Pues si

PLAYSONG LIBRARY V0.1

Foenix C256U / c256U+ / FMX

https://github.com/econtrerasd/VickyGraph

Table of content

REQUIREMENTS

The VickyGraph library requires:

1. A Foenix U, Foenix U+ or Foenix FMX computer
   (It's meant to be portable to any Foenix computer supporting Vicky II)
2. The Calypsi Compiler for the WDC 65816 target

   https://www.calypsi.cc/

3. Calypsi's Foenix 256 board support (also available on the previous link)

INSTALLING THE LIBRARY

The playSong library can be obtained from the following GitHub

https://github.com/econtrerasd/playSong

Get the Zip file and uncompressed the whole directory in a subdirectory, the resulting files are based on Calypsi's "Hello World for c256U" that already include the Foenix C256 board support.

If you already have Calypsi Installed, you can go to the unpacked directory and build the project by typing

```
make play.pgz
```

to compile the demo.

DESCRIPTION

This "playSong" library provides a set of functions to:

1. Initialize the SID chip
2. Parse strings that contain music notation, using a format inspired on what Basic v7 provided for the C64 and store them in (proprietary) Tracker format in memory

3. Show the Track(s) encoded in memory
4. Play the Tracks of song contained in memory using the SID chip in the Foenix C256.

Motivations to create this library:

- Build a simple Music system that anyone can integrate into their programs when using Calypsi C
- Understand the workings of the SID chip when used for Music
- Understand how Trackers work by building the basis of a very simple Tracker

Possible Improvements for next versions (in no specific order)

1. Implement support for other sound chips used in the Foenix C256
    a. TI SN76489
    b. Yamaha OPL3
2. Work on implementing more advanced SID effects in instruments such as:
    a. Filters (High, Band, Low)
    b. ring modulation
3. Support more musical improvements such as:
    a. Portamento
    b. Vibrato
    c. Pitch Bend notes
    d. Arpeggio chords to play chords using only one voice
4. Include an Interrupt / Timer mechanism to implement playing Music as a background process.
5. Implement save file formats (formats proposed below)
    a. Song file - include all music tracks in String Music format (this would produce a very compact file!, but would need to be parsed first by the library before playing)
    b. Binary Track Format – Ready to play music file, could be loaded directly into memory and just play it by providing the memory address as a parameter.

INTRODUCTION TO THE SID CHIP

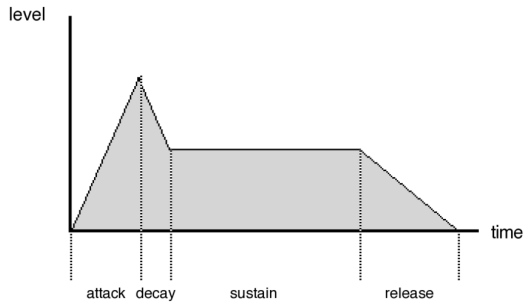
There are multiple resources to read about the capabilities and history of the SID chip, as a primer I recommend reading its Wikipedia article at the following link:

https://en.wikipedia.org/wiki/MOS_Technology_6581

Now that you know a bit more about the SID Chip, lets focus on what you need to understand to create a simple sound using the SID chip.

Steps to create a simple sound in the SID.

1. **Set the Volume** – Please consider that there is only one volume selectable (0-15) that functions as the main volume for all 3 voices.
2. **Set the Waveform Envelope Parameters** – An Envelope can be defined by providing parameters (in the range 0-15) for each of the *Attack, Decay, Sustain* and *Release* parts of the Envelope



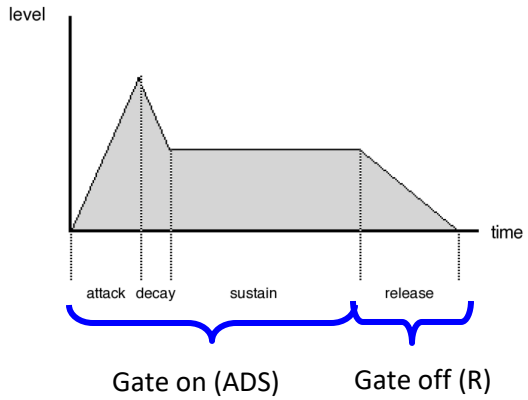
Envelope controls: Attack / Decay / Sustain / Release

3. **Select the Sound Frequency** – Provide the Low byte & High byte of the desired Sound frequency
4. **Select a Waveform** – The SID chip would produce a base sound using either a Triangle (sine like), Sawtooth, Pulse (Square) or Noise waveform. These waveforms are selected by setting individual bits in the SID control register.

   Please note that to initiate a sound the waveform values must be set along with the 'Gate Bit' bit (bit 1) of the control register

SID Chip waveforms

5. **Trigger on the Gate Bit –** Aside from the bits to select the waveform, **you need to set** *bit 1 of the* control register to trigger the selected waveform into reproducing sound. This trigger executes the *Attack, Decay and Sustain* parts of the Envelope.



6. **Provide a Delay –** After you trigger the Gate Bit, the sound will move from the Attack, Decay and Sustain part of the Envelope, it takes some time for the sound to move through these parts of the Envelope, depending on how you configured these parameters. Typically long attack/decay values would require a bit more time for the sound to complete this cycle.

   The program the produces sound must (patiently) wait for the sound to complete before initiating the daly.

   For music typically the shortest delay represents a 1/16th note or rest, so that it syncs with the fastest note or rest.

   Not all instruments have notes that necessarily complete their Attack/Decay/Sustain cycle in 1/16th duration, for example: if using an organ, when you play a note it moves almost instantaneously from the *Attack/Decay* portion directly to the *Sustain*, this sustain will keep playing for the whole duration of the note (not just 1/16th) until you release the note, and then it will go into *Decay* and finish almost instantaneously.

   Music played on the SID must take this into account so that it waits enough time for the sound to develop before turning off the Gate Bit, or else the sound will seem to be cut in the middle.

7. **Trigger off the Gate Bit –** After you wait enough time for your waveform to go through the *Attack, Decay & Sustain* you trigger off the Gate Bit, this causes the Envelope to enter the *Release* phase, depending on the value provided for release it will produce a fast or slow fade out of the current played sound.

   Please note that the waveform selection bits must be provided at the same time as the Gate Bit off state

Warning: Note that no other sound can be triggered in if you don't Gate off the previous note

BOB YANNES, creator of the SID CHIP summed up this process briefly during an Interview in August 1996: ***"The Envelope Generator was simply an 8-bit up/down counter which, when triggered by the Gate bit, counted from 0 to 255 at the Attack rate, from 255 down to the programmed Sustain value at the Decay rate, remained at the Sustain value until the Gate bit was cleared then counted down from the Sustain value to 0 at the Release rate."***

If you are interested, the complete interview can be found on the following link:

http://sid.kubarth.com/articles/interview_bob_yannes.html

MUSIC FORMAT

This library receives Music encoded in a string, using the following conventions for Notes, Rests and Configuration parameters

Note Syntax

Allows you to select the next note played in the current track, and optionally its octave and duration.

{A-G *(Note)*} [#|$ *(Modifier)*] [1-7 *(Octave)*] [whqes. *(Duration)*]

Note code	Note
A	La
B	Si
C	Do
D	Re
E	Mi
F	Fa
G	Sol

Modifier Code	Note Qualifier
#	Sharp
$	Flat

Duration code	Description
w	Whole
h	Half
q	Quarter
t	Eight
s	Sixteenth
.	Adds one half to current duration

Please respect the Upper case or Lower case of the codes as shown

Required and Optional Parameters

- Required Parameters are specified by regular brackets {}
- Optional Parameters are specified by square brackets []. If you do not use optional parameters default values are applied.

Parameter	Default Value
Octave	4
Sharp – Flat	If omitted, base note considered
Duration	One Quarter duration - *q*

Note Examples

Note Code	Interpretation
C#3h	C Sharp on the 3rd Octave with 2/4 duration
F	F on the 4th Octave with a 1/4 duration
D$5w.	Flat D on the 5th Octave with 6/4 duration

Rest syntax

Allows you to set a pause or rest and its duration in the current track.

{R} [whqes. *(Duration)*]

Duration code	Description
w	Whole
h	Half
q	Quarter
t	Eight
s	Sixteenth
.	Adds one half to current duration

Please respect either the Upper case or Lower case of the codes as shown

Required and Optional Parameters

- Required Parameters are specified by regular brackets {}
- Optional Parameters are specified by square brackets []. If you do not use optional parameters default values are applied.

Parameter	Default Value
Duration	One Quarter duration - *q*

Rest Examples

Rest Code	Interpretation
R	1/4 Duration rest
R.	3/8 Duration rest
Rh.	3/4 Duration rest

Configuration parameters

Volume

Specifies a volume change in the current Track , when using the SID to play the song keep in mind that the volume changes for all three SID voices

{V}{1-f *(Hex value)*}

Volume value	Description
0-9	Volume 0 to 9
a-f	Volume 10 to 15

Volume values are expressed in hexadecimal to use only one character

Octave

All subsequent notes played if they do not specify an Octave in the Note syntax will use this Octave

{O}{1-7 *(Value)*}

Octave value	Description
1-7	Octave 1 to 7

Instrument

Changes the Instrument used in the current track, all subsequent notes of the track will use this Instrument. There are currently ten instruments pre-defined.

{I}{1-a *(Instrument code)*}

Instrument code	Description
1	Piano
2	Accordion
3	Distorted Synth
4	Gunshot / Percussion
5	Flute
6	Guitar
7	Harpsichord
8	Organ
9	Oboe
a	Xylophone

Instrument codes are used to allow selection of instruments with only one character

PLAYSONG LIBRARY FUNCTIONS

Action	Function
Clean Tracks	cleanTracks (rows, cols, Tracker[cols])
Add Track	addTrack (music_String , trackNumber, rows, cols, Tracker[cols])
Show Tracks	showTracks (rows, cols, Tracker[cols])
Play SID song	playSongSID (rows,cols,Tracker[cols], tempo)
select SID instrument	setInstrumentSID (voice,instrument)
delay	delay (value)
Reset SID chip	resetSID ()

Description of functions

void cleanTracks(int rows,int cols,unsigned char (*Tracker)[cols])

Objective: clean song tracks stored in an array

Parameter	Type	Description
rows	integer	Number of rows in array
cols	integer	Number of columns (each track uses 3 columns, SID uses one track for each of the 3 voices, so it could have a maximum of 9 columns)
Tracker	unsigned char	A bidimensional array of bytes with size [rows][cols]

int addTrack(char tune[], int trackNumber, int rows, int cols, unsigned char (*Tracker)[cols])

Objective: add one track of music encoded string to a Track number

Parameter	Type	Description
tune	char[]	String containing a tune in music format
trackNumber	integer	Number of tracks to add, for SID music trackNumber value must be between 1-3 to match the three available voices.
rows	integer	Number of rows in array. To store music, consider 16 rows for each 4/4 music bar.
cols	integer	Number of columns (each track uses 3 columns, SID uses one track for each of the 3 voices, so it could have a maximum of 9 columns)
Tracker	unsigned char	A bidimensional array of bytes with size [rows][cols]

void showTracks(int rows,int cols,unsigned char (*Tracker)[cols]);

Objective: Print onscreen the contents of all tracks contained in the array

Parameter	Type	Description
rows	integer	Number of rows in array
cols	integer	Number of columns (each track uses 3 columns, SID uses one track for each of the 3 voices, so it could have a maximum of 9 columns)
Tracker	unsigned char	A bidimensional array of bytes with size [rows][cols]

void playSongSID(int rows,int cols, unsigned char (*Tracker)[cols], int tempo);

Objective: Play music contained on the array, SID chip will only play the first three tracks (one for each SID voice)

Parameter	Type	Description
rows	integer	Number of rows in array
cols	integer	Number of columns (each track uses 3 columns, SID uses one track for each of the 3 voices, so it could have a maximum of 9 columns)
Tracker	unsigned char	A bidimensional array of bytes with size [rows][cols]
tempo	integer	The speed at which music is played, bigger values mean a faster melody. Default tempo = 100

int setInstrumentSID(int voice, int instrument);

Objective: Draw a filled rectangle from (x0,y0) to (x1,y1) using color (col)

Note: This is an internal function to configure the SID voice, it is called by the *playSongSID* function

Parameter	Type	Description
Voice	integer	SID voice number to configure as specified instrument
instrument	integer	number to specify which of the pre-defined instruments to configure

void resetSID();

Objective: clears all SID registers

Note: Call this function before playing any sound or after playing a sound to reset the SID chip

void delay (float value);

Objective: Used to spend cycles from the CPU to produce a delay

Note: Internally called by the *playSongSID* function

Example usage:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "playsong.h"

int main ()
{
// variable to track usedRows for the song
int usedRows;
// strings variables for storing song in Music string format
char trk1[256], trk2[256], trk3[256];

// Define Tracker Memory
int tracks=3;              // # of tracks in music score (3 for SID)
int rows=300;              // rows = # of bars * 16
int cols=tracks*3;         // calculate required columns

// Malloc seems to have a limit of assigning 4K blocks (in calypso compiler)
// this limits the max # of cols for a 3 track score to 448 rows
// 9 bytes * 448 rows = 4032 bytes
// If you have a longer song you can divide your score
//    into 4K chunks (28 music bar chunks)

// Array that will hold all encoded tracks in memory
unsigned char (*Tracker)[cols];

// Get enough memory to save all song data
   Tracker = malloc (sizeof (unsigned char[rows][cols]));

// clean memory in array before adding the music data
   cleanTracks(rows,cols,Tracker);

// Add music data
   strcpy (trk1,"I1VfRwRhRB5qAtGtFtEtRhRwRwRhRqB5AtGtFtEtRhEwG5EEEAFFFAFAFBGBGRwRhRB5AtGtFtEtRFGhG6");
   strcpy (trk2,"I1D4B3BBD4B3BRRhD4CB3C4D4hD4B3BBD4B3BRRhD4qD#4RwRB3BBRD4D4D4F#3D4F#3D4G3BGBD4B3BBD4B3BRRhD4RRhG3");
   strcpy (trk3,"I1RG3GGRGGRRwRwRGGGRGGRRwRwRGGGRF#F#F#RwRwRGGGRGGRRwRh.");

// Add Tracks to Tracker
   usedRows = addTrack (trk1,1,rows,cols,Tracker);   // Add tune in Track 1
   usedRows = addTrack (trk2,2,rows,cols,Tracker);   // Add tune in Track 2
```

```
   usedRows = addTrack (trk3,3,rows,cols,Tracker);   // Add tune in Track 3
// Show Song Title
   printf ("Demo: Turkish Dance\r");

// Play the song stored in array @ Tempo 140
   playSongSID(usedRows,cols,Tracker,140);

// Clean up audio from the SID Chip
   resetSID();

// Return Memory requested to malloc
   free (Tracker);

// End program
   return (0);
}
```