Chapter 25O

Online Content: Microcontrollers IV Creating PlaySong() Data

25O.1 What is PlaySong?

The PlaySong function provided in the skeleton code outputs a series of single frequency notes through the DAC for specified periods of time to play a song.

25O.1.1 How do I tell PlaySong what song to play?

PlaySong takes takes a pointer to a song structure which contains the song to play:[1]

```
void PlaySong(struct Song *song)
```

25O.1.2 Ok. But what is a Song structure?

The `Song` structure is a C structure containing five elements;

```
struct Song {
    const uint16_t* notes;     // a pointer to an array of n counter match values
    const uint32_t* htime;     // a pointer to an array of n Delay() times
    const uint32_t num_notes;  // number of elements in the notes and htime arrays
    const char name[16];       // name to display while playing a song (8 chars max)
    const int tempo;           // 0 = normal; 1 = 50% slower; 2 = half speed
};
```

[1] PlaySong() was created as a quick and dirty hack to demonstrate the SAMD21 DAC and timer interrupt. The author of PlaySong() knows nothing about music that he did not read in Wikipedia. So if you are looking for Mozart, you are in the wrong place. Also, the timer divider must be an integer, meaning the songs may drive you nuts if you have perfect or even reasonably good pitch.

The notes array consists of a set of SAMD21 timer/counter match values which are defined to correspond with notes on the treble clef from B4 through G6. Each note is defined as the nearest integer value to its actual frequency. There is also a rest/silent value ("QUIET") created by setting the match value to 0xffff, a frequency too low to be heard (less than 6Hz).

```
// note definitions (48MHz/128 = 375,000)
#define G6_NOTE ((375000/1568)-1)  // G6
#define F6_NOTE ((375000/1397)-1)  // F6
#define E6_NOTE ((375000/1318)-1)  // E6
#define D6_NOTE ((375000/1175)-1)  // D6
#define C6_NOTE ((375000/1040)-1)  // C6
#define B5_NOTE ((375000/988)-1)   // B5
#define A5_NOTE ((375000/880)-1)   // A5
#define G5_NOTE ((375000/783)-1)   // G5
#define F5_NOTE ((375000/698)-1)   // F5
#define E5_NOTE ((375000/659)-1)   // E5
#define D5_NOTE ((375000/587)-1)   // D5
#define C5_NOTE ((375000/523)-1)   // C5
#define B4_NOTE ((375000/494)-1)   // B4
#define QUIET   0xffff             // 5Hz - effectively silent note
```

The htime (hold time) array consists of a set of values in mSec for the duration of the corresponding note:

```
// note duration definitions
#define BRK  50             // time between notes (mSec)
#define QTN  350            // time for quarter note (mSec)
#define QTNH (QTN * 1.5)    // quarter note held
#define HFN  (QTN * 2)      // half note is twice as long as quarter note
#define HFNH (QTN * 3)      // half note held
#define WHN  (QTN * 4)      // whole note
#define ETN  (QTN / 2)      // eighth note
```

The hold time array must have the same number of elements as the notes array. Note `notes[x]` is played for `htime[x]` mSec. A fixed 50mSec pause (`BRK`) is inserted after each note is played. The normal tempo may be changed by changing the duration of the quarter note as all other notes are defined as a multiple of it.

`num_notes` is the number of notes in the notes array. You should define it as the size of the notes array divided by the number of bytes per note and the C compiler will do the calculation for you.

The name array holds the name of the song as a null terminated string. Only the first eight characters will be displayed on the LCD display in the last microcontroller lab. However, it is declared as 16 bytes to provide room for the terminating null. The Chapter 25L `PlaySong()` function prints the name of each song on the debug console as it starts to play.

The last item lets you slow down the tempo by optionally increasing the duration of each note by 50% if set to 1, or 100% if set to 2.

25O.1.3 Hum. Sounds complicated. How about an example?

Ok, lets create a Song structure for *The Itsy Bitsy Spider*.[2] First find some sheet music for the song you want to transcribe (and annotate it with the notes if, like me, you don't read music):[3]

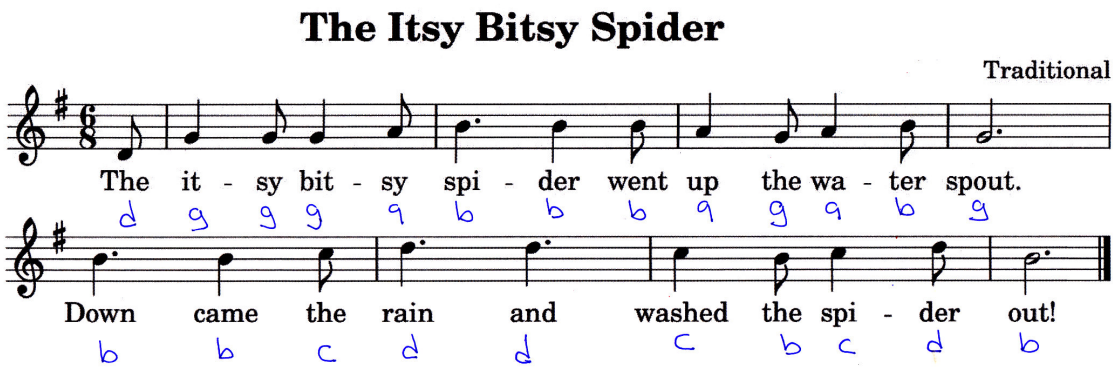

Figure 1: The Itsy Bitsy Spider (annotated)

Next transcribe the music into the `notes[]` and `htime[]` arrays. Every song should end with a one second pause.

```
const uint16_t spider_notes[] = {
D5_NOTE, G5_NOTE, G5_NOTE, G5_NOTE, A5_NOTE, B5_NOTE, B5_NOTE, B5_NOTE,
A5_NOTE, G5_NOTE, A5_NOTE, B5_NOTE, G5_NOTE, QUIET,
B5_NOTE, B5_NOTE, C6_NOTE, D6_NOTE, D6_NOTE, C6_NOTE, B5_NOTE, C6_NOTE,
D6_NOTE, B5_NOTE, QUIET, QUIET};

const uint32_t spider_htime[] = {
ETN, QTN, ETN, QTN, ETN, QTNH, QTN, ETN, QTN, ETN, QTN, ETN, HFNH, QTN,
QTNH, QTN, ETN, QTNH, QTNH, QTN, ETN, QTN, ETN, HFNH, 1000};
```

Let the C compiler calculate the number of notes in the song:

```
#define SPIDER_NUM_NOTES sizeof(spider_notes) / sizeof(spider_notes[0])
```

Finally, create a Song structure to tie the notes array, the duration array, the song name (8 chars max) and the tempo together. I usually start with the normal tempo and adjust as necessary to make the song sound right.[4]

```
struct Song Spider = {.notes = &spider_notes, .htime = &spider_htime,
                      .num_notes = SPIDER_NUM_NOTES, .name = "Spider", .tempo = 0};
```

To test your new song, copy the `notes[]` array, the `htime[]` array, the number of notes calculation and the Song structure definition to the §25L.3 test program and add a second call to PlaySong() with a pointer to your new Song structure to your "Play a Song" while() loop:

---

[2] Code available at `https://LAoE.link/micro4/25O_PlaySong_Data.c`

[3] I created this sheet music in LilyPond (`https://LAoE.link/LilyPond.html`). Perhaps some enterprising reader somewhere will write a LilyPond .ly file to PlaySong code generator and send it to us.

[4] But then I have already admitted I don't know anything about music.

```
/*************** Play a Song *****************/
  while(1) {
    PlaySong(&Mary);
    PlaySong(&Spider);
  }
/*********************************************/
```

Please send comments or corrections to: authors@LAoE.link