## Question 1:

***main.c***

```
/**
 * main.c
 */
#include <stdint.h>
#include "tm4c123gh6pm.h"
#include "PLL.h"
#include "DAC.h"
#include "Piano.h"
#include "Sound.h"

int main(void)
{
  //uint8_t Data = 1;  // 0 to 15 DAC output

  PLL_Init(Bus80MHz);  // set system clock to 80 MHz

  //DAC_Init();
  Piano_Init();
  Sound_Init();

  int curNote = 0;

  while(1) {
    //DAC_Out(Data);    was used to test DAC
    curNote = Piano_In();
    Sound_Play(curNote);
    //Data = 0x0F & (Data+1);  // 0, 1, 2, ..., 14, 15, 0, 1, 2, ...
  }
}
```

***DAC.c***

```
/*
 * DAC.c
 *
 *  Created on: Dec 4, 2024
```

```
 *      Author: aanha
 */
#include <stdint.h>
#include "tm4c123gh6pm.h"

void GPIOPortB_Init() {
   SYSCTL_RCGC2_R |= 0b10;              // 1) turns on clock E
   GPIO_PORTB_AMSEL_R &= ~(0xFF);          // 3) disable analog function on PE5-0
   GPIO_PORTB_PCTL_R &= ~(0xFF);           // 4) enable regular GPIO
   GPIO_PORTB_DIR_R |= 0xFF;               // 5) outputs on PE5-0
   GPIO_PORTB_AFSEL_R &= ~(0xFF);          // 6) regular function on PE5-0
   GPIO_PORTB_DEN_R |= 0xFF;               // 7) enable digital on PE5-0
}

void GPIOPortD_Init() {
   SYSCTL_RCGC2_R |= 0b1000;               // 1) turns on clock D
   GPIO_PORTD_AMSEL_R &= ~(0xFF);          // 3) disable analog function on PD5-0
   GPIO_PORTD_PCTL_R &= ~(0xFF);           // 4) enable regular GPIO
   GPIO_PORTD_DIR_R |= 0xFF;               // 5) outputs on PD5-0
   GPIO_PORTD_AFSEL_R &= ~(0xFF);          // 6) regular function on PD5-0
   GPIO_PORTD_DEN_R |= 0xFF;               // 7) enable digital on PD5-0
}

void DAC_Init(void) {                   // main will call this function to init all the ports
   GPIOPortB_Init();
  // GPIOPortD_Init();
}

void DAC_Out(uint8_t data) {          // write 4-bit into ports to send data
   // clear (&=) and load (|=) bit into pins
   GPIO_PORTB_DATA_R &= ~(0b1111);   // pin 2 no worky

   //uint8_t dataLeft = data & 0b1100;  // will save the left two values
   //dataLeft = dataLeft << 1;         // moves over 1
   //uint8_t dataRight = data & 0b0011;
   //uint8_t newData = dataLeft | dataRight;

   GPIO_PORTB_DATA_R |= data;

}
```

***DAC.h***

```
/*
 * DAC.h
 *
 *  Created on: Dec 4, 2024
 *      Author: aanha
 */

#ifndef DAC_H_
#define DAC_H_

void DAC_Init(void);
void DAC_Out(uint8_t data);


#endif /* DAC_H_ */
```

***Piano.c***

```
/*
 * Piano.c
 *
 *  Created on: Dec 9, 2024
 *      Author: aanha
 */
#include <stdint.h>
#include "tm4c123gh6pm.h"


void GPIOPortE_Init() {
   SYSCTL_RCGC2_R |= 0b10000;              // 1) turns on clock B
   GPIO_PORTE_AMSEL_R &= ~(0xFF);          // 3) disable analog function on PB5-0
   GPIO_PORTE_PCTL_R &= ~(0xFF);           // 4) enable regular GPIO
   GPIO_PORTE_DIR_R &= ~(0xFF);            // 5) outputs on PB5-0
   GPIO_PORTE_AFSEL_R &= ~(0xFF);          // 6) regular function on PB5-0
   GPIO_PORTE_DEN_R |= 0xFF;              // 7) enable digital on PB5-0
}
/*void GPIOPortB_Init() {
```

```
   SYSCTL_RCGC2_R |= 0b10000;             // 1) turns on clock E
   GPIO_PORTE_AMSEL_R &= ~(0xFF);          // 3) disable analog function on PE5-0
   GPIO_PORTE_PCTL_R &= ~(0xFF);          // 4) enable regular GPIO
   GPIO_PORTE_DIR_R &= ~(0xFF);            // 5) outputs on PE5-0
   GPIO_PORTE_AFSEL_R &= ~(0xFF);          // 6) regular function on PE5-0
   GPIO_PORTE_DEN_R |= 0xFF;              // 7) enable digital on PE5-0
}*/

void Piano_Init(void) {
   //GPIOPortE_Init();
   GPIOPortE_Init();
}

int Piano_In(void) {
   int note = 0;
   if (GPIO_PORTE_DATA_R == 0b001) { // pin 0 pressed
      note = 523;     // c
   }
   else if (GPIO_PORTE_DATA_R == 0b010) { // pin 1 pressed
      note = 392;    // g
   }
   else if (GPIO_PORTE_DATA_R == 0b100) { // pin 2 pressed
      note = 294;    // d
   }
   return note;
}
```

***Piano.h***

```
/*
 * Piano.h
 *
 *  Created on: Dec 9, 2024
 *      Author: aanha
 */

#ifndef PIANO_H_
#define PIANO_H_

void Piano_Init(void);
```

```
int Piano_In(void);


#endif /* PIANO_H_ */
```

***Sound.c***

```
/*
 * Sound.c
 *
 *  Created on: Dec 10, 2024
 *      Author: aanha
 */
#include <stdint.h>
#include "tm4c123gh6pm.h"


void Sound_Init(void) {
  DAC_Init();
  SysTick_Init();
}

void Sound_Play(int curnote) {
  int period = 0;

  if (curnote == 523) {          // plays note c
    // 80m/523 (sytic count for whole sin wave) / by 32 to get count
    period = (80000000/523)/32;
  }
  else if (curnote == 392) {     // plays note g
    period = (80000000/392)/32;
  }
  else if (curnote == 294) {     // plays note d
    period = (80000000/294)/32;
  }
  else {
    period = 0;
  }

  NVIC_ST_RELOAD_R = period-1;
}
```

***Sound.h***

```
/*
 * Sound.h
 *
 *  Created on: Dec 10, 2024
 *      Author: aanha
 */

#ifndef SOUND_H_
#define SOUND_H_

void Sound_Init(void);
void Sound_Play(int curnote);


#endif /* SOUND_H_ */
```

***SysTickInts.c***

```
// SysTickInts.c
// Edited to run on Tiva-C
// Use the SysTick timer to request interrupts at a particular period.
// Daniel Valvano, Jonathan Valvano
// June 1, 2015

/* This example accompanies the books
   "Embedded Systems: Introduction to MSP432 Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
   Volume 1 Program 9.7

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
```

```
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include <stdint.h>
#include "Sound.h"
#include "tm4c123gh6pm.h"
#include "DAC.h"

void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 12.5ns (assuming 80 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
volatile uint32_t Counts;

const uint8_t wave[32]= {
8,9,11,12,13,14,14,15,15,15,14,14,13,12,11,9,8,7,5,4,3,2,2,1,1,1,2,2,3,4,5,7};

void SysTick_Init(uint32_t period) {
          long sr = StartCritical();

          Counts = 0;

          NVIC_ST_CTRL_R = 0;                // disable SysTick during setup
          NVIC_ST_RELOAD_R = period - 1;       // maximum reload value
          NVIC_ST_CURRENT_R = 0;              // any write to current clears it
          NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000;          //
priority 2
```

```
        NVIC_ST_CTRL_R = 0x00000007;        // enable SysTick with no interrupts

        EndCritical(sr);
}
uint8_t index = 8;

void SysTick_Handler(void) {
        // make code to output to DAC
   uint8_t Data = wave[index];
   index = index + 1;

   if (index > 31) {
      index = 0;
   }

   DAC_Out(Data);
}
```

***SysTickInts.h***

```
// SysTickInts.h
// Edited to run on Tiva-C
// Use the SysTick timer to request interrupts at a particular period.
// Jonathan Valvano
// June 1, 2015

/* This example accompanies the books
   "Embedded Systems: Introduction to MSP432 Microcontrollers",
   ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
   Volume 1 Program 9.7

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
```

```
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#ifndef __SYSTICKINTS_H__
#define __SYSTICKINTS_H__

// **************SysTick_Init*********************
// Initialize SysTick periodic interrupts
// Input: interrupt period
//        Units of period are 12.5ns (assuming 80 MHz clock)
//        Maximum is 2^24-1
//        Minimum is determined by length of ISR
// Output: none
void SysTick_Init(uint32_t period);

#endif
```

***PLL.c***

```
// PLL.c
// Runs on LM4F120/TM4C123
// A software function to change the bus frequency using the PLL.
// Daniel Valvano
// May 2, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
   Program 2.10, Figure 2.37

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
```

```
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#include <stdint.h>
#include "PLL.h"
#include "tm4c123gh6pm.h"

// The #define statement SYSDIV2 in PLL.h
// initializes the PLL to the desired frequency.

// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz
// see the table at the end of this file

#define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_XTAL_M       0x000007C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
#define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
#define SYSCTL_RCC2_USERCC2     0x80000000  // Use RCC2
#define SYSCTL_RCC2_DIV400      0x40000000  // Divide PLL as 400 MHz vs. 200
                                            // MHz
#define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
#define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
#define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
#define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
#define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC

// configure the system to get its clock from the PLL
// SYSDIV = 400/freq -1
// bus frequency is 400MHz/(SYSDIV+1)
void PLL_Init(uint32_t freq){
  // 0) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
```

```
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator
source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 4) set the desired system divider and the system divider least significant bit
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;  // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~0x1FC00000) // clear system clock divider field
                  + (freq<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}


/*
SYSDIV2  Divisor  Clock (MHz)
 0        1       reserved
 1        2       reserved
 2        3       reserved
 3        4       reserved
 4        5       80.000
 5        6       66.667
 6        7       reserved
 7        8       50.000
 8        9       44.444
 9        10       40.000
 10       11       36.364
 11       12       33.333
 12       13       30.769
 13       14       28.571
 14       15       26.667
 15       16       25.000
```

| | | |
|---|---|---|
| 16 | 17 | 23.529 |
| 17 | 18 | 22.222 |
| 18 | 19 | 21.053 |
| 19 | 20 | 20.000 |
| 20 | 21 | 19.048 |
| 21 | 22 | 18.182 |
| 22 | 23 | 17.391 |
| 23 | 24 | 16.667 |
| 24 | 25 | 16.000 |
| 25 | 26 | 15.385 |
| 26 | 27 | 14.815 |
| 27 | 28 | 14.286 |
| 28 | 29 | 13.793 |
| 29 | 30 | 13.333 |
| 30 | 31 | 12.903 |
| 31 | 32 | 12.500 |
| 32 | 33 | 12.121 |
| 33 | 34 | 11.765 |
| 34 | 35 | 11.429 |
| 35 | 36 | 11.111 |
| 36 | 37 | 10.811 |
| 37 | 38 | 10.526 |
| 38 | 39 | 10.256 |
| 39 | 40 | 10.000 |
| 40 | 41 | 9.756 |
| 41 | 42 | 9.524 |
| 42 | 43 | 9.302 |
| 43 | 44 | 9.091 |
| 44 | 45 | 8.889 |
| 45 | 46 | 8.696 |
| 46 | 47 | 8.511 |
| 47 | 48 | 8.333 |
| 48 | 49 | 8.163 |
| 49 | 50 | 8.000 |
| 50 | 51 | 7.843 |
| 51 | 52 | 7.692 |
| 52 | 53 | 7.547 |
| 53 | 54 | 7.407 |
| 54 | 55 | 7.273 |
| 55 | 56 | 7.143 |

| | | |
|---|---|---|
| 56 | 57 | 7.018 |
| 57 | 58 | 6.897 |
| 58 | 59 | 6.780 |
| 59 | 60 | 6.667 |
| 60 | 61 | 6.557 |
| 61 | 62 | 6.452 |
| 62 | 63 | 6.349 |
| 63 | 64 | 6.250 |
| 64 | 65 | 6.154 |
| 65 | 66 | 6.061 |
| 66 | 67 | 5.970 |
| 67 | 68 | 5.882 |
| 68 | 69 | 5.797 |
| 69 | 70 | 5.714 |
| 70 | 71 | 5.634 |
| 71 | 72 | 5.556 |
| 72 | 73 | 5.479 |
| 73 | 74 | 5.405 |
| 74 | 75 | 5.333 |
| 75 | 76 | 5.263 |
| 76 | 77 | 5.195 |
| 77 | 78 | 5.128 |
| 78 | 79 | 5.063 |
| 79 | 80 | 5.000 |
| 80 | 81 | 4.938 |
| 81 | 82 | 4.878 |
| 82 | 83 | 4.819 |
| 83 | 84 | 4.762 |
| 84 | 85 | 4.706 |
| 85 | 86 | 4.651 |
| 86 | 87 | 4.598 |
| 87 | 88 | 4.545 |
| 88 | 89 | 4.494 |
| 89 | 90 | 4.444 |
| 90 | 91 | 4.396 |
| 91 | 92 | 4.348 |
| 92 | 93 | 4.301 |
| 93 | 94 | 4.255 |
| 94 | 95 | 4.211 |
| 95 | 96 | 4.167 |

```
 96      97     4.124
 97      98     4.082
 98      99     4.040
 99      100    4.000
 100     101    3.960
 101     102    3.922
 102     103    3.883
 103     104    3.846
 104     105    3.810
 105     106    3.774
 106     107    3.738
 107     108    3.704
 108     109    3.670
 109     110    3.636
 110     111    3.604
 111     112    3.571
 112     113    3.540
 113     114    3.509
 114     115    3.478
 115     116    3.448
 116     117    3.419
 117     118    3.390
 118     119    3.361
 119     120    3.333
 120     121    3.306
 121     122    3.279
 122     123    3.252
 123     124    3.226
 124     125    3.200
 125     126    3.175
 126     127    3.150
 127     128    3.125
*/
```

***PLL.h***

```
// PLL.h
// Runs on LM4F120/TM4C123
// A software function to change the bus frequency using the PLL.
// Daniel Valvano
```

// May 3, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
   Program 2.10, Figure 2.37


 */

// The #define statement SYSDIV2 initializes
// the PLL to the desired frequency.
// SYSDIV = 400/freq -1
// bus frequency is 400MMHz/(SYSDIV+1)

// configure the system to get its clock from the PLL

#ifndef __PLL_H__
#define __PLL_H__

void PLL_Init(uint32_t freq);
#define Bus80MHz     4
#define Bus80_000MHz 4
#define Bus66_667MHz 5
#define Bus50_000MHz 7
#define Bus50MHz     7
#define Bus44_444MHz 8
#define Bus40_000MHz 9

```
#define Bus40MHz     9
#define Bus36_364MHz 10
#define Bus33_333MHz 11
#define Bus30_769MHz 12
#define Bus28_571MHz 13
#define Bus26_667MHz 14
#define Bus25_000MHz 15
#define Bus25MHz     15
#define Bus23_529MHz 16
#define Bus22_222MHz 17
#define Bus21_053MHz 18
#define Bus20_000MHz 19
#define Bus20MHz     19
#define Bus19_048MHz 20
#define Bus18_182MHz 21
#define Bus17_391MHz 22
#define Bus16_667MHz 23
#define Bus16_000MHz 24
#define Bus16MHz     24
#define Bus15_385MHz 25
#define Bus14_815MHz 26
#define Bus14_286MHz 27
#define Bus13_793MHz 28
#define Bus13_333MHz 29
#define Bus12_903MHz 30
#define Bus12_500MHz 31
#define Bus12_121MHz 32
#define Bus11_765MHz 33
#define Bus11_429MHz 34
#define Bus11_111MHz 35
#define Bus10_811MHz 36
#define Bus10_526MHz 37
#define Bus10_256MHz 38
#define Bus10_000MHz 39
#define Bus10MHz     39
#define Bus9_756MHz 40
#define Bus9_524MHz 41
#define Bus9_302MHz 42
#define Bus9_091MHz 43
#define Bus8_889MHz 44
```

```
#define Bus8_696MHz 45
#define Bus8_511MHz 46
#define Bus8_333MHz 47
#define Bus8_163MHz 48
#define Bus8_000MHz 49
#define Bus8MHz     49
#define Bus7_843MHz 50
#define Bus7_692MHz 51
#define Bus7_547MHz 52
#define Bus7_407MHz 53
#define Bus7_273MHz 54
#define Bus7_143MHz 55
#define Bus7_018MHz 56
#define Bus6_897MHz 57
#define Bus6_780MHz 58
#define Bus6_667MHz 59
#define Bus6_557MHz 60
#define Bus6_452MHz 61
#define Bus6_349MHz 62
#define Bus6_250MHz 63
#define Bus6_154MHz 64
#define Bus6_061MHz 65
#define Bus5_970MHz 66
#define Bus5_882MHz 67
#define Bus5_797MHz 68
#define Bus5_714MHz 69
#define Bus5_634MHz 70
#define Bus5_556MHz 71
#define Bus5_479MHz 72
#define Bus5_405MHz 73
#define Bus5_333MHz 74
#define Bus5_263MHz 75
#define Bus5_195MHz 76
#define Bus5_128MHz 77
#define Bus5_063MHz 78
#define Bus5_000MHz 79
#define Bus4_938MHz 80
#define Bus4_878MHz 81
#define Bus4_819MHz 82
#define Bus4_762MHz 83
```

```
#define Bus4_706MHz 84
#define Bus4_651MHz 85
#define Bus4_598MHz 86
#define Bus4_545MHz 87
#define Bus4_494MHz 88
#define Bus4_444MHz 89
#define Bus4_396MHz 90
#define Bus4_348MHz 91
#define Bus4_301MHz 92
#define Bus4_255MHz 93
#define Bus4_211MHz 94
#define Bus4_167MHz 95
#define Bus4_124MHz 96
#define Bus4_082MHz 97
#define Bus4_040MHz 98
#define Bus4_000MHz 99
#define Bus4MHz     99
#define Bus3_960MHz 100
#define Bus3_922MHz 101
#define Bus3_883MHz 102
#define Bus3_846MHz 103
#define Bus3_810MHz 104
#define Bus3_774MHz 105
#define Bus3_738MHz 106
#define Bus3_704MHz 107
#define Bus3_670MHz 108
#define Bus3_636MHz 109
#define Bus3_604MHz 110
#define Bus3_571MHz 111
#define Bus3_540MHz 112
#define Bus3_509MHz 113
#define Bus3_478MHz 114
#define Bus3_448MHz 115
#define Bus3_419MHz 116
#define Bus3_390MHz 117
#define Bus3_361MHz 118
#define Bus3_333MHz 119
#define Bus3_306MHz 120
#define Bus3_279MHz 121
#define Bus3_252MHz 122
```

```
#define Bus3_226MHz 123
#define Bus3_200MHz 124
#define Bus3_175MHz 125
#define Bus3_150MHz 126
#define Bus3_125MHz 127

#endif
```

## Question 2:

| Data (N) | PB3 | PB2 | PB1 | PB0 | Vout (theoretical) | Vout (measured) |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 V | 0 V |
| 1 | 0 | 0 | 0 | 3.3 | 0.22 V | 0.21 V |
| 2 | 0 | 0 | 3.3 | 0 | 0.44 V | 0.47 V |
| 3 | 0 | 0 | 3.3 | 3.3 | 0.66 V | 0.72 V |
| 4 | 0 | 3.3 | 0 | 0 | 0.88 V | 1.01 V |
| 5 | 0 | 3.3 | 0 | 3.3 | 1.1 V | 0.24 V |
| 6 | 0 | 3.3 | 3.3 | 0 | 1.32 V | 1.35 V |
| 7 | 0 | 3.3 | 3.3 | 3.3 | 1.54 V | 1.57 V |
| 8 | 3.3 | 0 | 0 | 0 | 1.76 V | 1.81 V |
| 9 | 3.3 | 0 | 0 | 3.3 | 1.98 V | 2.1 V |
| 10 | 3.3 | 0 | 3.3 | 0 | 2.2 V | 2.34 V |
| 11 | 3.3 | 0 | 3.3 | 3.3 | 2.42 V | 2.53 V |
| 12 | 3.3 | 3.3 | 0 | 0 | 2.64 V | 2.81 V |
| 13 | 3.3 | 3.3 | 0 | 3.3 | 2.86 V | 2.96 V |
| 14 | 3.3 | 3.3 | 3.3 | 0 | 3.08 V | 3.11 V |
| 15 | 3.3 | 3.3 | 3.3 | 3.3 | 3.3 V | 3.29 V |

I found the theoretical Vout by taking the voltage 3.3 and multiplying it by the Data, N, divided by 15, as seen below:

$$Vout = 3.3 \times \frac{N}{15}.$$

My Vout measured came out as expected since I used a similar ratio of resistors to the DAC diagram shown in the lab manual allowing similar voltages to get passed through the DAC.

## Question 3:

The range is 0 to 3.3V since the power being supplied is up to 3.3V. Now since our DAC is 4-bits, you find the precision to be:

$$2^4 = 16\ alternatives.$$

As for the resolution, this is calculated by taking the range and the precision values and calculating them as so:

$$3.3 \div (2^4 - 1) = 0.22V.$$

## Question 4:

The formula used to calculate the timer load values for each piano note was:

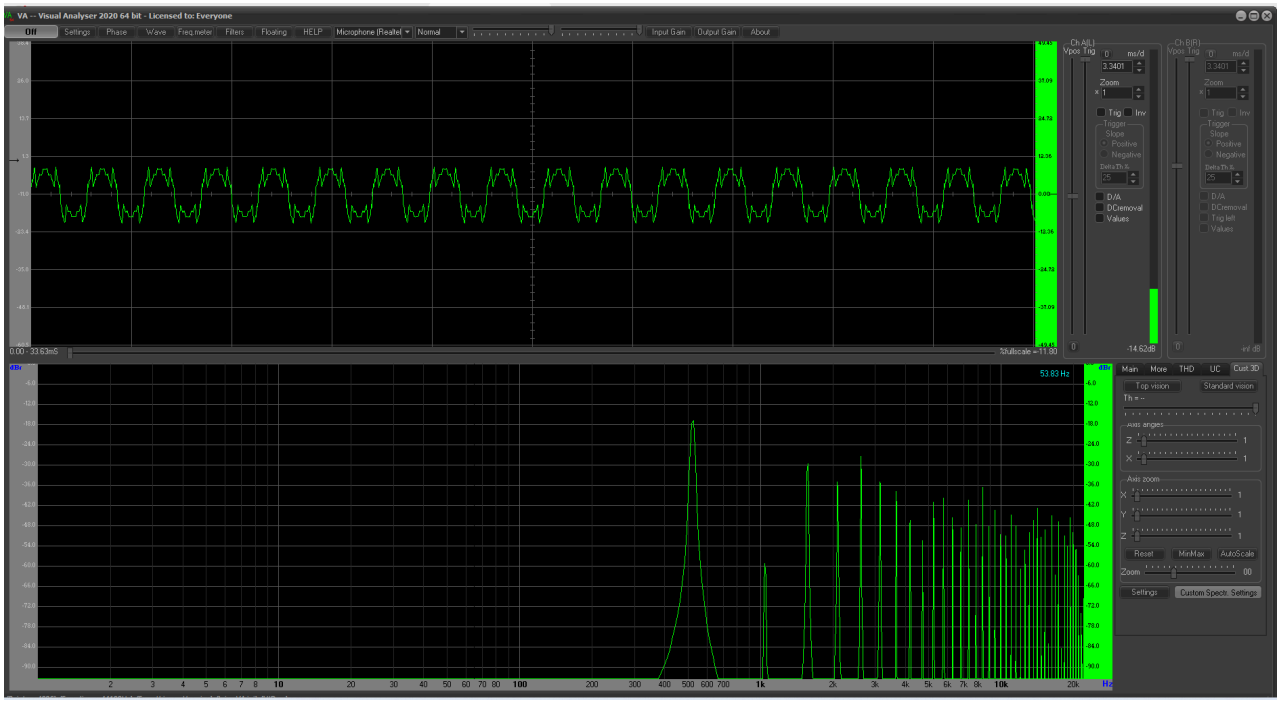
$$(80MHz \div note\ frequency) \div 32\ .$$

The first part of the formula was to find the systick count for an entire sine wave. We then divide that by 32, since we have 32 points in our wave array. So the calculations for the timer load values of each of my notes go as follows:

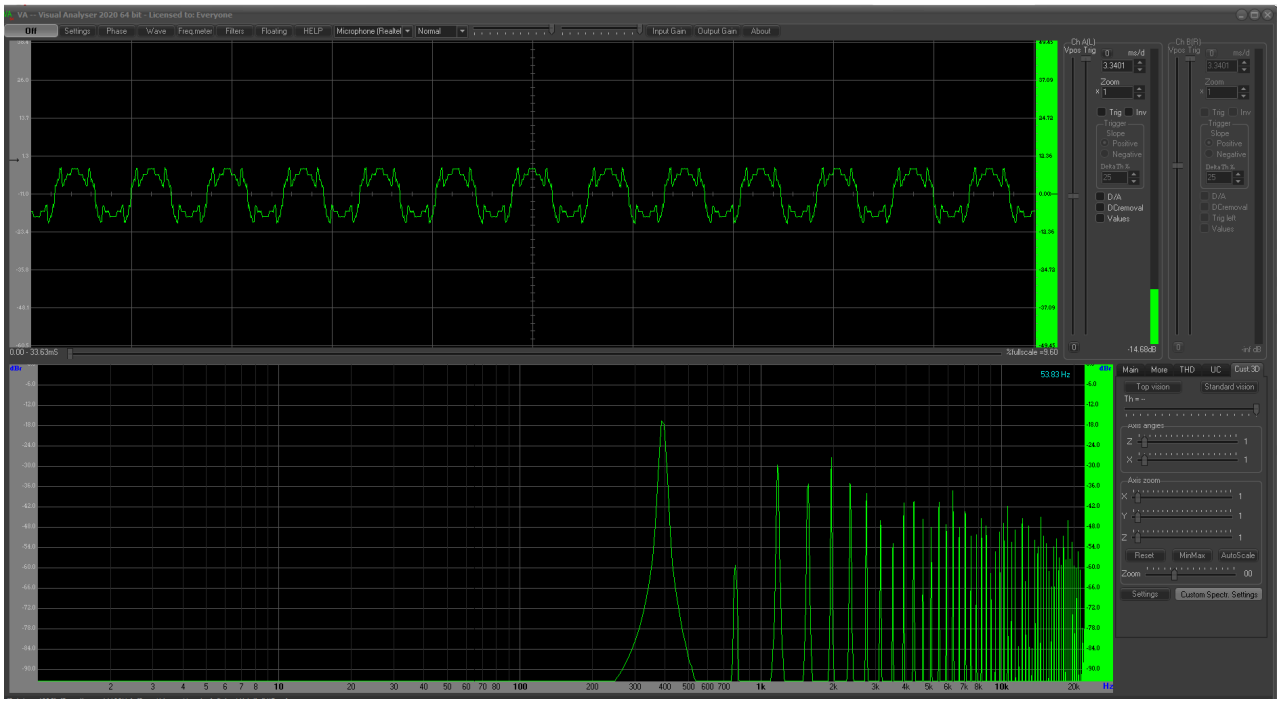
$$(80MHz \div 523) \div 32 = note\ C$$
$$(80MHz \div 392) \div 32 = note\ G$$
$$(80MHz \div 294) \div 32 = note\ D$$

Question 5:

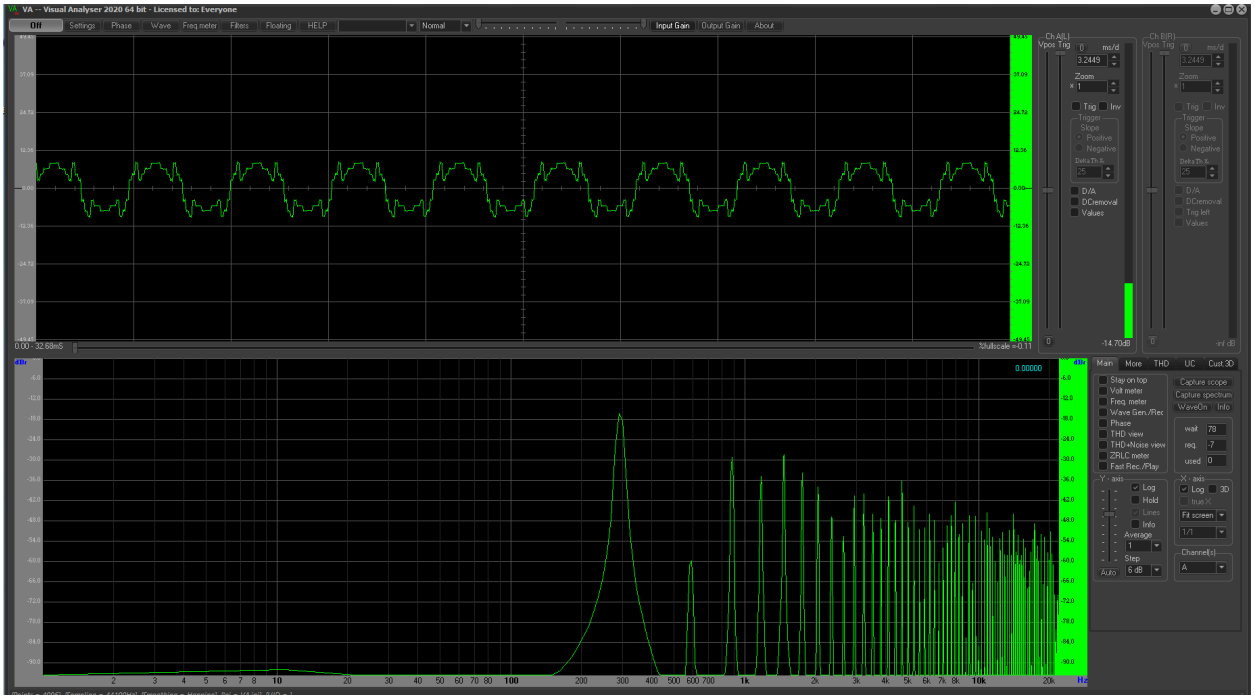
***Note C -***




***Note G -***



***Note D -***



## Question 6:

If the number of samples per cycle of the sinusoid were to double from 32 to 64, the quality of the DAC output waveform would improve. The reason why is that it provided more accuracy to the sinusoid since there are more values for the DAC to output making a more smooth waveform at every systick cycle.

## Question 7:

The quality of the DAC output would not change. The reason for this is that the DAC is only pushing out the data from the board into the GPIO data register and is reliant on the data provided from the wave array in the systickinit.c file. Since the wave array is not different, it would not improve the quality of the waveform.

## Question 8:

If we change the 4-bit DAC to a 6-bit DAC then the resolution would change as so:

$$3.3 \div (2^6 - 1) = 0.052V$$

## Question 9:

How I would set up the 6-bit DAC is by keeping the ratio of the resistors in a 1:2:4 ratio where the next resistance value goes up by double the previous. I would then create a resistance of 0.5k, 1k, 2k, 4k, 8k, and 16k that would connect to B5-B0 respectively. As for the DAC

implementation I would ensure to enable PortB pins 0 to 5 along with ensuring that I mask or clear all the 6 pins.

## Question 10:

The student I talked to about the assignment was Brenden Dack.