UTAustinX: UT.6.01x Embedded Systems - Shape the World
KarenWest (/dashboard) ▼

Courseware (/courses/UTAustinX/UT.6.01x/1T2014/courseware) **Course Info (/courses/UTAustinX/UT.6.01x/1T2014/info)**
Discussion (/courses/UTAustinX/UT.6.01x/1T2014/discussion/forum) **Progress (/courses/UTAustinX/UT.6.01x/1T2014/progress)**
Questions (/courses/UTAustinX/UT.6.01x/1T2014/a3da417940af4ec49a9c02b3eae3460b/)
Syllabus (/courses/UTAustinX/UT.6.01x/1T2014/a827a8b3cc204927b6efaa49580170d1/)
Embedded Systems Community (/courses/UTAustinX/UT.6.01x/1T2014/e3df91316c544d3e8e21944fde3ed46c/)

In each of the three test cases, the current across the active switch is $I_0=V_{ref}$ / (3R). This current is divided by 2 at each branch point. I.e., $I_1 = I_0/2$, and $I_2 = I_1/2$. Current injected from the lower bits will be divided more times. Since each stage divides by two, the exponential behavior is produced. An actual DAC is implemented with a current switch rather than a voltage switch. Nevertheless, this simple circuit illustrates the operation of the R-2R ladder function. When the input is 001, V_{ref} is presented to the left. The effective impedance to ground is 3R, so the current injected into the R-2R ladder is $I_0=V_{ref}$ / (3R). The current is divided in half three times, and $I_{001}=V_{ref}$ / (24R).

When the input is 010, V_{ref} is presented in the middle. The effective impedance to ground is still 3R, so the current injected into the R-2R ladder is $I_0=V_{ref}$ / (3R). The current is divided in half twice, and $I_{010}=V_{ref}$ / (12R).

When the input is 100, V_{ref} is presented on the right. The effective impedance to ground is once again 3R, so the current injected into the R-2R ladder is $I_0=V_{ref}$ / (3R). The current is divided in half once, and $I_{100}=V_{ref}$ / (6R).

Using the Law of Superposition, the output voltage is a linear combination of the three digital inputs, $I_{out}=(4b_2 + 2b_1 + b_0)V_{ref}$ / (24R). A current to voltage circuit is used to create a voltage output. To increase the precision one simply adds more stages to the R-2R ladder.

To generate sound we need a table of data and a periodic interrupt. Program 13.1 shows C code that defines a 16-element 3-bit sine wave. The frequency of the sound will be the interrupt frequency divided by 16 (size of the table). So, to create a 100 Hz wave we need SysTick to interrupt at 16*100 Hz, 1600 Hz. If the bus clock is 80 MHz, then the initialization should be called with an input parameter of value 50000. The `const` modifier will place the data in ROM. The interrupt software will output one value to the DAC. See Figure 13.15. In this example, the 3-bit DAC is interfaced to output pins PB2-0. To output to this DAC we simply write to Port B. In order to create the sound, it is necessary to output just one number to the DAC upon each interrupt. The DAC range is 0 to 87.5 μA.

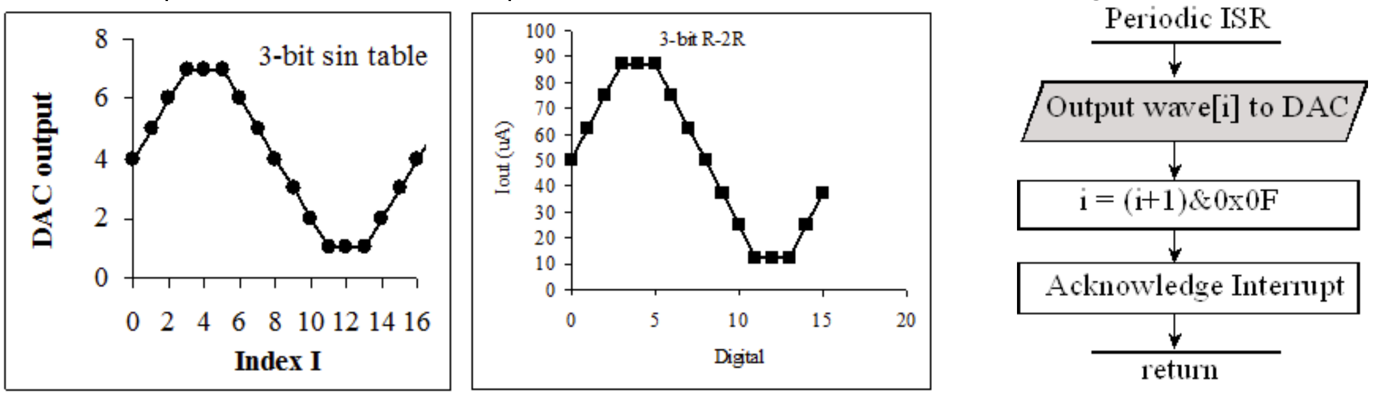


Figure 13.15. A DAC and a periodic interrupt are used to create sound. Output is discrete in time and voltage.

VIDEO 13.5C

C13_Video5c Show software R2R 3bit YouTube

6:03 / 6:03 1.0x

DR. JONATHAN VALVANO: Let's look at the software

needed to create the sound wave.

We saw the wave is periodic.

And we're going to use a data structure which contains

the numbers 4, 5, 6, 7, 7, 7, 6, 5, 4, 3, 2, 1, 1, 1, 2, 3.

This data structure, this array, is going

to contain the numbers which represent the shape of my wave.

We'll have an index which varies from 0 to 15 as we index into this array.

And the output is going to be the DAC, as a function of this index,

is going to represent the shape.

4, 5, 6, 7, 7, 7, 6, 5, 4, 3, 2, 1, 1, 1, 2, 3.

And then what will happen is the wave will repeat over and over again,

4, 5, 6, 7, 7, 7, 7.

DR. RAMESH YERRABALLI: OK, so let's take a look at the initialization of this DAC.

First, we'll look at the port initialization.

DR. JONATHAN VALVANO: And so the DAC initialization

will produce 3-bit output on port PB2, PB1, PB0.

And these three bits are connected to our DAC that we saw in the last video.

DR. RAMESH YERRABALLI: OK.

And now we will look at the initialization for the SysTick

periodic interrupt.

DR. JONATHAN VALVANO: We've used SysTick before.

The important thing for us to remember is this reload value

here will specify the rate at which SysTick interrupts.

And this will be an important design parameter for our system.

DR. RAMESH YERRABALLI: Next, let's look at the interrupt service routine.

So the SysTick handler is the interrupt service routine,

which gets executed every time the SysTick interrupt occurs.

DR. JONATHAN VALVANO: So recall that what we want to create

is this wave, 4, 5, 6, 7, 7, 7, 6, 5, 4, 3, 2, 1, 1, 1, 2, 3.

And then it will repeat over and over again.

This is our DAC output.

But now what we're going to do is change this access to time.

And we're going to do that by generating a SysTick interrupt.

And in the SysTick interrupt, I am going to do one DAC output.

And if I do exactly one DAC output, if I write to port B

exactly once per interrupt, what'll happen

is the time between these two numbers is going

to be a function of the period of the SysTick interrupts.

And so we can see that every 16 SysTick interrupts,

we're going to have one cycle of our sound wave.

We've got to finish and update our Index so that we're

pointing to the next entry of the table.

And then we quit.

So right here, we see we're going to increment the Index.

And when it gets to 16, we'll roll it back over.

So the Index goes 0, 1, 2, 3, up to 14, 15.

next time, it'll be 0 again.

So in summary, we're going to produce one output to the DAC every interrupt.

And we're going to control the frequency, the pitch, of this sound

by adjusting the rate at which we SysTick our interrupts.

DR. RAMESH YERRABALLI: So what about a plan for testing this, Jon?

DR. JONATHAN VALVANO: Real important.

We're going to do two types of tests.

The first is a heartbeat.

And a heartbeat is a minimally-intrusive debugging instrument.

Every time we're going to get an interrupt,

we're going to toggle port F, bit 3.

And so if we looked along this curve, we're going to see port F, bit 3

toggle every time there's an interrupt.

So in this way, we can see whether our software is living, or is it dead.

The second instrument is a non-intrusive.

We're going to use the oscilloscope and actually look at the DAC wave,

and see if we get this picture.

DR. RAMESH YERRABALLI: So let's look at the main, then.

DR. JONATHAN VALVANO: The main program initializes our devices.

And if the switch is pressed, what we'll do

is enable interrupts and set up the SysTick so

that it generates a periodic interrupt.

And if the switch is not pressed, we'll disable

interrupts, which will stop the sound.

Let's see how we calculate this number 50,000.

We know we want 100 Hertz sine wave.

And we know that there are exactly 16 outputs

of the DAC per one cycle of the sine wave, the output wave.

And we know we have an 80 Megahertz bus clock that's controlling SysTick.

And if we perform this calculation, we will

get the 50,000 used in this program.

DR. RAMESH YERRABALLI: OK, so we've designed everything.

So let's build the system.

DR. JONATHAN VALVANO: All right, let's build it.

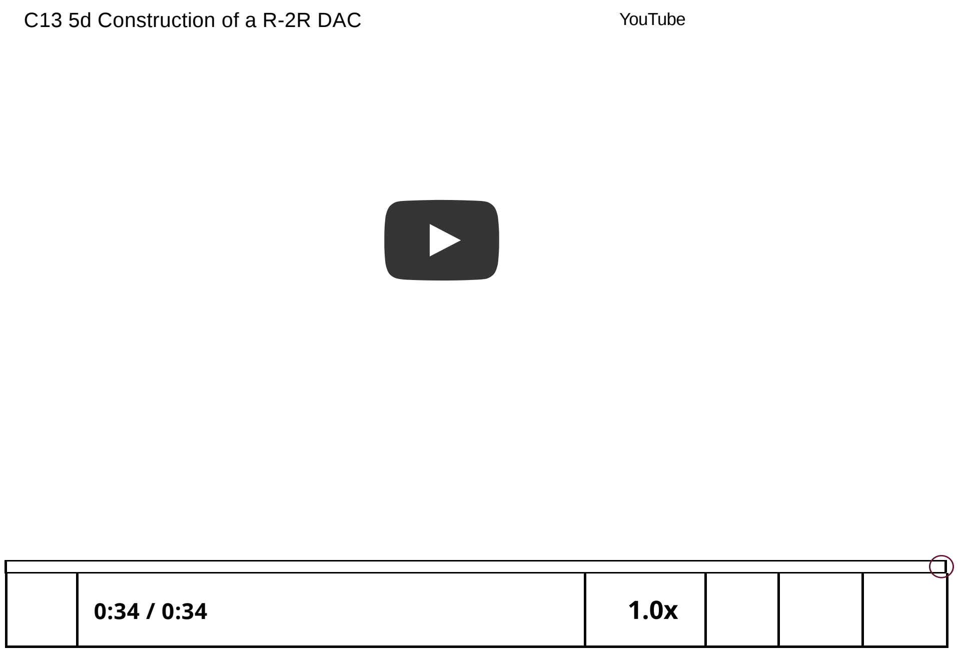
Help

```
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.

VIDEO 13.5D. CONSTRUCTION OF THE DAC CIRCUIT



DR. JONATHAN VALVANO: Let's build an R-2R ladder.

This DAC can be built with a pile of resistors of the same value.

Each of the legs along the bottom is 22k.

Each 11k arm across the top can be made by placing

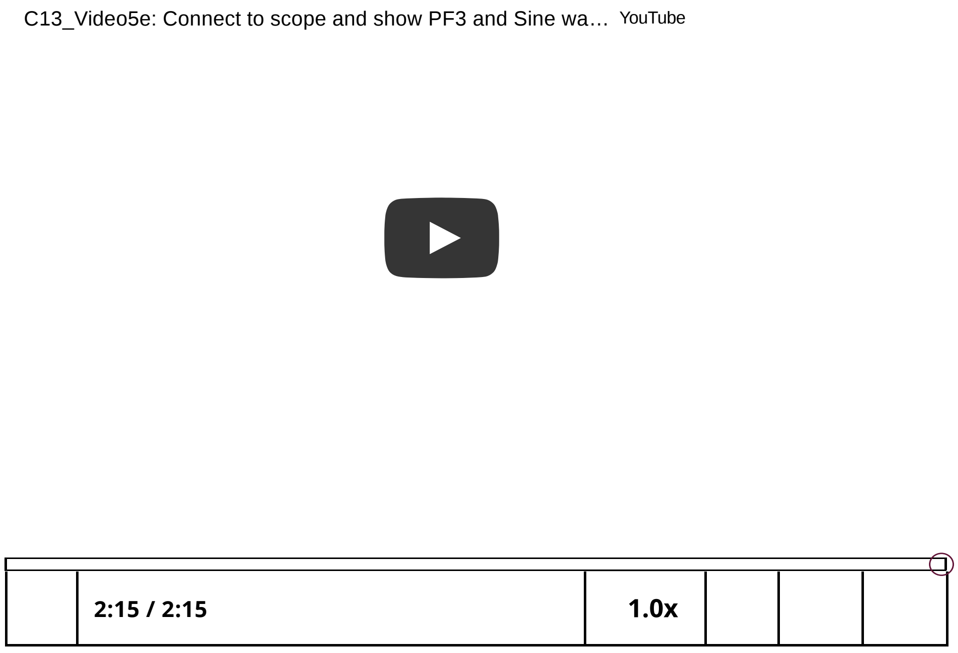
two 22k resistors in parallel.

Next we connect the headphone jack, and lastly, we

attach it to the micro controller.

Remember to connect up the ground.

VIDEO 13.5E. DEMONSTRATION OF THE SOLUTION USING A SCOPE



DR. JONATHAN VALVANO: We have written the software, downloaded

onto the launch pad, and we have built the R2R ladder.

Next we're going to debug it.

What we have is one channel of the logic analyzer

connected here to the DAC output, and we have the second channel

of the logic analyzer connected to the

Help