**C13 DAC and Sound**

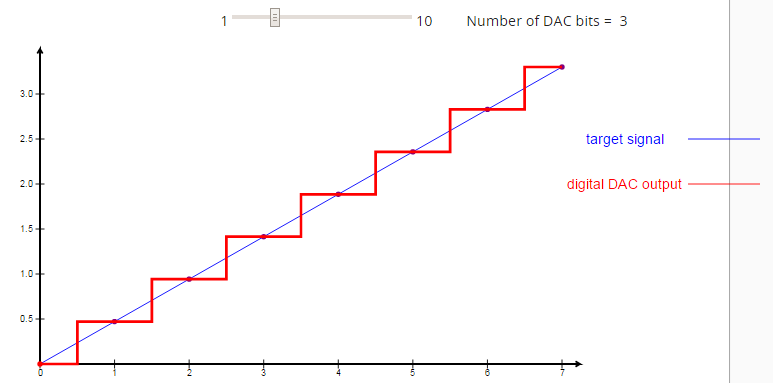
**C13.1 Approximating Continuous Signals in the Digital Domain**

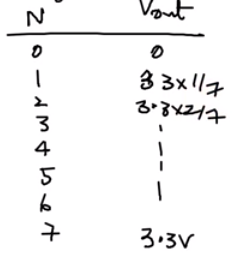
Can be approximated either with amplitude quantization or time quantization.

Nyquist theorem states that if you sample a signal at , then the digital samples only contain frequency components from 0 to 0.5 . Any frequencies above 0.5 would produce an aliasing error. Aliasing is when the digital signal appears to have a different frequency than the original analog signal.

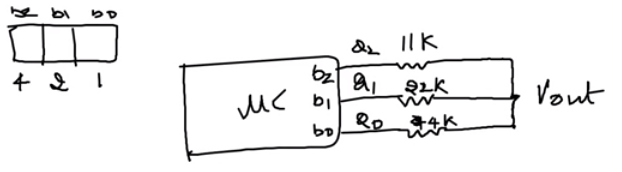
This means that if you are sampling a signal with a frequency of , you need to sample it at because you want the digital sample to contain components from 0 to 0.5 , which is 0 to .

**C13.2 Digital to Analog Conversion**

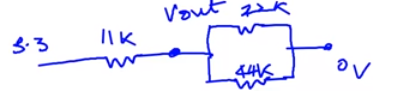
There are two types of circuits to build a digital to analog converter (DAC) – Binary Weighted Circuit and R-R Ladder. In this lesson, we focus on the Binary Weighted Circuit. Let’s say this is a 3 bit DAC, we should expect 8 levels of voltages:



We would then build the following circuit, which is a 3 bit circuit:

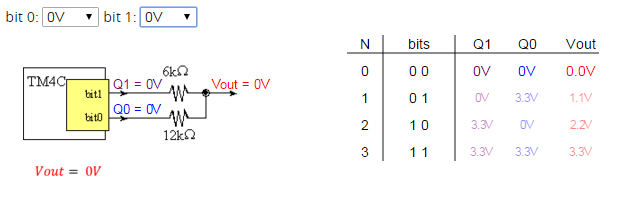


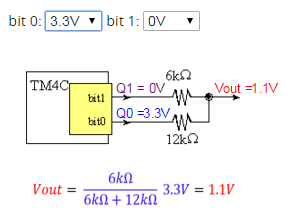
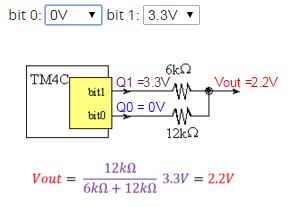
represent the bits, if we output a voltage at any of these pins, it would represent a 1 for that bit. For example, if we output a voltage at , then we get this circuit:



Solving for Vout, gives 3.3V \* 4/7.

**Binary Weighted DAC**



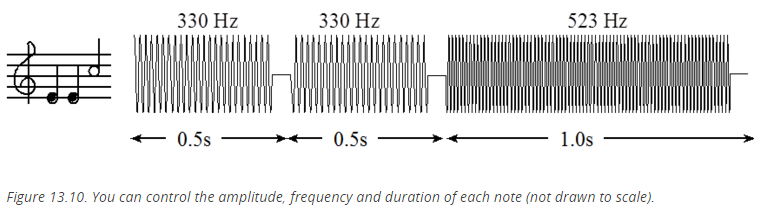
 

**C13.4 Sound as an Analog Signal: Loudness, Pitch and Shape**

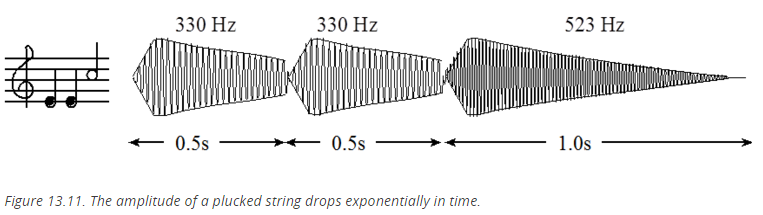
|  |  |
| --- | --- |
| Note | frequency |
| C | 523 Hz |
| B | 494 Hz |
| Bb | 466 Hz |
| A | *440 Hz* |
| Ab | 415 Hz |
| G | 392 Hz |
| Gb | 370 Hz |
| F | 349 Hz |
| E | 330 Hz |
| Eb | 311 Hz |
| D | 294 Hz |
| Db | 277 Hz |
| C | 262 Hz |

*Table 13.2. Fundamental frequencies of standard musical notes. The frequency for ‘A’ is exact.*

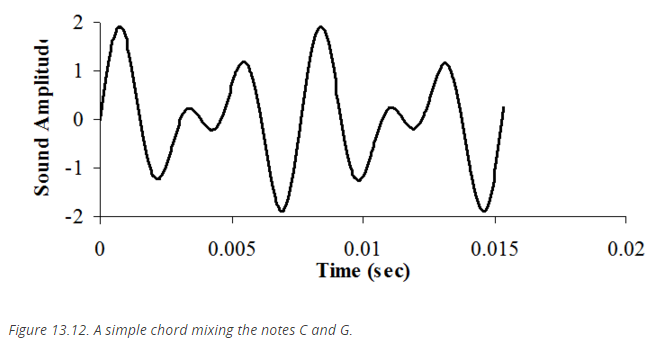
The **tempo** of the music defines the speed of the song. In 2/4 3/4 or 4/4 music, a **beat** is defined as a quarter note. A moderate tempo is 120 beats/min, which means a quarter note has a duration of ½ second. A sequence of notes can be separated by pauses (silences) so that each note is heard separately.



The **envelope** of the note defines the amplitude versus time relationship.



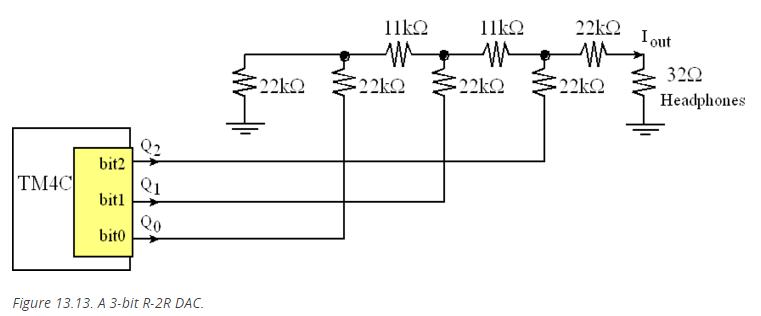
A **chord** is created by playing multiple notes simultaneously. When two piano keys are struck simultaneously both notes are created, and the sounds are mixed arithmetically. You can create the same effect by adding two waves together in software, before sending the wave to the DAC. Figure 13.12 plots the mathematical addition of a 262 Hz (low C) and a 392 Hz sine wave (G), creating a simple chord.



**R-2R Ladder DAC**

|  |  |  |
| --- | --- | --- |
| *N* | Q2       Q1       Q0 | *Iout* (μA) |
| 0 | **0    0    0** | 0.0 |
| 1 | **0    0    3.3** | 12.5 |
| 2 | **0    3.3  0** | 25.0 |
| 3 | **0    3.3  3.3** | 37.5 |
| 4 | **3.3  0    0** | 50.0 |
| 5 | **3.3  0    3.3** | 62.5 |
| 6 | **3.3  3.3  0** | 75.0 |
| 7 | **3.3  3.3  3.3** | 87.5 |

*Table 13.3. Specifications of the 3-bit R-2R DAC.*



const unsigned char SineWave[16] = {4,5,6,7,7,7,6,5,4,  
                                    3,2,1,1,1,2,3};  
unsigned char Index=0;           // Index varies from 0 to 15

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*DAC\_Init\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Initialize 3-bit DAC  
// Input: none  
// Output: none

void DAC\_Init(void){unsigned long volatile delay;  
  SYSCTL\_RCGC2\_R |= SYSCTL\_RCGC2\_GPIOB; // activate port B  
  delay = SYSCTL\_RCGC2\_R;    // allow time to finish activating  
  GPIO\_PORTB\_AMSEL\_R &= ~0x07;      // no analog  
  GPIO\_PORTB\_PCTL\_R &= ~0x00000FFF; // regular GPIO function  
  GPIO\_PORTB\_DIR\_R |= 0x07;      // make PB2-0 out  
  GPIO\_PORTB\_AFSEL\_R &= ~0x07;   // disable alt funct on PB2-0  
  GPIO\_PORTB\_DEN\_R |= 0x07;      // enable digital I/O on PB2-0  
}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*Sound\_Init\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Initialize Systick periodic interrupts  
// Input: interrupt period  
//        Units of period are 12.5ns  
//        Maximum is 2^24-1  
//        Minimum is determined by length of ISR  
// Output: none

void Sound\_Init(unsigned long period){  
  DAC\_Init();          // Port B is DAC  
  Index = 0;  
  NVIC\_ST\_CTRL\_R = 0;         // disable SysTick during setup  
  NVIC\_ST\_RELOAD\_R = period-1;// reload value  
  NVIC\_ST\_CURRENT\_R = 0;      // any write to current clears it  
  NVIC\_SYS\_PRI3\_R = (NVIC\_SYS\_PRI3\_R&0x00FFFFFF)|0x20000000;   
// priority 1  
  NVIC\_ST\_CTRL\_R = 0x0007; // enable,core clock, and interrupts  
}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*DAC\_Out\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// output to DAC  
// Input: 3-bit data, 0 to 7  
// Output: none

void DAC\_Out(unsigned long data){  
  GPIO\_PORTB\_DATA\_R = data;  
}

// the sound frequency will be (interrupt frequency)/(size of the table)  
void SysTick\_Handler(void){  
  Index = (Index+1)&0x0F;      // 4,5,6,7,7,7,6,5,4,3,2,1,1,1,2,3...   
  DAC\_Out(SineWave[Index]);    // output one value each interrupt  
}

void main(void){  
  PLL\_Init();          // bus clock at 80 MHz  
  Switch\_Init();       // Port F is onboard switches, LEDs, profiling  
  Sound\_Init(50000);   // initialize SysTick timer, 1.6kHz  
  while(1){ }  
}

Program 13.1. The periodic interrupt outputs one value to the DAC.

**Miscellaneous**

Let N be the size of the table, then the pitch will be the interrupt rate divided by N: