

The Commodore 64 Orchestrion

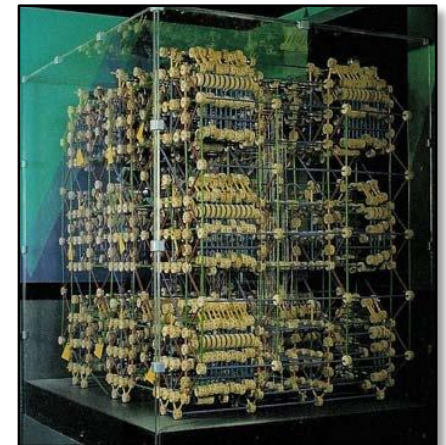
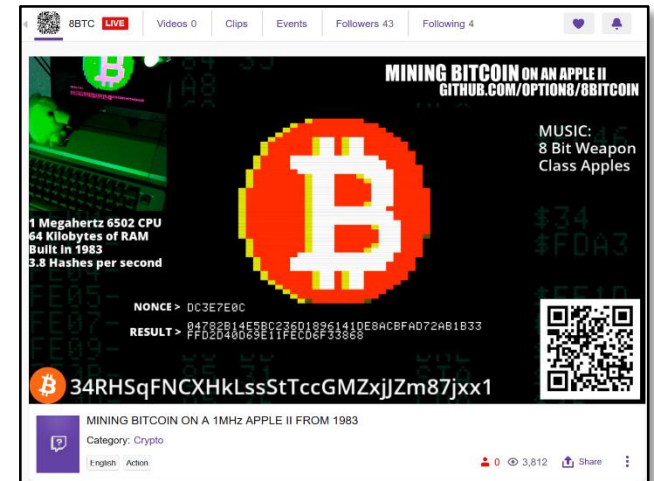
CRX 2019

David Youd
David Knapp
Joeri van Haren

Slide Deck V1.0

Retrocomputing Projects

- Ways people create in the retrocomputing space:
 - Doing 8-bit things in modern settings
 - e.g., create an 8-bit bitcoin mining rig
 - Doing modern things in 8-bit settings
 - e.g., build a working 6502 in Minecraft Redstone
 - Doing 8-bit things in analog
 - e.g., TinkerToy computer that plays tic-tac-toe
 - Doing analog things in 8-bit
 - e.g., C64 KoalaPad Theremin simulator
- Same with music:
 - e.g., new performances of old chiptunes, new chiptunes of old music, etc.



8-Bit Symphony

- I've had the privilege of correspondence with Chris Abbott (UK) during the years in which he produced a full Commodore 64 game music Symphony*



Abbott

Youd



- This got me wondering if I could reverse the idea: take a short symphonic piece and adapt it to a large number of C64s...

* Hull Philharmonic (UK) June 15th 2019, <https://www.8-bit-symphony.com/>

Named the Effort “The Commodore 64 Orchestrion”

- Orchestrion [awr-**kes**-tree-uhn]: *a machine that plays music and is designed to sound like an orchestra or band*
- Goal: Use a larger number of Commodore 64s to play music

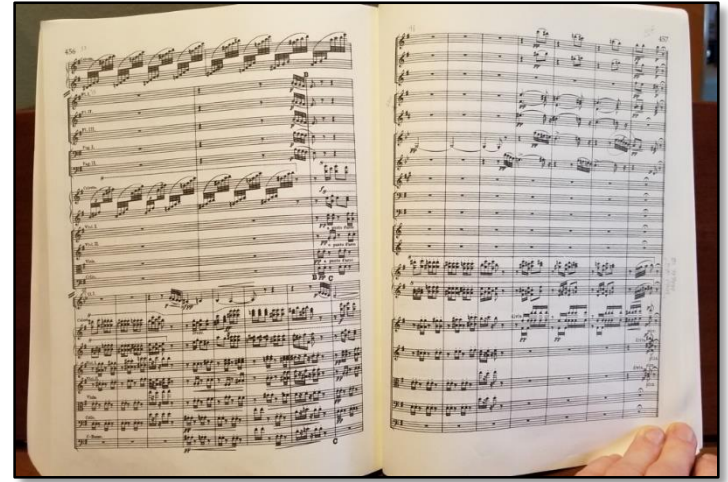


The Englehardt Orchestrion (1915)
Contains a piano, two ranks of flute pipes,
tambourine, wood block, triangle, snare
drum, bass drum, and cymbal.

On display at the Musée Mécanique
Pier 45, Fisherman's Wharf, San Francisco
Photo by David Bedel

Making The YouTube Proof Of Concept

Multiple-Commodore Adaptation of Tchaikovsky's Dance of the Sugar Plum Fairy



- Selected DotSPF because:
 1. Recognizable
 2. Short
 3. Piece characterized more by note content than by timbre / tone colors
 - (easy to adapt to simple waveforms)
 4. Many midi scores available online
 - They're all riddled with note errors, but still a leg up
 - Having a paper score reference was a *must*



Creating the Score

- Sibelius software used to create the score
 - Sibelius is the industry standard for music engraving and score production
- DotSPF Staves:
 - Flute I, II, III
 - Oboe I/II
 - English Horn
 - A Clarinet I, II
 - Bass Clarinet
 - Bassoon I, II
 - Horn I/II, III/IV
 - Celesta
 - Violin I, II
 - Viola
 - Cello
 - Double Bass

4 Full Score

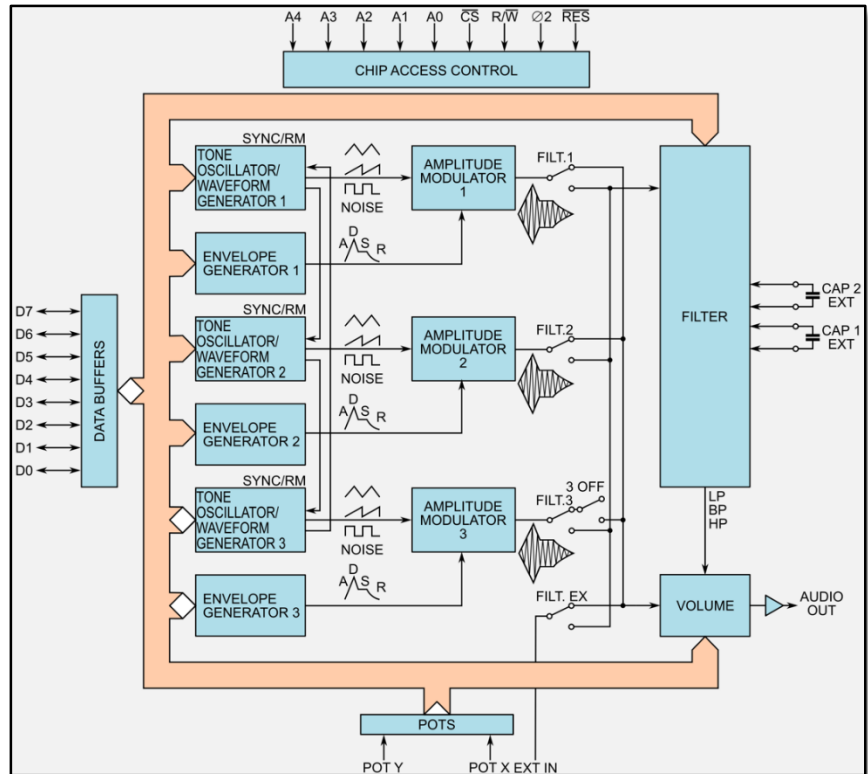
The image shows a page of a musical score, page 4, labeled 'Full Score'. It contains measures 19 through 24. The staves are arranged in two systems. The first system includes Flute I, II, III; Oboe II; English Horn; A Clarinet I, II; Bass Clarinet; Bassoon I, II; French Horn I, II, III, IV; and Celesta. The second system includes Violin I, II; Viola; Cello; and Double Bass. The music is written in 2/4 time with a key signature of one sharp (F#). Various dynamics like *mf*, *sf*, *pp*, *p*, and *f* are indicated throughout the score.

Sibelius Processing

- Wanted to get all the notes correct in 64-bit space before exporting midi data into the 8-bit space
- I'm not even an armchair conductor, so an important first step was to move away from transposed staves
 - Transposing instruments have notes written at a pitch different than what will be produced by the instrument
 - Allows performers to reuse a set of fingerings across members of their differently-pitched instrument family.
- Examples:
 - English Horn: C4->F3, A-Clarinet: C4->A3, etc.
- Sibelius made it easy to normalize the pitches

30 Voices Using 10 SID Chips

- Each voice has:
 - An ADSR envelope
 - Waveforms: sawtooth, triangle, pulse, and whitenoise
- SID voices share:
 - Filter (lo, band, hi)
 - Master volume
- Chip has many cool features
 - (not for this talk)

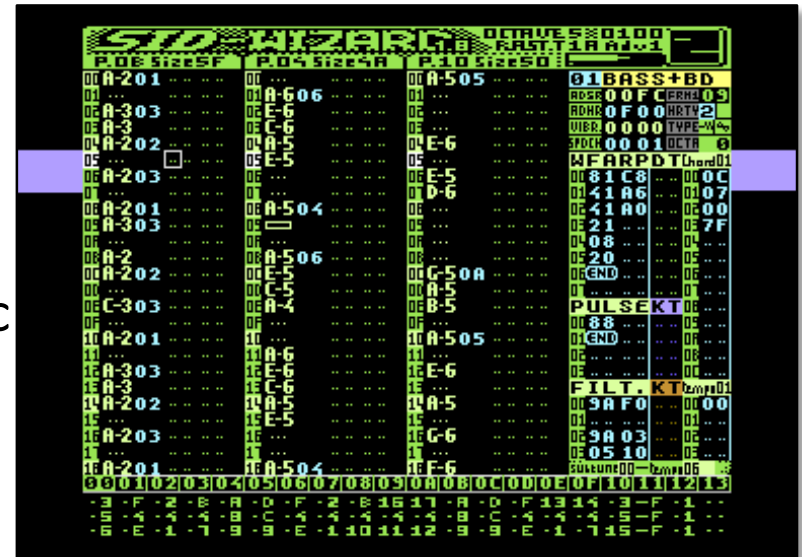


SID chip has three voices

- Choose 10 SIDs (30 voices) in order to represent each instrument type in the DotSPF score
 - The celesta takes 6 voices all by itself

SID-Wizard: A C64 Tracker

- *SID* is an overloaded term, can refer to:
 - The sound chip in a C64,
 - A piece of C64 music, or
 - A file containing C64 music
- SID files are generally created in music tools called *trackers*
 - Very compressed music representations
 - Voice organized into columns, with notes spanning rows
 - Loops can be created, with modified playback (pitch, tempo, etc.), in arbitrary order
- Selected **SID-Wizard** for creating the 10 SID files
 - Many excellent cross-platform SID-creation tools out there
 - e.g., GoatTracker
 - But SID-Wizard is a native C64 application
 - Let's you spend more time on a Commodore doing Commodore stuff ☺



Created by Mihály Horváth (aka Hermit)
v1.8 2018, <https://csdb.dk/release/?id=165302>

The YouTube Proof of Concept

The Nutcracker, Op. 71, Act II: No. 14. Pas de deux - Dance of the Sugar Plum Fairy

Pyotr Ilyich Tchaikovsky (1840-1893)

Ten Commodore 64 "Orchestrion" adaptation by David Youd (2019)



$\frac{4}{4}$: Flute I, II, III

$\frac{2}{4}$: Oboe I, II, English Horn

$\frac{4}{4}$: Clarinet I, II, Bass Clarinet

$\frac{4}{4}$: Bassoon I, II, Horn I

$\frac{1}{4}$: Horn II, III, IV



$\frac{24}{4}$: Celesta

$\frac{24}{4}$: Celesta






$\frac{2}{4}$: Violin I, Contrabass

$\frac{2}{4}$: Violin II

$\frac{6}{4}$: Viola, Violoncello

- VICE-Emulated version uploaded April 2019: https://youtu.be/zsd_L8eN18c
- Got lots of positive reception. However, received a few dissenting PMs, no doubt due to C64 regional/cultural differences. Let me explain...

*Sure, Quality Takes A Hit, But Do You See How **BIG** It Is?!?*

	UK/Europe	'Merica
Food		
Vehicles		
Multi-SID music		

- The UK/European scene is uncontested when it comes to creating high-quality SID music that pushes a C64's capabilities to the limits
 - I didn't do *any of that*.
 - Unsurprisingly, I've had a few SID cultural gatekeepers tell me that I'm having fun the wrong way. ☺

This Just In: A New 8-SID Demo (Last Weekend):



- “The Tuneful Eight”
 - <https://www.youtube.com/watch?v=KnFin5dxWLg>
 - Awarded 2nd place in "Alternative Platform Compo" at Evoke 2019 in Cologne (August 16th-18th, 2019)
 - Coding: Steffen Goerzig, Music: Lman - Markus Klein, GFX: Joe - James Svärd
- Specs:
 - Performed on a single Ultimate64 Elite
 - Runs special firmware to support 2x2 UltiSIDs and 2x2 FPGASIDs for a total of 8 replacement SIDs (24 voices)

Getting MIDI into SID-Wizard

SID-Wizard Can Import MIDI From Any DAW (i.e. Sibelius)



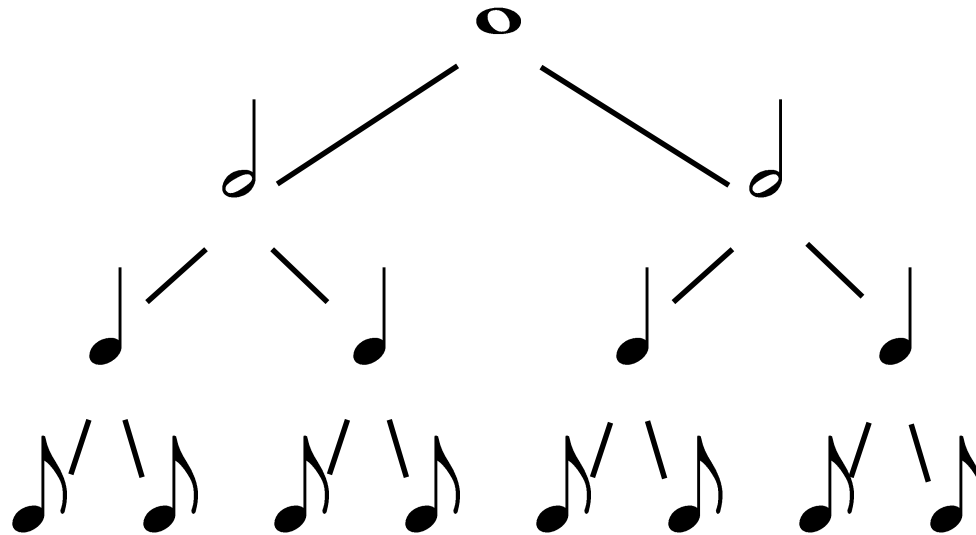
- NecroPolo (et al.) created an overview* (above) of the SID-Wizard Midi import process -- in summary:
 1. Separate polyphony into (up to three) separate single-voice tracks, then export as a midi file
 2. On a modern OS, process with the SWMconvert utility (a trivial-to-port C program) to create a native C64 SID-Wizard SWM file
 3. Run SID-WIZARD-1.8.PRG (C64) and design the instrument waveforms

Simple, But There's A Few Gotchas

- Two problems with the midi import process:
 1. Timing of triplets get mangled
 2. Even after separating staves, Sibelius sometimes exports them together in a common track (if they share an instrument type)
 - SID-Wizard needs these separated
- A little background will give context to each problem (and its solution)...

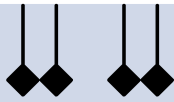

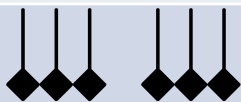

Binary Division of Note Durations

- A note can be divided into two notes of half the duration



- Sometimes you need to divide durations into three equal parts...

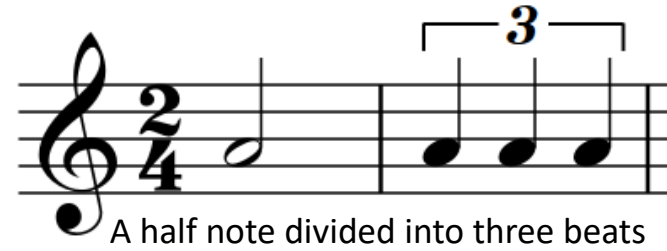
Long Ago: Notes Could Evenly Divide by Three Into Other Note Types

Mensuration sign		Mensural note subdivisions: breve -> semibreve -> minim	Measure sign	Example usage
Ⓒ	Imperfectum minor	■ = ◆◆ = 	$\frac{2}{4}$ or Ⓒ	Twinkle, Twinkle, Little Star
⓪	Perfectum minor	■ = ◆◆◆ = 	$\frac{3}{4}$ or $\frac{3}{2}$	Chim Chim Cher-ee (Mary Poppins)
Ⓒ̣	Imperfectum maior	■ = ◆◆ = 	$\frac{6}{8}$ or $\frac{6}{4}$	Pop Goes the Weasel
⓪̣	Perfectum maior	■ = ◆◆◆ = 	$\frac{9}{8}$ or $\frac{9}{4}$	Clair de Lune (Debussy)

- Ternary divisions were “perfections” (influence: the Holy Trinity)
- Modern notation only kept the *Imperfectum minor* binary divisions (influence: four elements/humors/seasons, etc.)

Today: Triplet Notation Often Used to Indicate Division by Three

- Dividing a duration into three equal parts usually requires grouping notes into a triplet
- **Problem:** Many notation-aware Commodore music programs lack support for triplets
 - e.g., Electronic Art's *Music Construction Set*, Commodore 128 BASIC 7.0's music commands, Commodore's *Music Composer*, etc.*
- SID Trackers (i.e. SID-Wizard) are row-based and can create arbitrary note durations
 - However, those offering midi importers tend to make binary-division assumptions, failing to import triplets correctly
- **Solution:** What if we could get our 14th century on, and create 3-part divisions without triplets?
 - Turns out we can scale music durations to do exactly that



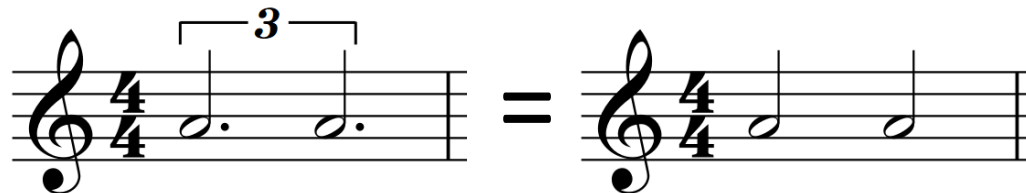
* Of course, some do support triplets -- e.g., Brøderbund's *The Music Shop*, Activision's *Music Studio*, Compute's *Enhanced Sidplayer* format, etc.

Scaling Durations

- Modern music notation allows note durations to be scaled:

Triplet notes: $\overset{3}{\bullet} = \frac{2}{3} \times \bullet$ Dotted notes: $\bullet. = \frac{3}{2} \times \bullet$

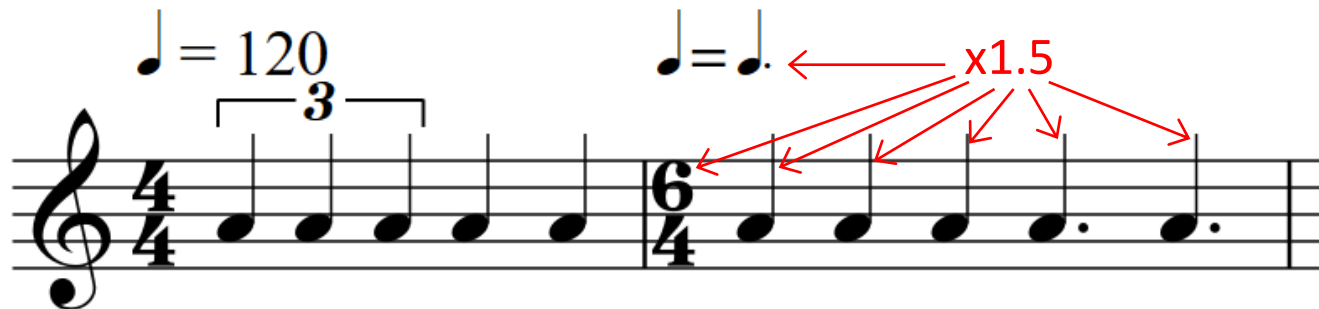
The two (above) scalings are reciprocal, so:



(Yes, you can put just two notes in a triplet. You can also mix note types and rests in a triplet as well)

Metric Modulation To Remove Triplets

- Nearly all C64 music software supports dots and ties, therefore *metric modulation* can be used to remove triplets
 - An example: Multiply the time signature, the BPM, and all note durations by $3/2$



















- The 2nd measure has the **same rhythm and the same perceived tempo** as the 1st measure
 - Notation indicates 2nd measure performed at $120 * 1.5 = 180$ BPM
- This solution generalizes (eliminate 5-tuplet groupings, etc.)

SID-Wizard Tempos and Rows-Per-Beat

- The metric modulation fix to Problem #1 creates an additional opportunity
 - Can use metric modulation functionality to select the meter for each SID that minimizes the tracker row counts
 - This requires different SID-Wizard tempos
- SID-Wizard offers 64 play speeds: tempo 0 (fastest) to \$3F (slowest)
 - Perceived music tempo is unaffected by changes in SID-Wizard tempo when the (rows-per-beat * SID-Wizard tempo) remains a constant
- The 10 SIDs in the YouTube Orchestrion demo use different SID-Wizard tempos
 - However, the rows-per-beat * SID-Wizard tempo is always 24
 - Beats Per Minute (BPM) for NTSC $\approx 3007.4744 / 24 \approx 150$ BPM
 - This makes the YouTube video more visually appealing
 - Each SID plays back at a different speed, but the music still lines up in time

Mapping Note Durations To Non-fractional SID-Wizard Row Counts

Metric mod factor	Rows per 	SID-Wiz tempo															
0.25	1	24 (\$18)	3	2	1.33	1.5	1	0.67	0.75	0.5	0.33	0.38	0.25	0.17	0.19	0.13	0.08
0.5	2	12 (\$0C)	6	4	2.67	3	2	1.33	1.5	1	0.67	0.75	0.5	0.33	0.38	0.25	0.17
1	4	6	12	8	5.33	6	4	2.67	3	2	1.33	1.5	1	0.67	0.75	0.5	0.33
1.5	6	4	18	12	8	9	6	4	4.5	3	2	2.25	1.5	1	1.13	0.75	0.5
3	12	2	36	24	16	18	12	8	9	6	4	4.5	3	2	2.25	1.5	1
6	24	1	72	48	32	36	24	16	18	12	8	9	6	4	4.5	3	2

- Green (above) shows available integer solutions to duration divisions for a varying number of rows per quarter note
- Each SID uses the minimum rows-per-beat count based on the granularity of its note durations, e.g.:
 - Horns only use multiples of quarter notes, so 1 row-per-beat is sufficient
 - However, the celesta requires 24 rows-per-beat
 - 3 rows for 32nd notes, and 4 rows for 16th triplet notes (circled above)*

* Assuming a 4/4 version of the score (originally 2/4 with finest duration granularity of 64th notes)

Problem #2:

Separating Polyphony In a Track

- A Midi file is basically a stream of on/off note events and the durations (delta time) between these events
- In the special case where one voice (without polyphony) belongs to one track/channel, the delta time explicitly indicates note durations
 - This is what the SID-Wizard midi import quantizer wants
 - Can't be guaranteed for midi exports from Sibelius...

Midi Delta-Time Rewriting

- Sibelius will sometimes group different staves from the same instrument type into a single track in its midi exports
- Fix: Wrote Python to perform delta-time rewriting needed to pull out each channel into its own track
 - This way, note on/off delta-time events represent individual note durations (required for SID-Wizard import)
 - Problem solved

2-note polyphony
example:



Mixed delta times:

Track	Channel	Notes	Delta time
1	0	E4 on	0
1	1	C3 on	0
1	0	E4 off, F4 on	1280
1	1	C3 off, G3 on	640
1	0	F4 off, G4 on	640
1	0	G4 off	1280
1	1	G3 off	0

Python →

Separate delta times:

Track	Channel	Notes	Delta time
1	0	E4 on	0
1	0	E4 off, F4 on	1280
1	0	F4 off, G4 on	1280
1	0	G4 off	1280
2	1	C3 on	0
2	1	C3 off, G3 on	1920
2	1	G3 off	1920



Python Processing Pipeline #1: Sibelius Output to SID-Wizard Format

- Python Input:

- MIDI score from Sibelius

- Rewrite delta time for channel separation
 - CSV mapping file to direct the 30 staves' content into 10 3-track SID groupings
 - Groupings based on SID filter sharing constraints
 - Perform specific metric modulation for each SID grouping
 - Make all tracks equal length in time

- Python Outputs:

- 10 midi files (for testing)
 - A D64 (C64 floppy image) containing 10 SWM files for manual processing in SID-Wizard
 - Calls out to SWMConvert and VICE's C1541



SID-Wizard Processing

SID-Wizard: Instruments

- SID-Wizard sound design is a manual step in the pipeline
- Many instrument waveforms needed:
 - Flute, Oboe, English Horn, Clarinet in A, Bass Clarinet in B, Bassoon, Horn in F, Celesta, Violin, Viola, Cello, and Contrabass
- Fortunately, SIDWizard is distributed with reasonable (albeit 8580-specific) implementations of most of these. They're found in:
 - `examples\instruments`, and
 - `examples\instruments\didnt-fit-on-disk`
- Instrument assignments were manual (not automated in the python)

SID-Wizard: Exporting 10 SIDs

- (Manually) used the C64 utility SID-MAKER-1.8.PRG to export each of the 10 SWM files as standard SID files
- SID file format:
 - 124 bytes of header info
 - e.g., author name, NTSC or PAL, 6581 or 8580, etc.
 - native C64 code that plays the music
 - Contains a music engine and the note data
 - (Does not contain the C64 code to drive the music engine)

The YouTube
VICE Demo Was
Easy



The Hardware
Demo Was...

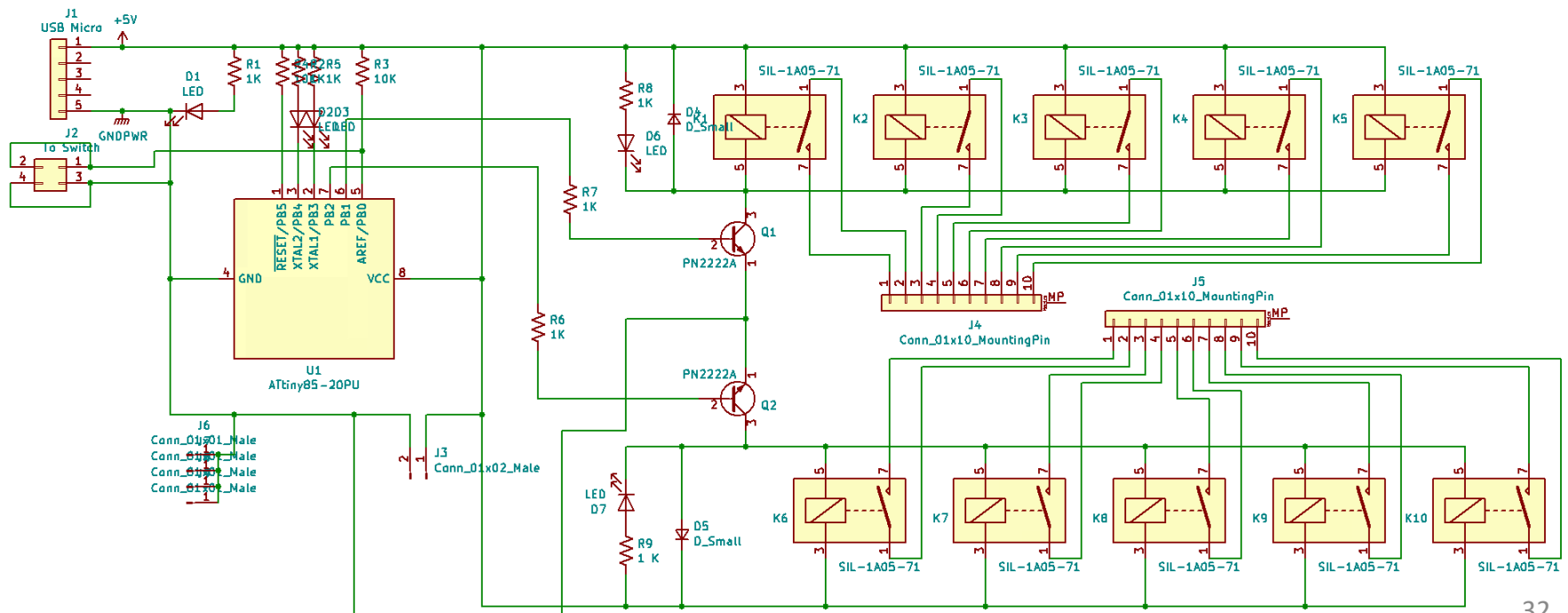


Hardware Goals

- Play music on 10 C64s simultaneously
 - Make the sound decent quality
- Eliminate any potential electrical damage to C64s
- Make it cheap

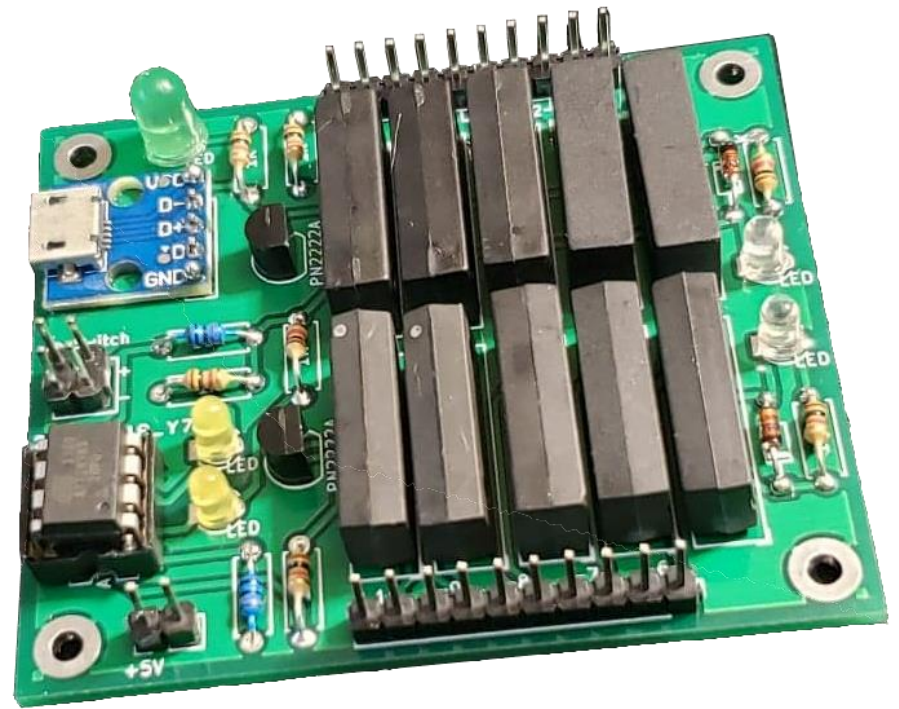
Start 10 C64s Simultaneously

- “Trigger Box” sends the start signal to all the Commodore’s joystick ports
 - 1/8” stereo cables to DE-9
 - ATtiny85 used for trigger debouncing
 - 2 sets of 5 relays powered by 2N2222 transistors
 - Diode protection on relays
 - Powered via USB



Trigger Box

- If anyone is interested we have a couple of extras



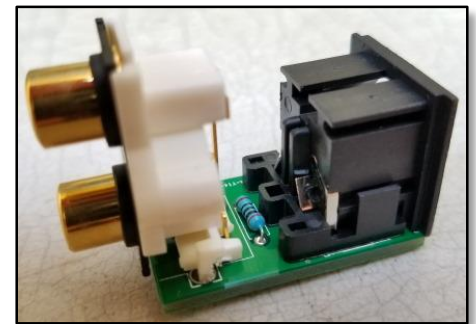
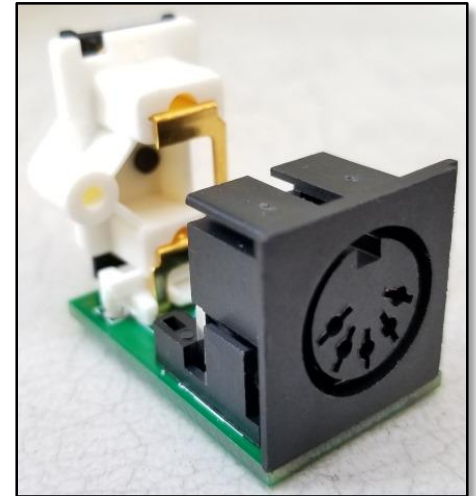
Eliminate Potential Damage to C64s

- Trigger
 - Solution: use reed relays to close switch on joystick FIRE button
 - 1/8" stereo cable channels connected with no ground, for electrical isolation
- Power supplies
 - Put C64 savers on each power supply to protect against overvoltage
 - Problem: that gets EXPENSIVE
 - \$40 per x 10 C64s = \$400 in savers alone
 - Solution: build our own savers
 - Based on Ray Carlson design with Zeners
 - See Knapp's talk for design
 - Bonus: fun project for people to build at CRX

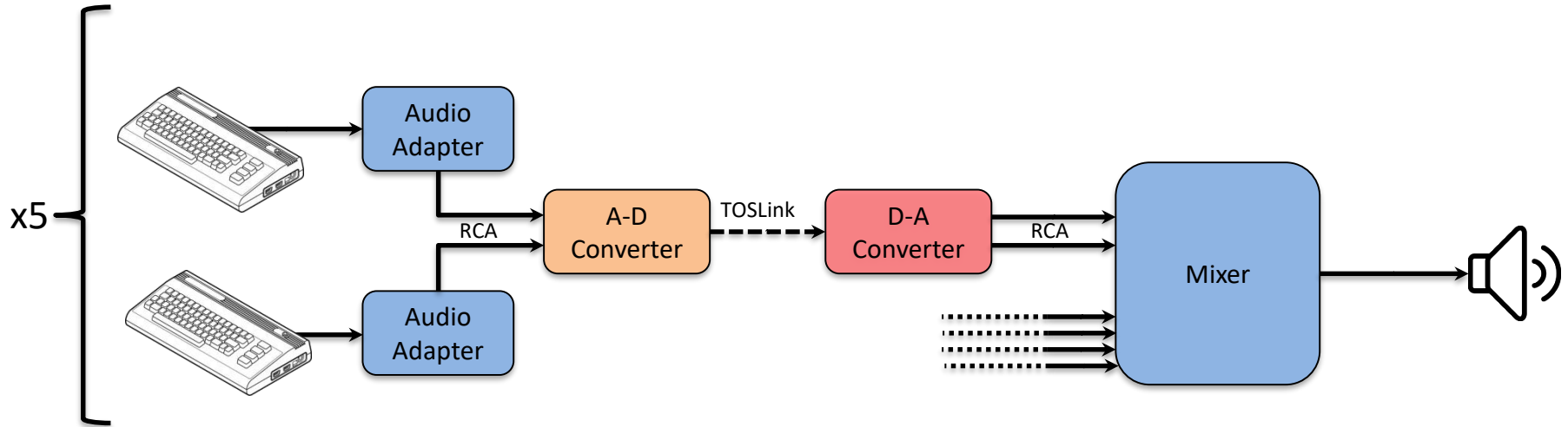


C64 Sound Output

- Made little adapters for 5-pin DIN cable to RCA plugs for audio/composite video
 - Cost (including parts) \$2.50 each
 - 100-ohm impedance to ground on SID chip sound input for noise reduction
- Sound from each pair of C64s run through stereo optical converter / decoder
 - Eliminates ground loops
 - Grounds only shared between pairs of C64s



Sound Path



- Output from each pair of C64s
 - Fed into stereo A-D converter (\$10)
 - TOSLink optical cable (\$2)
 - Converted back to stereo by D-A converter (\$8)
- 5 pairs sent to mixer

Creating The 10 Cartridges

Playing The SIDs

- SID Players are available for most modern platforms
 - Player must emulate enough of a C64 and the SID chips to execute the native C64 code in the SID
 - For online play of over 50 thousand SIDs (the HVSC collection), I highly recommend Jens-Christian Huus's <https://deepsid.chordian.net/>
- SID playback on a Commodore 64:
 - Requires code (not included in the SID file) to repeatedly call the song update routine
 - This usually occurs either 50 or 60 times a second via
 - a raster interrupt (IRQ) @ 50Hz PAL or 60Hz NTSC
 - a CIA (Complex Interface Adaptor) timer (IRQ if CIA#1, NMI if CIA#2),
 - or some delay loop (time delay, non-interrupt raster position check, etc.)
- We burned 10 cartridges which contain our music and driver code
 - Helps with running the majority of the C64s “headless”
 - And hot swapping drive connections was not an option

Cartridge Boilerplate

*=\$8000

```
; CART ROM HEADERS
WORD COLDSTART          ; CARTRIDGE COLD START VECTOR
WORD WARMSTART          ; CARTRIDGE WARM START VECTOR
BYTE $C3,$C2,$CD,$38,$30 ; CARTRIDGE AUTOSTART STRING ("CBM80")

; 16K CART ($8000-$9FFFF ROML AND $A000 TO $BFFF ROMH) THAT KEEPS THE
; 8K KERNAL ROM ($E000-$FFFF)
; DO THE KERNAL SETUP STUFF THAT WOULD HAVE HAPPENED HAD THE CART NOT BEEN
; DETECTED IN THE $FCE2 KERNAL SETUP ROUTINE:
COLDSTART
SEI
STX $D016              ; VIC-II INIT (FOR PAL/NTSC CHECK, ETC.)
                        ; X=0 AFTER MATCHING 'CBM80' IN KERNAL'S
                        ; $FD02 CART CHECK CODE
JSR $FDA3              ; INITIALIZE CIA CHIPS (INCLUDING CIA1 TIMER A @ ~60HZ)
JSR $FD50              ; CLEAR AND TEST RAM
JSR $FD15              ; SET I/O VECTORS ($0314-$0333) TO KERNAL DEFAULTS
JSR $FF5B              ; INIT VIDEO
CLI

WARMSTART
JMP MAININIT           ; NON-CART TESTING: SYS32794
```

- Note: The KERNAL is kept by the cartridges. (Without the KERNAL, the code samples in this presentation would need to be modified)

Setup Raster IRQ To Drive Music Update

```
SEI                ; DISABLE MASKABLE INTERRUPTS, AND THEN TURN THEM OFF (BELOW)
LDA #%01111111    ; BIT 7 (OFF) MEANS THAT ANY 1S WRITTEN TO CIA ICRS TURN THOSE BITS OFF
STA $DC0D         ; CIA#1 INTERRUPT CONTROL REGISTER (IRC): DISABLE ALL INTERRUPTS
STA $DD0D         ; CIA#2 ICR: DISABLE ALL INTERRUPTS
LDA $DC0D         ; ACK (CLEAR) ANY PENDING CIA1 INTERRUPTS (READING CLEARS 'EM)
LDA $DD0D         ; SAME FOR CIA2
ASL $D019         ; TOSS ANY PENDING VIC INTERRUPTS (WRITING CLEARS 'EM, VIA RMW MAGIC)

LDA #$01
STA $D01A         ; ENABLE RASTER-COMPARE INTERRUPT
LDA #$80
STA $D012         ; SET SCAN LINE THAT WILL TRIGGER THE RASTER-COMPARE INTERRUPT
LDA #%00011011    ;
STA $D011         ; BIT 7 IS THE 9TH BIT OF THE SCAN LINE VALUE

LDA #<IRQHANDLER ; HOOK INTERRUPT ROUTINE
STA $0314
LDA #>IRQHANDLER
STA $0315

CLI                ; RESTORE INTERRUPTS, HOOKING COMPLETE
```

- Music plays at the rate that it's updated (in this case, 60 times per second)
 - 60 Hz update frequency is the default SID-Wizard NTSC assumption
- This is how nearly everyone drives their SID music play routines

Raster IRQ Handler (simplified)

```
IRQHANDLER
INC $D019          ; CLEAR (ACK) OUR RASTER-COMPARE INTERRUPT
                   ;   INC (OR ASL) USES RMW (READ-MODIFY-WRITE) TO READ VALUE, WRITE THAT
                   ;   VALUE BACK ON IT (THIS DOES THE INTERRUPT ACK), THEN WRITE RESULT
                   ;   OF THE INC OPERATION (WHICH DOES NOTHING).  AVOIDS AN LDA FIRST.

LDA #$0E
STA $D020          ; BORDER COLOR CHANGE TO INDICATE RASTER TIME CONSUMED
JSR UPDATERMUSIC
LDA #$06
STA $D020          ; RESTORE BORDER
JMP $EA31          ; EXIT THROUGH THE KERNAL'S 60HZ IRQ HANDLER ROUTINE
```

- Code (not shown) includes logic to stop the music if the trigger button is held for 3 seconds.

C64 Synchronization Concerns

- Joeri showed me a Commodore he was repairing that had the older VIC-II chip in it
 - It later occurred to me that an old VIC-II chip might have a different screen refresh rate. Sure enough, it does!
- Given 1,022,727 CPU cycles per second (NTSC), and 90 seconds of music:

NTSC VIC-II Variant	Lines per frame	Cycles per line	Cycles per frame	Refresh rate	Music updates	Total cycles
New: 6567R8	263	65	17095	59.83Hz	5385	1538550
Old: 6567R56A	262	64	16768	60.99Hz	5490	1509120

- This predicts that the music update would be called 2% more frequently with the older 6567R56A chip! ☹
- Therefore, we changed over to CIA timer-based interrupts, so that we aren't tied to inconsistent screen refresh rates
- Note: Timers from CIA 6526/6526A chips manufactured ≥ 1987 throw an interrupt one cycle earlier than they should
 - But we can live with a one-cycle difference every 60hz
 - (CIA 6526/6526A manufactured < 1987 and the 8521 are unaffected)

Setup CIA#1 Timer IRQ

```
SEI                ; DISABLE MASKABLE INTERRUPTS, AND THEN TURN THEM OFF (BELOW)
LDA #%01111111    ; BIT 7 (OFF) MEANS THAT ANY 1S WRITTEN TO CIA ICRS TURN THOSE BITS OFF
STA $DC0D          ; CIA#1 INTERRUPT CONTROL REGISTER (IRC): DISABLE ALL INTERRUPTS
STA $DD0D          ; CIA#2 ICR: DISABLE ALL INTERRUPTS
LDA $DC0D          ; ACK (CLEAR) ANY PENDING CIA1 INTERRUPTS (READING CLEARS 'EM)
LDA $DD0D          ; SAME FOR CIA2
ASL $D019          ; TOSS ANY PENDING VIC INTERRUPTS (WRITING CLEARS 'EM, VIA RMW MAGIC)

LDA #<SONGSPEED    ; SET UP CIA#1 TIMER A DURATION (SECONDS = SONGSPEED/1022730 for NTSC)
STA $DC04          ; KERNAL CIA#1 TIMER USES 17045 CYCLE COUNT FOR 60HZ
LDA #>SONGSPEED    ; HOWEVER, 17094 GIVES STABLE SYNC WITH NTSC ON NEW VIC-II
STA $DC05          ; RASTER BAR WILL SLOWLY MOVE ON OLD VIC-II, INDICATING OLD CHIP
LDA #<IRQHANDLER   ; HOOK INTERRUPT ROUTINE (NORMALLY POINTS TO $EA31)
STA $0314
LDA #>IRQHANDLER
STA $0315
LDA #%10000001     ; CIA#1 ICR: B0->1 = ENABLE TIMER A INTERRUPT,
STA $DC0D          ; B7->1 = FOR B0-B6, 1 BITS GET SET, AND 0 BITS IGNORED
LDA $DC0E          ; CIA#1 TIMER A CONTROL REGISTER
AND #%10000000     ; PRESERVE KERNAL-SET TOD CLOCK NTSC OR PAL SELECTION
ORA #%00010001     ; B0->1=START TIMER A,
                   ; B3->0=TIMER CONTINUOUS RUN MODE,
                   ; B4->1=FORCE LATCHED VALUE INTO TIMER A COUNTER
                   ; LATCHED VALUE (WHICH CAN BE CHANGED AT ANY TIME) WILL BE RELOADED
                   ; WHEN TIMER REACHES 0 (IN EITHER ONE-SHOT OR CONTINUOUS RUN MODE).
                   ; FORCING LOAD JUST PUTS IT IN THE TIMER IMMEDIATELY.
                   ; B5->0=TIMER A DECREMENTS EACH CPU CYCLE (MAPPING THE C64 HAS THIS WRONG)

CLI                ; RESTORE INTERRUPTS, HOOKING COMPLETE
```

CIA Timer IRQ Handler (simplified)

```
IRQHANDLER
LDA $DC0D          ; ACK (CLEAR) CIA#1 INTERRUPT
LDA #$0E
STA $D020          ; BORDER COLOR CHANGE TO INDICATE RASTER TIME CONSUMED
JSR UPDITEMUSIC
LDA #$06
STA $D020          ; RESTORE BORDER
JMP $EA31          ; EXIT THROUGH THE KERNAL'S 60HZ IRQ HANDLER ROUTINE
```

- Our timer interrupt handler still modifies the border to show SW play routine cycle consumption
- Bar position is stationary on new VIC-II chips, but slowly sweeps the screen on old VIC-II chips
 - Turns out, two of the orchestrion C64s have the older chip



Versa64Cart Makes Assigning SIDs to C64s Easy!

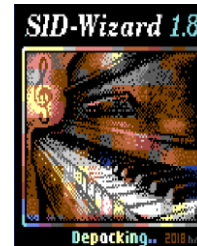
- Used 64K EEPROM chips to hold 4 16K cartridge images
 - Could easily have gotten away with 8K cartridges
- 16K cartridge uses:
 - 8K at \$8000 (pin 8 GAME low)
 - plus 8K at \$A000 (pin 9 EXROM low)
 - Requires SID to be relocated in memory at runtime





Python Processing Pipeline #2: SID-Wizard Output to Cartridge Code

- Python Input:
 - 10 SID files
- Python Outputs:
 - 3 64K BIN files (carts 1-4, 5-8, and 9-10) for cartridge EEPROM burning
 - BIN = a PRG with the first two (load address) bytes removed
 - 10 CRT files, for cartridge testing in the VICE emulator
 - CRT is a BIN file with a bunch of headers describing the cartridge
 - e.g., expansion port pins 8 (GAME) and 9 (EXROM), chip type (e.g., ROM, RAM), image size, etc.
 - Coded python to the specification for 16K cartridge CRT headers:
http://vice-emu.sourceforge.net/vice_16.html#SEC330



First Complete Test:



- Full setup requires 13 110v and 11 USB power outlets

Future Explorations?

Different Music?

- This was a lot of effort for 90 seconds of music
- Joeri thinks that The Nutcracker is boring, and that I should adapt Daglish's *The Last Ninja* to 10 C64s
 - We're still negotiating... 😊

Resynchronization Needed for Longer Music

- Our trigger box approach resyncs the C64s' CIA timers (crystal-based) whenever the song is (re)started
 - Our music is only 90 seconds long, so no resync needed
- However, much longer music will require some kind of synchronization broadcast signal
- How often to resync?
 - Resyncing at the update rate is overkill, as the update rate is comfortably beyond human temporal perceptual limits
 - For NTSC 59.95 Hz, jiffy music update resolution is 16.8 ms
 - Sync only required every minute or so
 - Simple (naive) approach: Assume that for every sync, there are $n-1$ music update + delay pairs, followed by one music update. Immediately after the last update, block until next sync
 - If syncing once a minute, the above approach requires cycle count to be accurate to within 1 part in 3600 (trivial)

Synchronization Approaches

- Got some great sync suggestions from Mario Schallner's / Marilli Man's SID Trackers Facebook group
- Internal sync approach:
 - Krzysztof Kluczek suggests that the songs could sync from CIA Time of Day (TOD) clock interrupts
 - Too slow to drive the music (10ths of seconds granularity), but sufficient for resyncing every few minutes
 - TOD clock not derived not from the crystal, but the AC input (so make sure all boxes are on the same outlet)
- External sync approaches:
 - Max Hall says that Ben Daglish did a two C64 version of his Trap music using joystick ports to periodically maintain sync
 - Marcin Skoczylas has a current FPGA project where he uses a similar approach
 - Instead of active polling for joystick port signals, external signals can trigger interrupts
 - CIA#1 can throw IRQ based on the cassette read line, and CIA#2 can throw NMI based on Pin B of the User Port

If We Were *Really* Ambitious...



- This project had a lot of moving parts, but at no time did it feel intractable



- For an extra challenge, trigger box could also be made to collect periodic heartbeats sent from the headless C64s to detect if crashed or out of sync



- If really ambitious, put a GPS chip on each music cartridge for 1 nanosecond time resolution



- If really (really) ambitious, all the sync broadcasts and heartbeats could be wireless if ZigBee was added to the music cartridges 😊

Quick Plug for The Current 8-Bit Symphony Kickstarter (ends Sep 8th)

- 8-bit symphony: professional orchestrations of Commodore 64 game tunes
 - Rob Hubbard, Martin Galway, Ben Daglish, Paul Norman, etc.
- The current Kickstarter aims to have a professional recording of the music that I saw performed in Hull (it was amazing!)
 - If funded, will be recorded in Prague with a John-Williams-size studio orchestra, supervised by Rob Hubbard himself
- Links:
 - The Kickstarter: <https://www.kickstarter.com/projects/8-bit-symphony/8-bit-symphony-pro-double-orchestral-cd-of-8-bit-classics>
 - 25-minute music sampler: <https://soundcloud.com/chrisabbott/8-bit-symphony-volume-1-sample-caution-spoilers-for-8-bit-symphony-concert>
- **If you love C64 music like I do, it's a no-brainer to support this one**

Questions?

Suggestions?

Thanks!