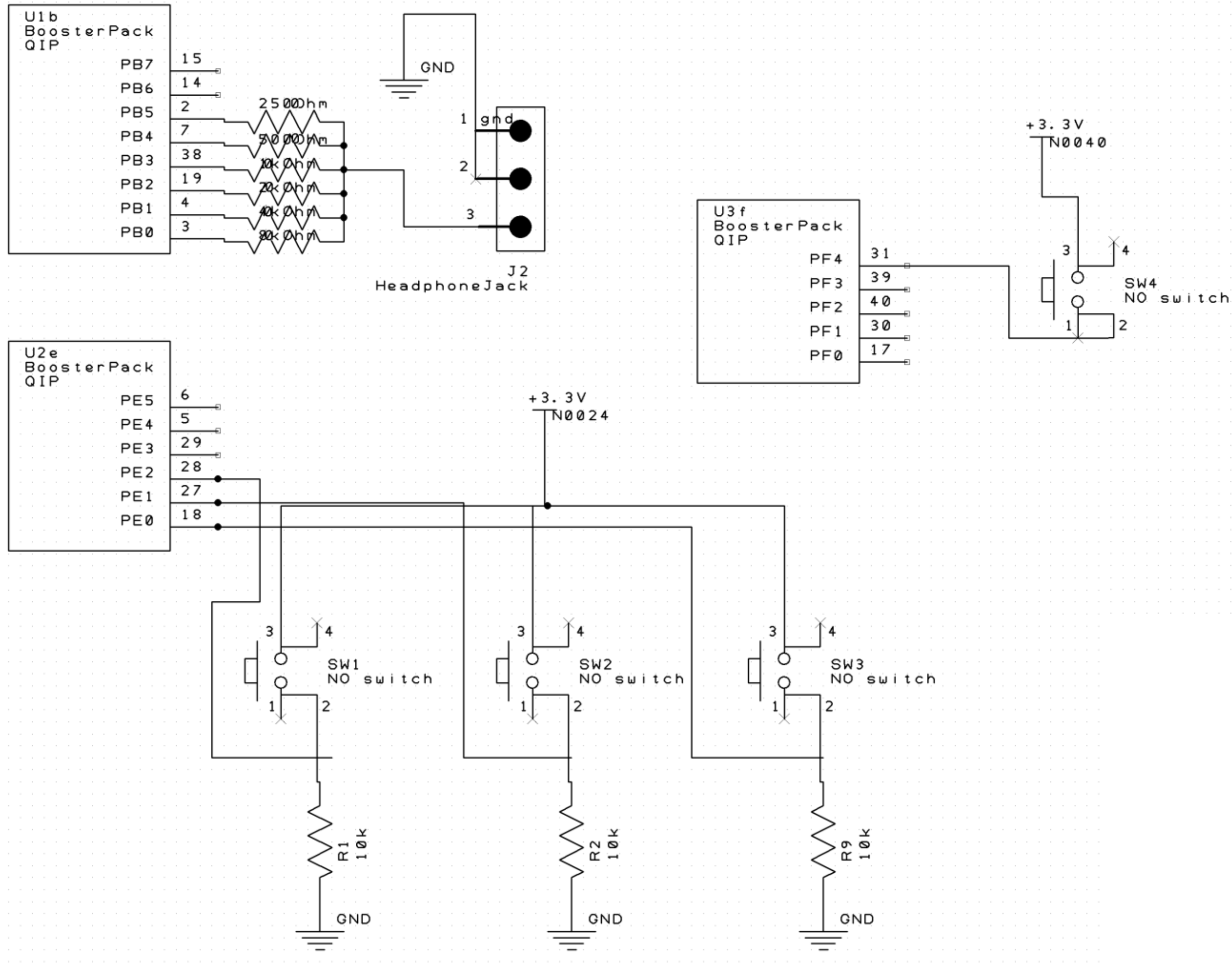
Lab 6 Deliverable

2. Circuit Diagram showing the DAC and any other hardware used in this lab, PCB Artist.

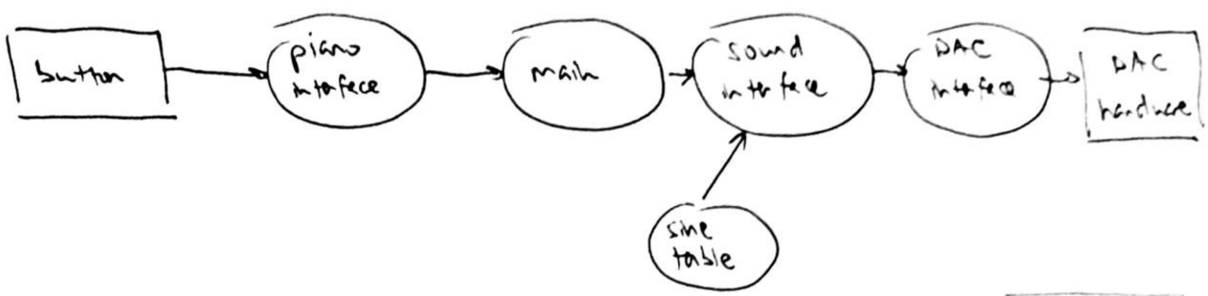


3. Software Design

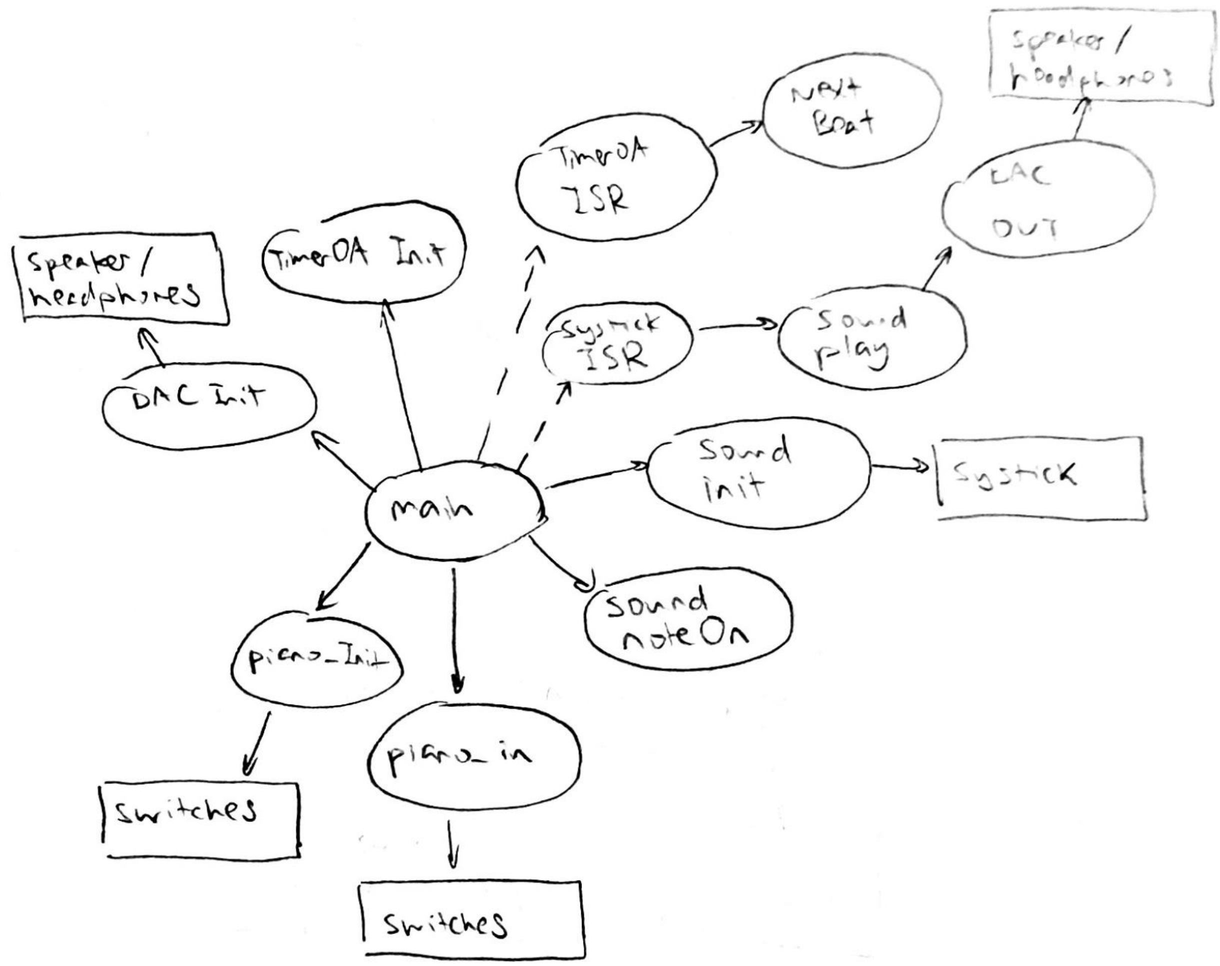
a. Draw Pictures of the Data Structures Used to Store the Sound Data

b. If you organized the system different than Figure 6.6 and 6.7, then draw its data flow and call graphs

Data Flow



Call Graph



Sound.c Design

Sound.c: implements ability to sound more than one note at a time, & an envelope.

Arrays Sin64_32pts_Sine and Sin64_32pts hold wave form information.

dBtable holds envelope information.

notePeriodTable holds the period of notes 0~108 divided by 32. For note n, every notePeriodTable[n] microseconds, the sine wave pointer should be incremented to produce the note's period.

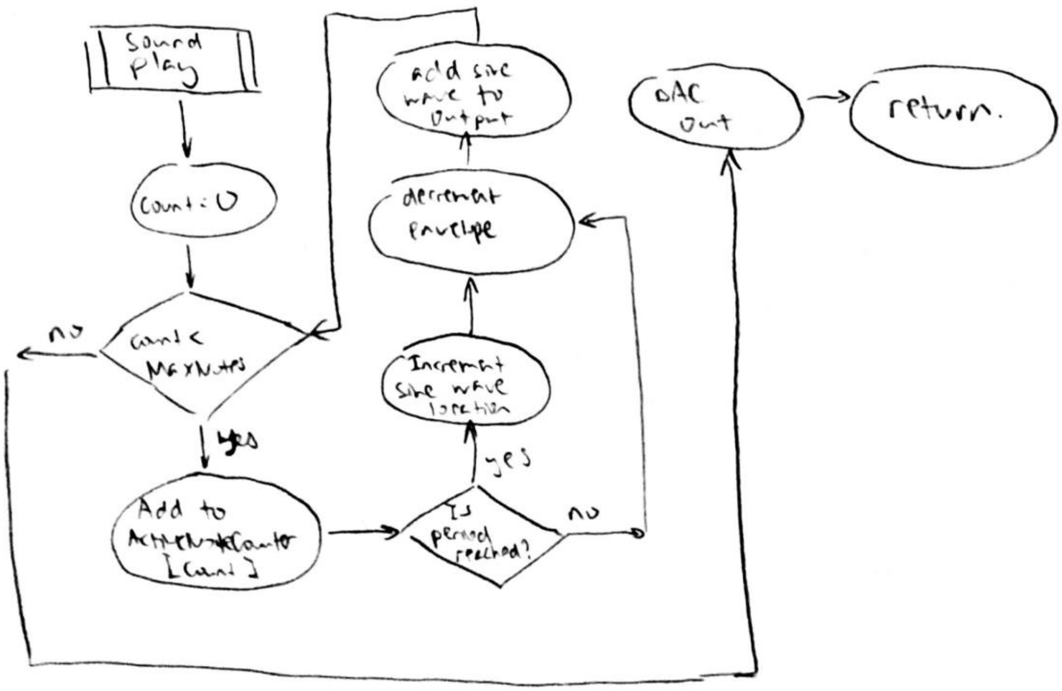
Active Note Periods	Holds the active note's period info
Active Note Envelope	Holds the active note's envelope mask location.
Active Note Counter	Holds the number of µs since the note was generated
Active Note WaveLocation	Holds the note's location in its sine wave.

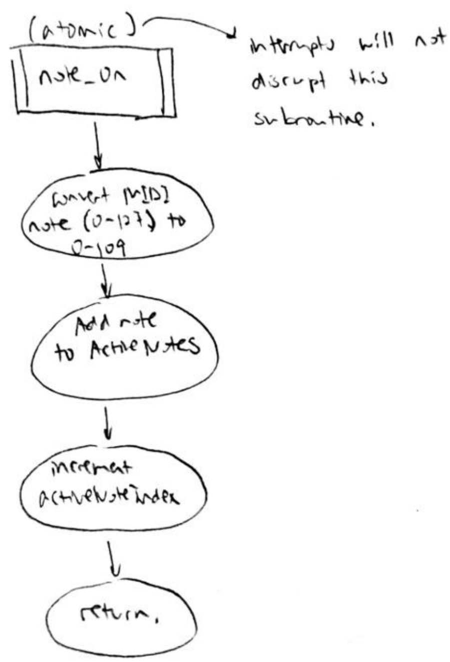
Above 4 arrays are used to achieve the sounding of 4 notes at one time.

The subroutine noteOn() adds a note to the active notes.

All active notes are played in Sound_Play().

The SysTick only calls Sound_Play().





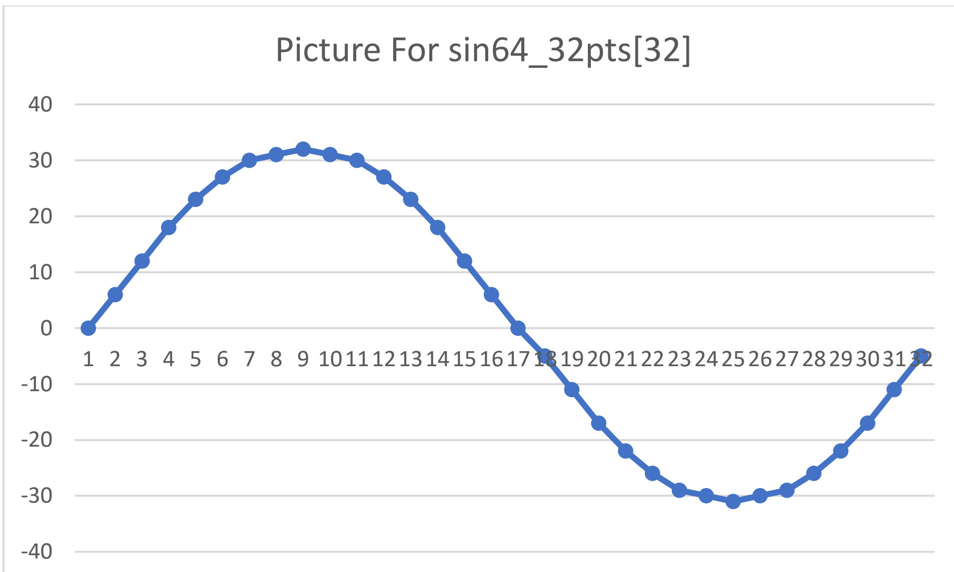
songBeat data structure:

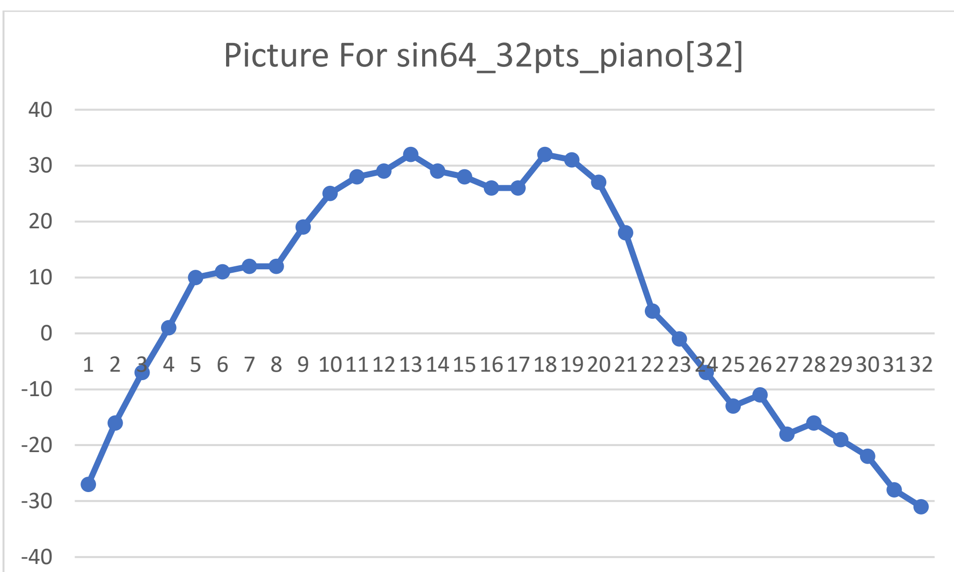
$\boxed{\text{songbeat}}$

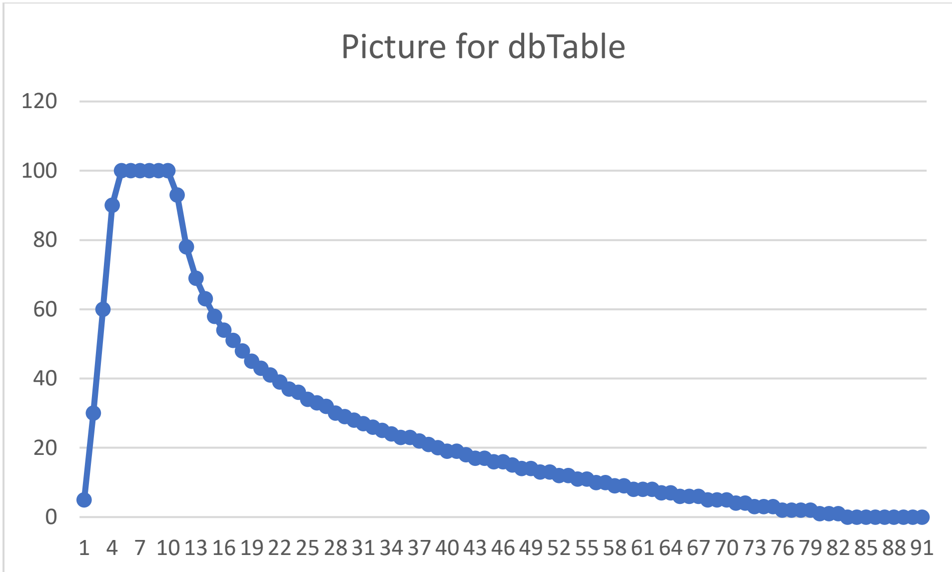
└→ notes in this beat (0~4) (integer)

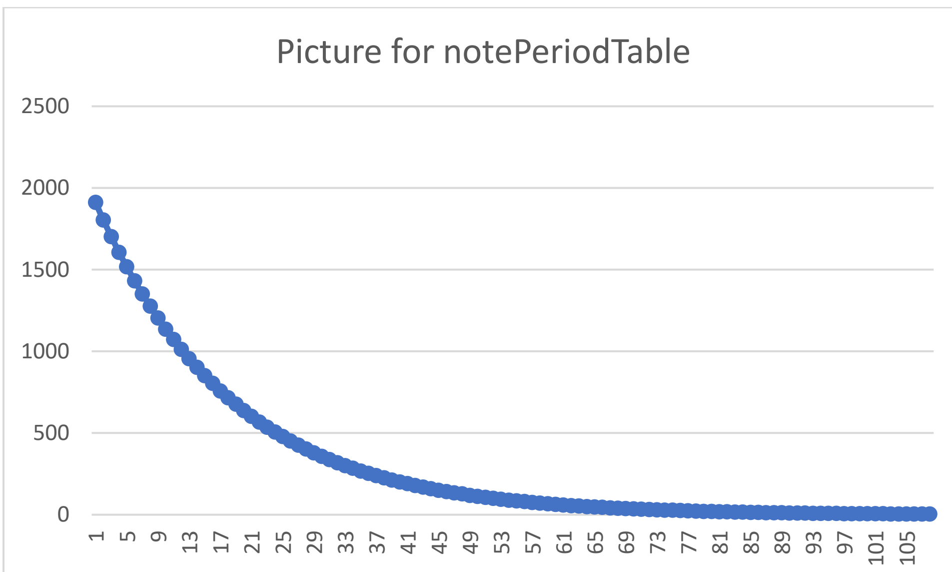
└→ notes[4] (integer)

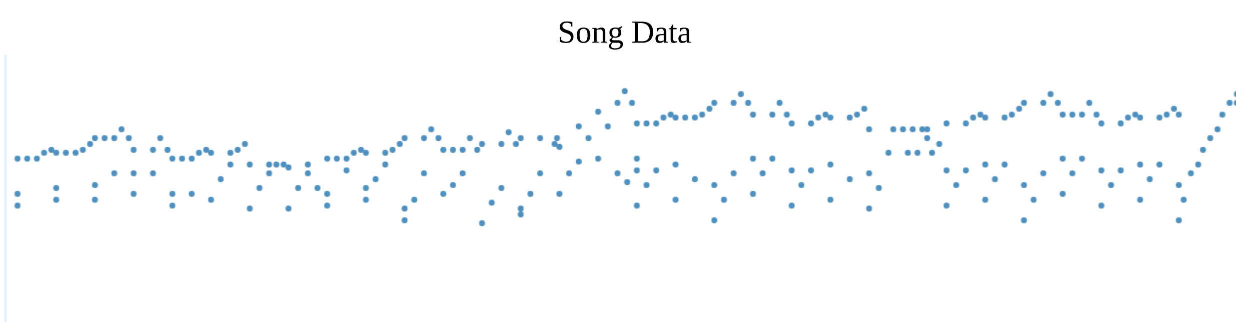
A song is made up of an array of equally spaced song beats, which may be empty.



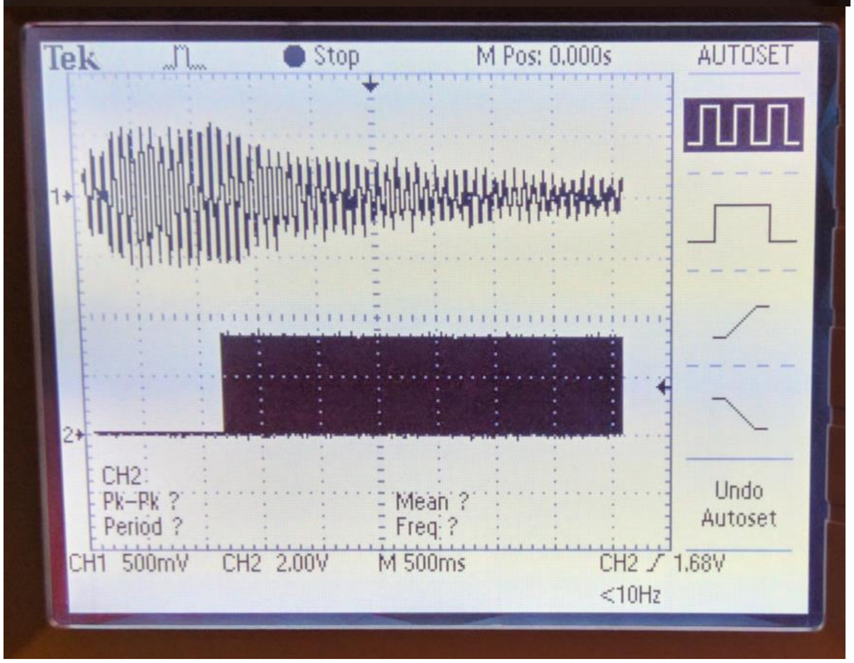








4. Picture of the Dual Scope



5. Measurement Data

Bit2	Theoretical V (Calculated)	Measured V (Using TexasDisplay)
0	0.0000	0.00
1	0.05241	0.05
2	0.10479	0.10
3	0.15716	0.16
4	0.20954	0.21
5	0.26192	0.27
6	0.31429	0.32

7	0.36667	0.37
8	0.41905	0.42
9	0.47142	0.48
10	0.5238	0.53
11	0.57618	0.59
12	0.62855	0.64
13	0.68093	0.69
14	0.73331	0.75
15	0.78568	0.80
16	0.83806	0.84
17	0.89044	0.90
18	0.94281	0.95
19	0.99519	1.00
20	1.04757	1.06
21	1.09994	1.11
22	1.15232	1.16
23	1.2047	1.22
24	1.25707	1.26
25	1.30945	1.32
26	1.36183	1.38
27	1.4142	1.43
28	1.46658	1.48
29	1.51896	1.54
30	1.57133	1.59
31	1.62371	1.64
32	1.67609	1.68
33	1.72846	1.71
34	1.78084	1.76
35	1.83322	1.82
36	1.88559	1.87
37	1.93797	1.92
38	1.99035	1.97
39	2.04272	2.03
40	2.0951	2.08
41	2.14748	2.13
42	2.19985	2.19
43	2.25223	2.24
44	2.30461	2.29
45	2.35698	2.35
46	2.40936	2.40
47	2.46174	2.45
48	2.51411	2.50
49	2.56649	2.55
50	2.61887	2.60
51	2.67124	2.66

52	2.72362	2.71
52	2.776	2.76
54	2.82837	2.82
55	2.88075	2.87
56	2.93313	2.92
57	2.9855	2.98
58	3.03788	3.03
59	3.09026	3.08
60	3.14263	3.13
61	3.19501	3.19
62	3.24739	3.23
63	3.29976	3.29

Resolution: $= \frac{3.29}{2^6-1} = 0.05222222222$
Range: 3.29
Precision: $Precision = Number\ of\ Alternatives = 64$
Accuracy: $= \frac{1}{n}\sum|x_n - x_{mi}| = 0.011031$

<u>6. Brief, One Sentence Answers:</u>
a. When does the interrupt trigger occur?

The interrupt trigger (the event that calls for the interrupt) occurs when the SysTick counts from 1 to 0.

b. In which file is the interrupt vector?

Startup.s.

c. List the steps that occur after the trigger occurs and before the processor executes the handler.

1. Current instruction finished.
2. Push 8 registers R0-R3, R12, LR, PC, PSR.
3. Set LR to 0xFFFFFFF9.
4. Set IPSR to ISR number.
5. Set PC to ISR address.

d. It looks like BX LR instruction simply moves LR into PC, how does this return from interrupt?

Since before the ISR is called, the LR is set to 0xFFFFFFF9, the computer knows that it is returning from an interrupt; the BX LR instruction, if meeting a LR that tells it to return from interrupt, will pop the registers and then restore original context.