Cypress MicroSystems

Application Note AN2024

Polyphonic Piano

By: David Van Ess
Associated Project: Yes
Associated Part Family: CY8C25xxx, CY8C26xxx

Summary

A polyphonic piano can be created using the analog features of the PSoC™ microcontroller and leveraging the hardware Multiply/Accumulate (MAC) capability to create innovative digital processing algorithms.

Introduction

To demonstrate the configurable nature of the PSoC MCU and how the blocks can be used in creative ways, a 10-Voice Polyphonic Piano was implemented.

Specifications include:

- 8 x 5 Scan Matrix
- 37 Keys (From C 131Hz to C1047 Hz)
- Intelligent Key Scan Algorithm
- Key Debounce
- 10-Voice Polyphonic
- Sounds Stored in ROM Table
- Algorithmic Volume Envelope

This application uses the following PSoC microcontroller resources:

- 2665 Bytes Flash
- 104 Bytes RAM
- 16-Bit Counter=2 Digital PSoC Blocks
- 8-Bit DAC=1 PSoC Analog Block
- SC Inverter=1 PSoC Analog Block
- Multiply/Accumulate (MAC)

A Five Mile View

At the highest level, the software to control this application has two primary functions:

- Scan the keyboard,
- Output the collected notes to the DAC at an update rate of 15 kHz.

Figure 1 shows that control of a 5 by 8 keyboard matrix requires 13 GPIO pins. The DAC and Inverter each require a buffered analog output for a total of 15 pins. For this application, the 28-pin CY8C25443 was used, however it could have been implemented with the 20-pin C48C26233.

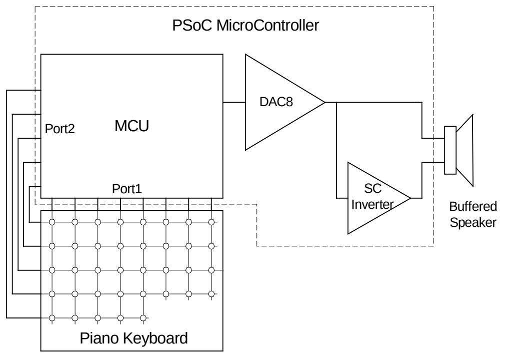

Figure 1: High Level Block Diagram

The Control Software

Timers are used to set up interrupts to handle two different processes:

- Generate the signal to be output by the DAC,
- Scan keyboard to determine which keys have been pressed or released.

Figure 2 shows the structure and process flow.

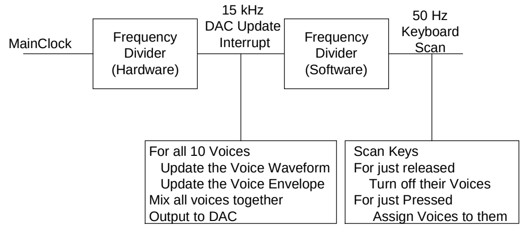

Figure 2: Process Flow Diagram

The keyboard scan determines which of 37 keys have been pressed/released. The update routine assembles the signal to be sent to the DAC. There is not enough CPU time to calculate all the signals for 37 keys. So, 10 voices were implemented along with control software to assign or unassign a particular key to a particular voice. Each voice is a data object that contains:

- The key assigned to the voice.
- The frequency assigned to the voice.
- The control status of the voice.

This technique should be called, “Making a 37-key piano function for a 10-fingered piano player.” However, a more technical term is a “10-Voice Polyphonic Piano.”

Sound Production

The time domain characteristics of a piano sound were analyzed and separated into two parts:

- A cyclic waveform that defines the primary piano sound.
- An envelope that modulates the cyclic waveform to simulate the attack and sustain of a piano key.

These waveforms are shown in Figures 3 and 4.

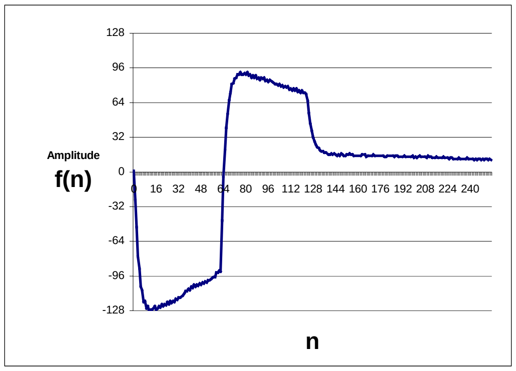

Figure 3: Cyclic Piano Waveform

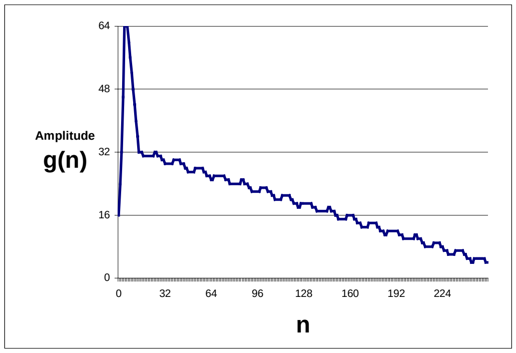

Figure 4: Piano Modulation Envelope

While the frequency of the cyclic waveform is set by the frequency of the desired key, the sustain lasts about 1¼ seconds for all keys. The product of these two signals produces a relatively good piano sound.

Equation (1) shows how the signal sent to the DAC is assembled from all 10 voices.

$$DACValue = \sum_{i=0}^{9} Envelope_i * Waveform_i \quad \textbf{(1)}$$

This algorithm is efficiently implemented with the dedicated onboard Multiply/Accumulate (MAC). The value of the *Envelope* and *Waveform* are each stored in a 256-byte ROM table. Equations (2) and (3) show how each value is determined.

$$Envelope_i = g(Epointer_i) \quad \textbf{(2)}$$
$$Waveform_i = f(Wpointer_i) \quad \textbf{(3)}$$

Additional envelope and waveform ROM files can be included to add other instruments. However, each file will take an additional 256 bytes.

If the *Epointer* is incremented by 1 at the 15 kHz update rate, the cyclic frequency generated for that voice is:

$$58.5934\,Hz = \frac{15KHz}{256}$$

A particular desired frequency is obtained using a particular increment value. Equation (4) shows how to calculate the increment value, given a desired frequency.

$$pointerdel = \frac{DesiredFrequency}{58.5934} \quad \textbf{(4)}$$

To generate A440 (440 Hz), an increment value of 7.5094 is required. A table of the 37 different keys is stored in ROM. Each note value has 16-bit resolution, allowing the frequencies generated to have an error less than 0.1 Hz. Two envelope rates are also stored. “Sustain” is the value needed for a 0.8 Hz (1.25 sec) tone length and “Release” is the value needed to quickly finish the note when the key has been released.

Putting the Whole Thing Together

Each voice has the follow variables:

VoiceStatus	Either value of key assigned *or* 255 if voice is not assigned
Wpointer	Pointer to the Waveform ROM table
Wpointerdel	Frequency being generated
Epointer	Pointer to the Envelope ROM table
Epointerdel	Either the **Sustain** or **Release** rate
Waveform	Calculated value of the waveform
Envelope	Calculated value of the envelope

During the Update Interrupt the following functions are performed:

```
for(i=0to 9){              //all voices
  if(VoiceStatus[i]==off){ //voice off
      Envelope[i] = 0      //no signal
      Waveform[i] = 0
  }
  else{
  Epointer[i] += Epointerdel[i]
  if(Epointer[i]overflows){
             // envelope complete
     VoiceStatus[i]=off
             // release finished voice
     Envelope[i]=0
             // no signal
     Waveform = 0
   }
   else{
     Envelope[i] = g[Epointer[i]]
     Wpointer[i] += Wpointerdel[i]
     Waveform[i] = f[Wpointer[i]]
    }
  }
}
Accum = 0         //clear accumulator
for(I = 0 to 9){  //all voices
   Accum += (Envelope[i] * Wavefrom[i])
}
DACValue = Accum
```

During the Scan Interrupt, the following functions are performed:

```
Scan the keyboard
Determine keys to turn off
Determine key to turn on
// Turn off Keys
for(i = 0 to 36){      // all keys
   if( key[i] = to be turned off){
      for( j = 0 to 9 ){  // all Voices
         if(key[i] == VoiceStatus[j]){
            Epointerdel[j] = Release
         }
      }
   }
}
//Turn on Keys
for(I = 0 to 36){     //all keys
   if(key[i] is to be turned on){
      j = AvailableVoice
      if( j is a valid voice){
         Epointer[j] = 0
         Epointerdel[j] = Sustain
         Wpointer[j] = 0
         Wpointerdel[j] = KeyTable[i]
        VoiceStatus[j]=i
      }
   }
}
```

Future Options

Multiple Instrument Sounds

The sound template is stored in a ROM table. Multiple tables could be created and selected by a switch giving the keyboard multiple instruments capability. Up to 50 tables could be supported.

Pitch Control

Presently, the update rate is set to 15 kHz by dividing 48 MHz by 3,200. A pitch-control knob could be added to allow adjusting the pitch of the instrument. Changing pitch requires only changing the divisor.

Elimination of the Crystal

The PSoC chip includes a +/-2.5% Internal Main Oscillator. The frequency of the oscillator can be adjusted under control of the CPU. A simple calibration procedure could be developed to use during keyboard manufacturing. A reference frequency could be applied to an input of the PSoC chip. Firmware in the PSoC MCU would compare the internal frequency to the external reference frequency, calculate the difference, and store an adjustment value in the Flash memory. Upon power-on, the adjustment value would be copied to the Main Oscillator Trim register, providing an accurate on-chip oscillator. This would eliminate the cost of the external 32.768 kHz crystal.

Different Number of Voices

With a slower update rate, more voices can easily be added; but with some loss of fidelity. And, of course, fewer voices also can be implemented allowing the CPU to operate at a lower speed.