] Choi Y, Kim Y, Im M, Kim B, Yun K & Yoon E. (2006). Three Dimensional Electrode Structure Controlled by Dielectrophoresis for Flow-Through Micro Electroporation System. (pp 466-469). IEEE International Conference on Micro Electro Mechanical Systems.

[8] Chang F, Lee Y & Chiu Ch. (2008). Multiple Electrodes Arrayed Dielectrophoretic Chip with Application on Micro-Bead Manipulation. IEEE Proceedings.

[9] Ibrahim M, Elsayed F, Ghallab Y & Badawy W. (2009). An Electric Field Array Microsystem for Lab-on-Chip and Biomedical Analysis. (pp 89-92). IEEE Conference on Biomedical Circuits and Systems.

[10] Standard library SRAM Generator, from Artisan, User Manuel, revision ug_2004q1v0.

[11] Standard library 0.13um - 0.25 um ROM Generator, from Artisan, User Manuel, revision ug_2004q3v1.

[12] ASIC Design Flow Tutorial, Nano-Electronics and Computing Research Center, San Francisco State University.

[13] OpenCores. (2015). OpenCores. Recuperado el 2015, de http://opencores.org/

[14] Design Compiler User Guide, Synopsys, Version C-2009.06.

[15] Library Data Preparation for IC Compiler, User Guide, Synopsys, Version D-2010.03.

[16] Li H, Yanan Z, Akin D & Bashir R. (2005). Characterization and modeling of a microfluidic dielectrophoresis filter for biological species. (Vol 14. pp 103-112). IEEE Journal of Microelectromechanical Systems.

[17] Yuk K, Mc Conaghy C, Gascoyne P, Schwartz J, Vykoukal J & Andrews C. (2007). A High-Voltage SOI CMOS Exciter Chip for a Programmable Fluidic Processor System Current", K.W.; Biomedical Circuits and Systems. (Vol 1. pp 105-115). IEEE Transactions on Biomedical Circuits and Systems.

[18] Rosa C, Tilley P, Fox J & Kaler K. (October 2008). Microfluidic Device for Dielectrophoresis Manipulation and Electrodisruption of Respiratory Pathogen Bordetella pertussis. (pp 2426-2432). IEEE Transactions on Biomedical Engineering.

[19] Villemejane J, Mottet G, Francais O, Pioufle B, Woytasik M & Dufour-Gergam E. (2010). Nanomanipulation of Living Cells on a Chip Using Electric Field. (pp 229-232). IEEE International Symposium in Electronic Design, Test and Application.

Appendixes

This section has been constructed for easy access to the most relevant information about the developed work. Here is found the application program for the standard and extended versions, an illustration of the signal superposition methodology, the content of the base, temporary, and output data tables, a summary of the user interface, a compact description of the SoC design flow, and the final SoC parameters.

A.1 Application Program: Standard Version

The standard version has been developed for the SoC design. It can be stored in in-chip ROM or uploaded to chip RAM at boot time.

/* Name: boardv2.c

Author: Martha Lopez

Version: Board_Extended_v2, 256 data sine samples, buffer table OK, all frequencies, three operation modes

Copyright: (C) Copyright

Description: Board version, standard functionality, sine, saw-tooth, triangle */

// include files

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

//definitions and declarations

#define Pi 3.14159265358979323846264338327

static unsigned int sinedatint[256] =

{ 127, 130, 133, 136, 139, 142, 145, 148, 151, 154, 157, 160, 163, 166, 169, 172, 175, 178, 181, 184, 186, 189, 192, 194, 197, 200, 202, 205, 207, 209, 212, 214, 216, 218, 221, 223, 225, 227, 229, 230, 232, 234, 235, 237, 239, 240, 241, 243, 244, 245, 246, 247, 248, 249, 250, 250, 251, 252, 252, 253, 253, 253, 253, 253, 254, 253, 253, 253, 253, 253, 252, 252, 251, 250, 250, 249, 248, 247, 246, 245, 244, 243, 241, 240, 239, 237, 235, 234, 232, 230, 229, 227, 225, 223, 221, 218, 216, 214, 212, 209, 207, 205, 202, 200, 197, 194, 192, 189, 186, 184, 181, 178, 175, 172, 169, 166, 163, 160, 157, 154, 151, 148, 145, 142, 139, 136, 133, 130, 127, 123, 120, 117, 114, 111, 108, 105, 102, 99, 96, 93, 90, 87, 84, 81, 78, 75, 72, 69, 67, 64, 61, 59, 56, 53, 51, 48, 46, 44, 41, 39, 37, 35, 32, 30, 28, 26, 24, 23, 21, 19, 18, 16, 14, 13, 12, 10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 18, 19, 21, 23, 24, 26, 28, 30, 32, 35, 37, 39, 41, 44, 46, 48, 51, 53, 56, 59, 61, 64, 67, 69, 72, 75,78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123 };

static unsigned int toothsawdat[256] =

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 142, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233,

234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 252, 252, 253, 254, 255 };

static unsigned int triangdat[256] =

{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 255, 254, 252, 250, 248, 246, 244, 242, 240, 238, 236, 234, 232, 230, 228, 226, 224, 222, 220, 218, 216, 214, 212, 210, 208, 206, 204, 202, 200, 198, 196, 194, 192, 190, 188, 186, 184, 182, 180, 178, 176, 174, 172, 170, 168, 166, 164, 162, 160, 158, 156, 154, 152, 150, 148, 146, 144, 142, 140, 138, 136, 134, 132, 130, 128, 126, 124, 122, 120, 118, 116, 114, 112, 110, 108, 106, 104, 102, 100, 98, 96, 94, 92, 90, 88, 86, 84, 82, 80, 78, 76, 74, 72, 70, 68, 66, 64, 62, 60, 58, 56, 54, 52, 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4, 2 };

unsigned int timeindex1int[256], timeindex2int[256], timeindexbuffint [256];

unsigned int TempTable1int[256], TempTable2int[256]; // Temporary tables //modes 2 & 3, scale 0 to 255

unsigned int BuffTable[256]; // Output table, data to port

double trunc(**double** arg);

float tbs, tbs1, tbs2; // time between samples, signal 1 and 2

int dat_samples_buff1, dat_samples_temp1, dat_samples_temp2; //number of //samples in output table

int opmode; //operation mode: 1 single signlal, 2 superimposed, 3 separate //signals

```
int signaltype; //signal type: 1 sine, 2 saw-tooth, 3 triangle
float freq1, freq2; //selected frequency for outputs 1 and 2
int N, n1, n2; //samples per waveform cycle, signal 1 and 2
void GetOperParam()
{
printf ("Operation mode: 1, 2 or 3:\n ");
scanf ("%d", &opmode);
//printf ("Selected Operation mode = %d\n", opmode);
printf ("Signal type, 1 sine, 2 tooth, 3 triang:\n ");
scanf ("%d", &signaltype);
//printf ("Selected signal type = %d\n", signaltype);
printf ("Output single/low frequency in KiloHertz:\n ");
scanf ("%f", &freq1);
printf ("Samples per cycle: \n");
scanf ("%d", &n1);
if (n1<9) n1=8;
if (n1>8 & n1< 17) n1=16;
if (n1>16 & n1< 33) n1=32;
if (n1>32 & n1< 65) n1=64;
if (n1>64 & n1< 129) n1=128;
```

```
if (n1>128) n1=256;
printf(" %d\n", n1);
if (opmode>1)
{
printf ("Output high frequency2 in KiloHertz:\n ");
scanf ("%f", &freq2);
printf ("Samples per cycle 2: \n");
scanf ("%d", &n2);
if (n2<9) n2=8;
if (n2>8 & n2< 17) n2=16;
if (n2>16 & n2< 33) n2=32;
if (n2>32 & n2< 65) n2=64;
if (n2>64 & n2< 129) n2=128;
if (n2>128) n2=256;
printf(" %d\n", n2);
printf ("Leaving function GetOperParam\n");
}
}
void SineDisplay(int N)
{
```

```
unsigned int iter;
printf ("Entering function SineDisplay ORIGINAL SINE TABLE \n");
for (iter = 0; iter < N; iter++)
{
printf("[%d] ", iter);
printf(" %d\n", sinedatint[iter]);
}
}
void BuffTableGen(int N, int n)
{
unsigned int dsepi;
double dsepd;
unsigned int iter;
unsigned int i;
unsigned int j;
printf ("Entering function BuffTableGen mode 1\n");
i=0;
dsepd=N/n;
dsepi=dsepd;
j=dsepi;
```

```
printf("data separation int, mode1 = %d\n ", dsepi);
for (iter = 0; iter < N; iter=iter+dsepi)
{
/*printf("Data number = %d\n ", iter);*/
if (signaltype == 1) BuffTable[i]=sinedatint[iter];
if (signaltype == 2) BuffTable[i]=toothsawdat[iter];
if (signaltype == 3) BuffTable[i]=triangdat[iter];
/*printf("Original data = %.6ef\n ", sinedat[iter]);*/
printf("%d\n ", BuffTable[i]);
dat_samples_buff1=i;
i=i+1;
}
printf("Amount of data samples in buffer table = %d\n ", dat_samples_buff1);
printf ("Leaving function BuffTableGen mode 1\n");
}
void TempTable1Calc(int N, int n1)//prepare temp table signal 1, modes 2 & 3
{
unsigned int dsepi;
double dsepd;
unsigned int iter;
```

```
unsigned int i;
float t;
printf ("Entering function TempTable1Calc, modes 2 & 3\n");
i=0;
dsepd=N/n1;
dsepi=dsepd;
if ((dsepd-dsepi)>0.495)
{dsepi++; //if separation is 12.5 round up to 13
printf ("Table 1 separation %d \n", dsepi);
}
//printf("data separation in TEMPORARY TABLE 1 = %d\n ", dsepi);
printf ("it time out data\n");
for (iter = 0; iter < N; iter=iter+dsepi)
{
t=tbs1*i;
printf("[%d] >", iter);
printf("[%d] ", i);
//printf("%.3ef ", t);
if (signaltype==1) TempTable1int[i]=sinedatint[iter];
if (signaltype==2) TempTable1int[i]=toothsawdat[iter];
```

```
if (signaltype==3) TempTable1int[i]=triangdat[iter];

timeindex1int[i]=100000*t;

printf("t=%d ", timeindex1int[i]);

printf("%d ", TempTable1int[i]);

printf("%x\n ", TempTable1int[i]);

dat_samples_temp1=i;

i=i+1;

}

//printf("Data samples in temporary table1 = %d\n ",

// ( at_samples_temp1+1));

//printf ("Leaving function TempTable1Calc, modes 2 & 3\n");

}

void TempTable2Calc(int N, int n2)//prepare temp table signal 2, modes 2&3

{

unsigned int dsepi;

double dsepd;

unsigned int iter;

unsigned int i;

float t;

printf ("Entering function TempTable2Calc, modes 2 & 3\n");
```

```
i=0;
dsepd=N/n2;
printf(" %.2ef ", dsepd);
dsepi=dsepd;
if ((dsepd-dsepi)>0.495)
{dsepi++;
printf ("Table 2 separation %d \n", dsepi);
}
//printf("data separation in TEMPORARY TABLE 2 = %d\n ", dsepi);
printf ("it time out data\n");
for (iter = 0; iter < N; iter=iter+dsepi)
{
t=tbs2*i;
printf("[%d] >", iter);
printf("[%d] ", i);
//printf("%.3ef ", t);
if (signaltype==1) TempTable2int[i]=sinedatint[iter];
if (signaltype==2) TempTable2int[i]=toothsawdat[iter];
if (signaltype==3) TempTable2int[i]=triangdat[iter];
timeindex2int[i]=100000*t;
```

```
printf("t=%d ", timeindex2int[i]);
printf("%d ", TempTable2int[i]);
printf("%x\n ", TempTable2int[i]);
dat_samples_temp2=i;
i=i+1;
}
}
void BuffTableSuperposition()////prepare output table, mode 2
{
unsigned int i, j, k, l, m, dato1, dato2;
float t, tbsmin, tmax;
unsigned int tint, auxt1, auxt2;
printf ("Entering function BuffTableSuperposition, mode 2\n");
if (tbs1<tbs2)
tbsmin=tbs1;
else
tbsmin=tbs2;
if (freq1<freq2)
tmax=1/freq1;
else
```

```
tmax=1/freq2;
t=0; i=0; j=0; k=0;
dato1=TempTable1int[i];
dato2=TempTable2int[i];
BuffTable[k]=dato1+dato2;
printf("%d ", k);
printf(" %.2ef ", t);
printf(" %x ", dato2);
printf("+ %x ", dato1);
printf("= %d ", BuffTable[k]);
printf("= %x\n ", BuffTable[k]);
do{
t=t+tbsmin;
tint=t*100000+1;
auxt1=timeindex1int[i+1];
auxt2=timeindex2int[j+1];
m=0;
if (tint<auxt1)
l=0;
else{
```

```
m=1;
i=i+1;
//printf("new data table111, index %d \n", i);
dato1=TempTable1int[i];
}
//printf("m value after checking table1 %d \n", m);
if (tint<auxt2)
l=0;
else{
m=2;
j=j+1;
if (j==n2) j=0; // return to begin of temp table for low frequency
dato2=TempTable2int[j];
}
if(m>0)
{
k++;
if (opmode==2) BuffTable[k]=(dato1+dato2)/2;
if (opmode==3) BuffTable[k]=dato1+dato2*256;
timeindexbuffint[k]=tint;
```

```
printf("[%d] ", k);

printf("t= %d ", tint);

printf(" %x ", dato2);

printf("+ %x ", dato1);

printf("= %d ", BuffTable[k]);

printf("= %x\n ", BuffTable[k]);

}

}

while (t<tmax);

dat_samples_buff1=k;

printf("Amount of data samples in BUFFER TABLE = %d\n", k);

}

void tbsCalc(float freq, float n)//calculate time for requested frequency //
and number of samples

{

printf ("Entering function tbsCalc\n");

tbs=1/(freq*n);

}

void WriteToOut()//load data from output table, write to output port

{

unsigned int i;
```

```
unsigned int j;

unsigned int k;

float t;

printf ("Entering function WriteToOut\n");

printf("Time running between samples buffer1 = %.2ef\n", tbs1);

printf ("it time out data\n");

for (i = 0; i < dat_samples_buff1-1; i=i+1)

{

for (j = 0; j < tbs1*1e+5; j=j+1)// 1e+5 proportional to time // between samples

k=k+1;

t=tbs1*i;

printf("[%d] ", i);

printf(" %d ", timeindexbuffint[i]);

printf(" %x\n ", BuffTable[i]);

}

printf ("Leaving function WriteToOut\n");

}

int main(void)

{

N=256;
```

```
GetOperParam(); //Get operation parameters
//SineDisplay(N); //Display data samples for sine waveform
if (opmode==1)// operation mode = 1?
{
tbsCalc(freq1, n1); //calculate separation between samples
tbs1=tbs;
BuffTableGen(N, n1); //generate buffer table extracting samples
}
if (opmode>1)
{// operation mode= 2 or 3?
tbsCalc(freq1, n1); //calculate separation between samples, signal 1
tbs1=tbs;
TempTable1Calc(N, n1); //generate temp table for signal 1
tbsCalc(freq2, n2); //calculate separation between samples, signal 2
tbs2=tbs;
TempTable2Calc(N, n2); // generate temp table for signal 2
BuffTableSuperposition(); // generate buffer table modes 2 & 3
}
WriteToOut(); //write to output port
return 0 ;
```

```
}
```

A.2 Application Program: Extended Version

The extended version of the application program has been developed for the board based prototype implementation. It has added functionality compared to the SoC based design. Additional functions were defined and implemented according to experimental needs and developing research work in the particle manipulation area.

```
// Uses Luminary Driverlib for parallel port use
// Version date: Feb the 3rd, 2011
// Details: separates frequency ranges in low (<400 Hz) and high (>400Hz)
// Delivers superimposed frequencies in any mix of available waveforms
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "hw_memmap.h"
#include "hw_types.h"
#include "sysctl.h"
#include "hw_sysctl.h"
#include "gpio.h"
#include "hw_gpio.h"
```

```
//#define PORT_DATA(GPIO_PIN_0 |GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | //GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7)

#ifdef DEBUG

void

__error__(char *pcFilename, unsigned long ulLine)

{

}

#endif

#define Pi 3.14159265358979323846264338327

static unsigned int sinedatint[256] =

{ 127, 130, 133, 136, 139, 142, 145, 148, 151, 154, 157, 160, 163, 166, 169, 172, 175, 178, 181, 184, 186, 189, 192, 194, 197, 200, 202, 205, 207, 209, 212, 214, 216, 218, 221, 223, 225, 227, 229, 230, 232, 234, 235, 237, 239, 240, 241, 243, 244, 245, 246, 247, 248, 249, 250, 250, 251, 252, 252, 253, 253, 253, 253, 253, 254, 253, 253, 253, 253, 253, 252, 252, 251, 250, 250, 249, 248, 247, 246, 245, 244, 243, 241, 240, 239, 237, 235, 234, 232, 230, 229, 227, 225, 223, 221, 218, 216, 214, 212, 209, 207, 205, 202, 200, 197, 194, 192, 189, 186, 184, 181, 178, 175, 172, 169, 166, 163, 160, 157, 154, 151, 148, 145, 142, 139, 136, 133, 130, 127, 123, 120, 117, 114, 111, 108, 105, 102, 99, 96, 93, 90, 87, 84, 81, 78, 75, 72, 69, 67, 64, 61, 59, 56, 53, 51, 48, 46, 44, 41, 39, 37, 35, 32, 30, 28, 26, 24, 23, 21, 19, 18, 16, 14, 13, 12, 10, 9, 8, 7, 6, 5, 4, 3, 3, 2, 1, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 18, 19, 21, 23, 24, 26, 28, 30, 32, 35, 37, 39, 41, 44, 46, 48, 51, 53, 56, 59, 61, 64, 67, 69, 72, 75,78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123 };

static unsigned int toothsawdat[256] =
```

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 142, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 252, 252, 253, 254, 255 };

static unsigned int triangdat[256] =

{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170, 172, 174, 176, 178, 180, 182, 184, 186, 188, 190, 192, 194, 196, 198, 200, 202, 204, 206, 208, 210, 212, 214, 216, 218, 220, 222, 224, 226, 228, 230, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254, 255, 254, 252, 250, 248, 246, 244, 242, 240, 238, 236, 234, 232, 230, 228, 226, 224, 222, 220, 218, 216, 214, 212, 210, 208, 206, 204, 202, 200, 198, 196, 194, 192, 190, 188, 186, 184, 182, 180, 178, 176, 174, 172, 170, 168, 166, 164, 162, 160, 158, 156, 154, 152, 150, 148, 146, 144, 142, 140, 138, 136, 134, 132, 130, 128, 126, 124, 122, 120, 118, 116, 114, 112, 110, 108, 106, 104, 102, 100, 98, 96, 94, 92, 90, 88, 86, 84, 82, 80, 78, 76, 74, 72, 70, 68, 66, 64, 62, 60, 58, 56, 54, 52, 50, 48, 46, 44, 42, 40, 38, 36, 34, 32, 30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4, 2 };

unsigned int timeindex1int[256], timeindex2int[256], timeindexbuffint[256];

```
    unsigned int TempTable1int[256], TempTable2int[256]; // Temporary tables,
//modes 2 & 3, scale 0 to 255

    unsigned int BuffTable[4096]; // output table, all modes

    double trunc(double arg);

    float tbs, tbs1, tbs2; // time between samples, para senal 1 y 2

    int dat_samples_buff1, dat_samples_temp1, dat_samples_temp2; //data
samples

    int opmode; //operation modes: 1 single signal, 2 superposition, 3 separate
//signals

    int signaltype, signaltype2; //signal type: 1 sine, 2 saw-tooth, 3 triangle

    float freq1, freq2; //frecuency for output signals 1 & 2

    int N, n1, n2; //samples per waveform cycle

    void GetOperParam()

    {

    //printf ("Operation mode: 1, 2 or 3:\n ");

    // scanf ("%d", &opmode);

    opmode=2;

    //printf ("Selected Operation mode = %d\n", opmode);

    //printf ("Signal type, 1 sine, 2 tooth, 3 triang:\n ");

    //scanf ("%d", &signaltype);

    signaltype2=2;

    signaltype=2;
```

```
//printf ("Selected signal type = %d\n", signaltype);
//printf ("Output single/low frequency in KiloHertz:\n ");
//scanf ("%f", &freq1);
freq1=500;
//printf ("Samples per cycle: \n");
//scanf ("%d", &n1);
//if (freq1>399) {
//n1=240000/freq1; //write to port, high frequencies
//n2=16;
//}
//if (freq1<400) {
//n1=64;
//n2=64; //write to port, low frequencies
//}
n1=64;
if (n1<9) n1=16;
if ((n1>8) & (n1< 17)) n1=16;
if ((n1>16) & (n1< 33)) n1=32;
if ((n1>32) & (n1< 65)) n1=64;
if ((n1>64) & (n1< 129)) n1=128;
```

```
if (n1>128) n1=256;
//printf(" %d\n", n1);
if (opmode>1)
{
//printf ("Output high frequency2 in KiloHertz:\n ");
//scanf ("%f", &freq2);
freq2=5000;
//printf ("Samples per cycle 2: \n");
//scanf ("%d", &n2);
n2=64;
if (n2<9) n2=8;
if ((n2>8) & (n2< 17)) n2=16;
if ((n2>16) & (n2< 33)) n2=32;
if ((n2>32) & (n2< 65)) n2=64;
if ((n2>64) & (n2< 129)) n2=128;
if (n2>128) n2=256;
//printf(" %d\n", n2);
//printf ("Leaving function GetOperParam\n");
}
}
```

```
void SineDisplay(int N)
{
unsigned int iter;
// printf ("Entering function SineDisplay ORIGINAL SINE TABLE\n");
for (iter = 0; iter < N; iter++)
{
//printf("[%d] ", iter);
//printf(" %d\n", sinedatint[iter]);
}
}
void BuffTableGen(int N, int n)
{
unsigned int dsepi;
double dsepd;
unsigned int iter;
unsigned int i;
unsigned int j;
//printf ("Entering function BuffTableGen mode 1\n");
i=0;
dsepd=N/n;
```

```
dsepi=dsepd;

j=dsepi;

// printf("data separation int, modo1 = %d\n ", dsepi);

for (iter = 0; iter < N; iter=iter+dsepi)

{

// /*printf("Data number = %d\n ", iter);*/

if (signaltype == 1) BuffTable[i]=sinedatint[iter];

if (signaltype == 2) BuffTable[i]=toothsawdat[iter];

if (signaltype == 3) BuffTable[i]=triangdat[iter];

// /*printf("Dato original = %.6ef\n ", sinedat[iter]);*/

// printf("%d\n ", BuffTable[i]);

dat_samples_buff1=i;

i=i+1;

}

// printf("Amount of data samples in buffer table = %d\n ", //dat_samples_buff1);

//printf ("Leaving function BuffTableGen, mode 1\n");

}

void TempTable1Calc(int N, int n1)//prepare temp table, signal 1, modes 2&3

{
```

```
unsigned int dsepi;

double dsepd;

unsigned int iter;

unsigned int i;

float t;

// printf ("Entering function TempTable1Calc, modes 2 & 3\n");

i=0;

dsepd=N/n1;

dsepi=dsepd;

if ((dsepd-dsepi)>0.495)

{dsepi++; //if separation es >. 495 round up to next integer

// printf ("separation table 1 %d \n", dsepi);

}

//printf("data separation in TEMPORARY TABLE 1 = %d\n ", dsepi);

//printf ("it time out data\n");

for (iter = 0; iter < N+1; iter=iter+dsepi)

{

t=tbs1*i;

// printf("[%d] >", iter);

//printf("[%d] ", i);
```

```
//printf("%.3ef ", t);
if (signaltype==1) TempTable1int[i]=sinedatint[iter];
if (signaltype==2) TempTable1int[i]=toothsawdat[iter];
if (signaltype==3) TempTable1int[i]=triangdat[iter];
timeindex1int[i]=100000*t;
//printf("t=%d ", timeindex1int[i]);
//printf("%d ", TempTable1int[i]);
//printf("%x\n ", TempTable1int[i]);
dat_samples_temp1=i;
i=i+1;
}
//printf("Data samples in temporary table1 = %d\n ", (dat_samples_temp1+1));
//printf ("Leaving function TempTable1Calc, modes 2 & 3\n");
}
void TempTable2Calc(int N, int n2)//prepare temp table signal 2, modes 2&3
{
unsigned int dsepi;
double dsepd;
unsigned int iter;
```

```
unsigned int i;
float t;
//printf ("Entering function TempTable2Calc, modes 2 & 3\n");
i=0;
dsepd=N/n2;
//printf(" %.2ef ", dsepd);
dsepi=dsepd;
if ((dsepd-dsepi)>0.495)
{dsepi++;
// printf ("separation table 2 %d \n", dsepi);
}
//printf("data separation in TEMPORARY TABLE 2 = %d\n ", dsepi);
//printf ("it time out data\n");
for (iter = 0; iter < N; iter=iter+dsepi)
{
t=tbs2*i;
// printf("[%d] >", iter);
//printf("[%d] ", i);
//printf("%.3ef ", t);
if (signaltype2==1) TempTable2int[i]=sinedatint[iter];
```

```
if (signaltype2==2) TempTable2int[i]=toothsawdat[iter];
if (signaltype2==3) TempTable2int[i]=triangdat[iter];
timeindex2int[i]=100000*t;
//printf("t=%d ", timeindex2int[i]);
//printf("%d ", TempTable2int[i]);
//printf("%x\n ", TempTable2int[i]);
dat_samples_temp2=i;
i=i+1;
}
}
void BuffTableSuperposition()////prepare output table mode 2
{
unsigned int i, j, k, l, m, dato1, dato2;
float t, tbsmin, tmax;
unsigned int tint, auxt1, auxt2;
//printf ("Entering function BuffTableSuperposition, mode 2\n");
if (tbs1<tbs2)
tbsmin=tbs1;
else
tbsmin=tbs2;
```

```
if (freq1<freq2)
tmax=1/freq1;
else
tmax=1/freq2;
t=0; i=0; j=0; k=0;
dato1=TempTable1int[i];
dato2=TempTable2int[i];
BuffTable[k]=(dato1+dato2)/2;
//printf("%d ", k);
//printf(" %.2ef ", t);
//printf(" %x ", data2);
//printf("+ %x ", data1);
//printf("= %d ", BuffTable[k]);
//printf("= %x\n ", BuffTable[k]);
do{
t=t+tbsmin;
tint=t*100000+1;
auxt1=timeindex1int[i+1];
auxt2=timeindex2int[j+1];
m=0;
```

```
if (tint<auxt1)
l=0;
else{
m=1;
i=i+1;
//printf("new data table111, index %d \n", i);
dato1=TempTable1int[i];
}
//printf("m value after checking table1 %d \n", m);
if (tint<auxt2)
l=0;
else{
m=2;
j=j+1;
if (j==n2) j=0; //returns to beginning of temp table, low frequency
dato2=TempTable2int[j];
}
if(m>0)
{
k++;
```

```
if (opmode==2) BuffTable[k]=(dato1+dato2)/2;

if (opmode==3) BuffTable[k]=dato1+dato2*256;

timeindexbuffint[k]=tint;

// printf("[%d] ", k);

// printf("t= %d ", tint);

// printf(" %x ", dato2);

//printf("+ %x ", dato1);

//printf("= %d ", BuffTable[k]);

//printf("= %x\n ", BuffTable[k]);

}

}

while (t<tmax);

dat_samples_buff1=k;

//printf("Amount of data samples in BUFFER TABLE = %d\n", k);

}

void tbsCalc(float freq, float n)//calculates time between samples

{

// printf ("Entering function tbsCalc\n");

tbs=1/(freq*n);

}
```

```
void WriteToOutLow()//load data from buffer table, write to output port
{
unsigned int i;
unsigned int j;
unsigned int k;
//float t;
// printf ("Entering function WriteToOut\n");
// printf("Time running between samples buffer1 = %.2ef\n", tbs1);
//printf ("it time out data\n");
// delayed cycle for slow signal generation
k=11000/freq1; // k in inverse proportion of desired frequency
for (;;)
for (i = 0; i < dat_samples_buff1+1; i=i+1)
{
for (j = 0; j < k; j=j+1) {} // generates time between //samples for slow frequencies running k wait cycles between output updates
// printf("[%d] ", i);
// printf(" %d ", timeindexbuffint[i]);
// printf(" %x\n ", BuffTable[i]);
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0 |GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, BuffTable[i]);
```

```
}
// printf ("Leaving function WriteToOut\n");
}
void WriteToOutHigh()//load data from buffer table, write to output port
{
unsigned int i;
//unsigned int j;
// printf("Time running between samples buffer1 = %.2ef\n", tbs1);
//printf ("it time out data\n");
for (;;)
for (i = 0; i < dat_samples_buff1+1; i=i+1)
{
//for (j = 0; j < tbs1*12; j=j+1)//con 1e+5 is time between //samples
// t is accumulated time in cycle, last values is period T of //waveform
//t=tbs1*i;
// printf("[%d] ", i);
// printf(" %d ", timeindexbuffint[i]);
// printf(" %x\n ", BuffTable[i]);
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0 |GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, BuffTable[i]);
```

```
}
// printf ("Leaving function WriteToOut\n");
}
int main(void)
{
//
// If running on Rev A2 silicon, turn the LDO voltage up to 2.75V. //This is a workaround to allow the PLL to operate reliably.
//
if(DEVICE_IS_REVA2)
{
SysCtlLDOSet(SYSCTL_LDO_2_75V);
}
//
// Set the clocking to run directly from the crystal.
// Default values assume an external crystal of 6MHz. See Luminary
// driverlib documentation for other values.
// Clock source can be:
// 'SYSCTL_USE_OSC | SYSCTL_OSC_MAIN' - use the xtal without PLL
// 'SYSCTL_USE_PLL | SYSCTL_OSC_MAIN' - use the xtal with PLL
// If using the PLL, the oscillator runs at 200MHz and then you select
```

```
// a division of this frequency to clock the core. Otherwise the //divider just

//directly divides the XTAL frequency.

// Use a 6MHz external XTAL directly with no division

//printf ("Entering sysctlclokset \n");

SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_6MHZ);

// Use the XTAL directly to clock the PLL with division by 4

//SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |

// SYSCTL_XTAL_6MHZ);

// printf ("Entering sysctlperipheralenable \n");

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

//printf ("Entering define gpio as output \n");

GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_0 |GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7);

//printf ("Entering write to port \n");

GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0 |GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7,0x55);

// printf ("Leaving write to port \n");

// Use the XTAL directly to clock the PLL with division by 4
```

```
// SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
// SYSCTL_XTAL_6MHZ);
N=256;
GetOperParam(); //Get operation parameters
//SineDisplay(N); //Display data samples for sine waveform
if (opmode==1)// operation mode = 1?
{
tbsCalc(freq1, n1); //calculate separation between samples
tbs1=tbs;
BuffTableGen(N, n1); //generate buffer table extracting samples
}
if (opmode>1)
{// operation mode= 2 or 3?
tbsCalc(freq1, n1); //calculate separation between samples, signal 1
tbs1=tbs;
TempTable1Calc(N, n1); //generate temp table for signal 1
tbsCalc(freq2, n2); //calculate separation between samples, signal 2
tbs2=tbs;
TempTable2Calc(N, n2); // generate temp table for signal 2
BuffTableSuperposition(); // generate buffer table modes 2 & 3
```

```
}
if (freq1>399) WriteToOutHigh(); // write to output port, high //frequencies
if (freq1<400) WriteToOutLow(); //write to output port, low //frequencies
while(1);
```

Martha Salomé López de la Fuente works at Universidad de Monterrey, Mexico, as a research professor for the intelligent systems and robotics programs. Her contributions in the research and education areas relate to digital systems design, application specific integrated circuits, embedded systems applications, intelligent systems, and service robots. Since 1994 she has been teaching courses on microprocessors, digital electronics, embedded systems, integrated circuit design, and hardware architectures for service robots. Martha has presented her research work in forums such as Engineering in Medicine and Biology Conference, CERMA Electronics Robotics and Automotive Mechanics Conference, Eurasian Multidisciplinary Forum, and World Forum on Internet of Things. She has also published scientific articles in the IEEE Xplore digital library, the Review of Scientific Instruments journal, and the European Scientific Journal.

ISBN: 978-1-940366-44-9
9 781940 366449 >
Price: US $80

To order additional copies of this book, please contact:
Science Publishing Group
book@sciencepublishinggroup.com
www.sciencepublishinggroup.com