Mini Colecovision

SPARKLETRON

November 8, 2024

Jay Convertino

Contents

1 Introduction

Mini Colecovision is a portable console version of the original Colecovision. Much of the TTL and Analog Monostable circuits are emulated by a CPLD. The full PCB and CPLD code is in this repository. It emulates the original Colecovision with the additional super game module. No 3D printed case is included at the moment, but has been designed and tested. This manual is not at step by step document of how to build the unit, more of a highlight of aspects of the project.

1.1 Specifications

- Z80 CPU
- 32 KiB of RAM
- 32 KiB of ROM
- SN76489 Sound Chip
- YMZ284 Sound Chip
- TMS9118 Video Display Processor with 16 KiB of VRAM
- MAX7000S CPLD (EPM7128SLC)
- Main PCB, four layer
- Right Angle PCB, two layer

1.2 Parts List

Item	Qty	Reference(s)	Value
1	13	C1, C7, C8, C10 to C14, C18, C19, C23, C25, C29	100nF
2	1	C2	330uF
3	4	C3, C5, C24, C30	10uF
4	1	C4	100pF
5	1	C6	270pF
6	8	C9, C17, C22, C31 to C35	100nF
7	2	C15, C16	33pF
8	2	C20, C21	10nF
9	1	C26	0.47uF
10	1	C27	0.1uF
11	1	C28	2.2uF
12	1	D1	LED
14	1	J1	Conn_01x07
15	3	J2, J10, J11	Conn_01x02
16	1	J3	Conn_Coaxial
17	1	J4	Cartridge Port

18	1	J5	DB9 Male
19	1	J6	Conn_02x05_Odd_Even
20	1	J7	DB9 Male
21	1	J9	SJ1-3525NG
22	3	L1, L2, L3	4.7uH
23	1	Q1	2N3904
24	1	R1	4k7
25	1	R2	470R
26	2	R3, R28	100k
27	2	R4, R27	100K
28	2	R5, R6	2k2
29	3	R7, R8, R9	3K3
30	1	R10	75R
31	1	R11	510R
32	1	R12	100R
33	1	R13	3k3
34	4	R14, R15, R16, R17	1k
35	2	R18, R19	10k
36	1	R20	1K
37	1	R21	1k
38	1	R22	220R
39	1	R23	10K
40	2	R24, R26	1K
41	1	R25	68K
42	2	RN1, RN2	10k
43	1	RV1	10K
45	1	SW1	SW_Push
46	1	SW2	SW_SPDT
47	1	SW3	SW_SPDT
48	1	U1	TPA711D
49	1	U2	SN76489AN
50	1	U3	Z84C0010AEG
51	1	U4	CY62256-55PC
52	1	U5	27C256
53	1	U6	TMS9118NL
54	2	U7, U8	TMS4416
55	1	U9	EPM7128SLC
56	1	U10	74ABT125
57	1	U11	YMZ284
58	5	U12, U13, U14, U15, U16	74AHCT1G08
59	1	Y1	10.738635 MHz

2 Building

This document assumes some Electrical Engineering knowledge. Building circuits is not trivial due to the mix of SMD and through hole components. What follow are general steps to build the Mini Colecovision

- Create main PCB from schematic/gerber/coleco_original.zip
- Create Right Angle PCB from schematic/gerber/right_angle/right_angle.zip

- Program ROM with BIOS
- Populate main PCB
- Populate right angle PCB
- Power up and program CPLD
- Build your own case

2.1 Dependencies

The following are the dependencies needed to build the firmware and PCB for the system.

- Quartus 13.0 sp1
- python 3.X
- KiCAD v7.X

2.1.1 Protable Coleco Glue File List

- src
  - ’src/porta_glue_coleco.v’: ’file_type’: ’verilogSource’
- constr
  - ’constr/porta_glue_coleco.sdc’: ’file_type’: ’SDC’
- tb
  - ’tb/tb_porta_glue_coleco.v’: ’file_type’: ’verilogSource’

2.1.2 Fusesoc

Fusesoc is used for the simulation target only. There are no build targets due to the use of Quartus 13.0sp1. This makes the use of it a bit silly. It does make it easier to use in future projects where the RAM,ROM,CPU,VDP, and Sound chips are also IP cores.

2.1.3 Protable Coleco Glue Targets

- default

  Info: Default IP target for future tool intergration.

  - src
  - constr

- sim

  Info: Simulation target for basic test bench.

  - src
  - tb

2.1.4 Quartus

This project uses the last version of Quartus that supports the MAX7000S series. The version is 13.0sp1. The project is located at src/quartus13sp01/. Once you have the project open please follow the softwares steps for building and programming the CPLD bitfile.

2.2 PCB

The four layer PCB is fairly easy to populate. The right angle PCB is a dual layer PCB which is even easier. I recommend starting with resistors, then IC's, and then the rest. Surface mount parts should be done last. This is a fairly complex project to build, take great caution in making sure your CPU and CPLD are installed correctly. Its easy to rotate square packages these come in.

2.3 3D Printed Case

A 3D printed case model is not included. I've kept this for release in the future.

2.4 Programming

There are two devices that need to be programmed. ROM (read only memory) and the CPLD (complex programmable logic device). They use two different methods to be programmed. The ROM is done off the board and then installed. The CPLD is installed and the JTAG header is used to upload the bitfile.

2.4.1 ROM

A TL866 is an excellent device for programming the ROM with a BIOS. The open source minipro application works well with it and its clones. Below is a example command to use to program the ROM with a bios.

```
$ minipro -p ST27C256 -w coleco_bios.bin
```

2.4.2 CPLD

Quartus 13.0sp1 is the easiest way to build and program the MAX7000 CPLD. You will need an altera blaster. I recommend the chinese clone blasters, they actually worked the best. While the worst was the Terasic blaster which did not work at all. As for instructions on how to program it in Quartus, please see the software for details.

3 Usage

3.1 Directory Guide

Below highlights important folders from the root of mini_colecovision.

1. **docs** Contains all documentation related to this project.
   - **datasheets** Contains all datasheets for components.
   - **manual** Contains user manual and github page that are generated from the latex sources.
2. **img** Contains images of the project
3. **schematic** KiCAD v7.X schematic and PCB designs
   - **gerber** Contains gerber files and archives for production.
   - **pdf** PDF schematic
4. **src** CPLD firmware source
   - **protable_coleco** Contains verilog source code and constraits
   - **quartus13sp01** Quartus project to use to generate firmware file.

4 Module Documentation

What follows are PDF pages generated from natural docs HTML pages. This documents the source code used for the CPLD.

porta_glue_coleco.v

AUTHORS

JAY CONVERTINO

DATES

2024/11/06

INFORMATION

Brief

Colecovision SGM glue logic chip

License MIT

CONSTANTS

DEF_RESET_DELAY_BIT

Number of bits for reset delay register

DEF_FB_MONOSTABLE_COUNT

delay till state is at 1 instead of 0 (its stable state) for feedback stable circuit

DEF_IRQ_MONOSTABLE_COUNT

delay till state is at 1 instead of 0 (its stable state) for the controller (spinner) generated interrupt.

porta_glue_coleco

```
module porta_glue_coleco (
input
clk,
                                                                    input [
15:0]
A,
input
C1P1,
input
C1P2,
input
C1P3,
input
C1P4,
input
C1P6,
input
C1P7,
input
C1P9,
input
C2P1,
input
C2P2,
input
C2P3,
input
C2P4,
input
C2P6,
input
C2P7,
input
C2P9,
input
MREQn,
input
IORQn,
input
RFSHn,
input
M1n,
input
WRn,
input
RESETn_SW,
input
RDn,
                                                                    inout [
7:0]
D,
output
CP5_ARM,
output
CP8_FIRE,
output
CS_h8000n,
output
CS_hA000n,
output
```

```
CS_hC000n,
output
CS_hE000n,
output
SND_ENABLEn,
output
ROM_ENABLEn,
output
RAM_CSn,
output
RAM_OEn,
output
CSWn,
output
CSRn,
output
WAITn,
output
RESETn,
output
RAM_MIRRORn,
output
INTn,
output
AS,
output
AY_SND_ENABLEn
)
```

Colecovision Super Game Module Glue Logic

Ports

clk
input
Clock for all devices in the core

A
input[15: 0]
Address input bus from Z80

C1P1
input
DB9 Controller 1 Pin 1

C1P2
input
DB9 Controller 1 Pin 2

C1P3
input
DB9 Controller 1 Pin 3

C1P4
input
DB9 Controller 1 Pin 4

C1P6
input
DB9 Controller 1 Pin 6

C1P7
input
DB9 Controller 1 Pin 7

C1P9
input
DB9 Controller 1 Pin 9

C2P1
input
DB9 Controller 2 Pin 1

C2P2
input
DB9 Controller 2 Pin 2

C2P3
input
DB9 Controller 2 Pin 3

C2P4
input
DB9 Controller 2 Pin 4

C2P6
DB9 Controller 2 Pin 6

input

C2P7 input	DB9 Controller 2 Pin 7
C2P9 input	DB9 Controller 2 Pin 9
MREQn input	Z80 memory request input, active low
IORQn input	Z80 IO request input, active low
RFSHn input	Z80 Refresh input, active low
M1n input	Z80 M1 state, active low
WRn input	Z80 Write to bus, active low
RESETn_SW input	Input for reset switch
RDn input	Z80 Read from bus, active low
D inout[7: 0]	Z80 8 bit data bus, tristate IN/OUT
CP5_ARM output	DB9 Controller 1&2 ARM Select
CP8_FIRE output	DB9 Controller 1&2 FIRE Select
CS_h8000n output	Select when Z80 requests memory at h8000 (GAME CART), active low
CS_hA000n output	Select when Z80 requests memory at hA000 (GAME CART), active low
CS_hC000n output	Select when Z80 requests memory at hC000 (GAME CART), active low
CS_hE000n output	Select when Z80 requests memory at hE000 (GAME CART), active low
SND_ENABLEn output	SN76489 Sound chip enable, active low
ROM_ENABLEn output	Enable BIOS ROM, active low
RAM_CSn output	RAM chip select, active low
RAM_OEn output	RAM Ouput enable, active low
CSWn output	Chip Select Write for VDP, active low
CSRn output	Chip Select Read for VDP, active low
WAITn output	Wait state generator for Z80, active low
RESETn output	Timed reset generated by Logic, active low
RAM_MIRRORn output	Extended RAM, high is extended RAM, active low is mirrored.
INTn output	Interrupt generator for Z80, active low

AS output — AY sound chip address(0)/data(1) select

AY_SND_ENABLEn output — AY sound enable, active low

REGISTER INFORMATION

Core has 3 registers at the addresses that follow.

SOUND_CACHE	h51
RAM_24K_ENABLE	h53
SWAP_BIOS_TO_RAM	h7F

SOUND_CACHE

```
localparam SOUND_CACHE = 8'h51
```

Defines the address of r_snd_cache

SOUND CACHE REGISTER
7:0
CACHE LAST WRITE TO AY SOUND CHIP

Cache Sound Chip as the SGM games read from it (Yamaha chip does not have a read like a GI does).

RAM_24K_ENABLE

```
localparam RAM_24K_ENABLE = 8'h53
```

Defines the address of r_24k_ena

24K RAM ENABLE REGISTER	
7:1	0
ZERO	ENABLE 24K RAM, ACTIVE HIGH

Super Game Module 24K RAM enable using bit 0 (Active High)

SWAP_BIOS_TO_RAM

```
localparam SWAP_BIOS_TO_RAM = 8'h7F
```

Defines the address of r_swap_ena

SWAP BIOS TO RAM REGISTER			
7:4	3:2	1	0
ZERO	ONE	BIO TO RAM SWAP, ACTIVE LOW	ONE

Super Game Module BIOS to RAM swap on bit 1 (Active Low)

r_24k_ena

```
reg [ 7:0] r_24k_ena = 0
```

register for RAM_24K_ENABLE See Also: RAM_24K_ENABLE

r_swap_ena

```
reg [ 7:0] r_swap_ena = 8'h0F
```

register for 8K RAM/ROM swap See Also: SWAP_BIOS_TO_RAM

r_snd_cache

```
reg [ 7:0] r_snd_cache = 0
```

register for SOUND_CACHE See Also: SOUND_CACHE

r_int_p1

```
reg r_int_p1 = 1'b0
```

Interrupt from player one control

r_int_p2

```
reg r_int_p2 = 1'b0
```

Interrupt from player two control

r_wait

```
reg r_wait = 1'b0
```

Wait state generated register

r_reset_counter

```
reg [ 9:0] r_reset_counter = 0
```

Timed reset counter

r_resetn

```
reg r_resetn = 0
```

Registered reset output, active low

r_mono_count_p1

```
reg [11:0] r_mono_count_p1 = 0
```

monostable circuit counters, player 1 AND

r_mono_count_p2

```
reg [11:0] r_mono_count_p2 = 0
```

monostable circuit counters, player 2 AND

r_mono_count_int_p1

```
reg [ 5:0] r_mono_count_int_p1 = 0
```

monostable circuit counters, player 1 interrupt

r_mono_count_int_p2

```
reg [ 5:0] r_mono_count_int_p2 = 0
```

monostable circuit counters, player 2 interrupt

r_mono_p1

```
reg r_mono_p1 = 1'b0
```

Feedback from IRQ to controller 1 register

r_mono_p2

```
reg r_mono_p2 = 1'b0
```

Feedback from IRQ to controller 2 register

r_ctrl_fire

```
reg r_ctrl_fire = 1'b1
```

NAND Feedback Flip Flop FIRE select.

r_ctrl_arm

```
reg r_ctrl_arm = 1'b0
```

NAND Feedback Flip Flop ARM select.

ASSIGNMENT INFORMATION

How signals are created

s_ram_csn

```
assign s_ram_csn = (
s_y0_seln |
r_swap_ena[1]
) & (s_ram2_csn | ~r_24k_ena[0]) & (s_ram1_csn | ~r_24k_ena[0]) & s_ram0_cs
```

RAM Chip select when address is requested (active low).

(s_y0_seln | r_swap_ena[1])	address range starting at h0000, swap bios/rom bit is enabled (1 is disabled).
(s_ram1_csn | ~r_24k_ena[0])	address range starting at h4000, 24k enable bit from register.
(s_ram2_csn | ~r_24k_ena[0])	address range starting at h2000, 24k enable bit from register.
s_ram0_csn	address range starting h6000, this is always an available range.

RAM_OEn

```
assign RAM_OEn = RDn | s_ram_csn
```

RAM Output enable when read is requested (active low).

RDn	Z80 read request, active low.
s_ram_csn	See Also: s_ram_csn

RAM_CSn

```
assign RAM_CSn = s_ram_csn
```

RAM Chip Select output assignment.

s_ram_csn	See Also: s_ram_csn

RAM_MIRRORn

```
assign RAM_MIRRORn = (
r_24k_ena[0] |
                                                                          ~
r_swap_ena[1]
)
```

RAM Mirror enable. Output to AND gates that block address lines (active low)

r_24k_ena[0] If 24k ram extension is disabled, enable ram mirror

r_swap_ena[1] If ram/bios swap is disabled, enable ram mirror.

ROM_ENABLEn

```
assign ROM_ENABLEn = (
s_y0_seln |
                                                                          ~
r_swap_ena[1]
)
```

ROM enable (active low).

s_y0_seln Only select ROM when address range h0000 is enabled.

r_swap_ena[1] If ram/bios swap is disabled, enable ROM.

DECODER INFORMATION FOR U5

How address decoder is created.

s_enable_u5

```
assign s_enable_u5 = (
RFSHn &
                                                                          ~
MREQn
)
```

Enable the the decoder, duplicates U5 functionality from colecovision. always 1, RFSH is a double inversion on coleco (inverter + 138 internal)

RFSHn Z80 Refresh line, when not in refresh enable is active.

MREQn When the MREQn is active then encoder is enabled.

s_y0_seln

```
assign s_y0_seln = ~(
                                                      s_enable_u5 & ~A[15] & ~
A[14] &
                                                                          ~
A[13]
)
```

Address h0000, ROM/RAM

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

s_ram2_csn

```
assign s_ram2_csn = ~(
                                                   s_enable_u5 & ~A[15] & ~
A[14] &
A[13]
)
```

Address h2000, RAM

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

s_ram1_csn

```
assign s_ram1_csn = ~(
                                                    s_enable_u5 & ~A[15] &
A[14] &
                                                                          ~
A[13]
)
```

Address h4000, RAM

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

s_ram0_csn

```
assign s_ram0_csn = ~(
                                                    s_enable_u5 & ~A[15] &
A[14] &
A[13]
)
```

Address h6000, RAM

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

CS_h8000n

```
assign CS_h8000n = ~(
                                                    s_enable_u5 & A[15] & ~
A[14] &
                                                                          ~
A[13]
)
```

Address h8000, Game ROM bank select.

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

CS_hA000n

```
assign CS_hA000n = ~(
                                                    s_enable_u5 & A[15] & ~
A[14] &
A[13]
)
```

Address hA000, Game ROM bank select.

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

CS_hC000n

```
assign CS_hC000n = ~(
                                                    s_enable_u5 & A[15] &
A[14] &
                                                                          ~
A[13]
)
```

Address hC000, Game ROM bank select.

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

CS_hE000n

```
assign CS_hE000n = ~(
                                                    s_enable_u5 & A[15] &
A[14] &
A[13]
)
```

Address hE000, Game ROM bank select.

s_enable_u5 Enable decoder

A[15:13] Address lines used for select lines.

DECODER INFORMATION FOR U6

How address decoder is created

s_enable_u5

Enable the the decoder, duplicates U6 functionality from colecovision.

A[7] Address IO range h80 to hFF

IORQn When the IORQn is active then encoder is enabled.

s_ctrl_en_2n

```
assign s_ctrl_en_2n = ~(
                                                              s_enable_u6 & ~A[6] & ~
A[5] &
                                                                                    ~
WRn
)
```

h80 PORT IO for controller Fire Select

s_enable_u6	Enable decoder
A[6:5]	Address lines used for select lines.
WRn	Select write or read.

CSWn

```
assign CSWn = ~(
                                                               s_enable_u6 & ~A[6] &
A[5] &
                                                                                    ~
WRn
)
```

hBE PORT IO for VDP write

s_enable_u6	Enable decoder
A[6:5]	Address lines used for select lines.
WRn	Select write or read.

CSRn

```
assign CSRn = ~(
                                                               s_enable_u6 & ~A[6] &
A[5] &
WRn
)
```

hBF PORT IO for VDP read

s_enable_u6	Enable decoder
A[6:5]	Address lines used for select lines.
WRn	Select write or read.

s_ctrl_en_1n

```
assign s_ctrl_en_1n = ~(
                                                               s_enable_u6 & A[6] & ~
A[5] &
                                                                                    ~
WRn
)
```

hC0 PORT IO for controller ARM select

s_enable_u6 Enable decoder

A[6:5] Address lines used for select lines.

WRn Select write or read.

SND_ENABLEn

```
assign SND_ENABLEn = ~(
                                                    s_enable_u6 & A[6] &
A[5] &
                                                                      ~
WRn
)
```

hFF PORT IO for sound enable.

s_enable_u6 Enable decoder

A[6:5] Address lines used for select lines.

WRn Select write or read.

s_ctrl_readn

```
assign s_ctrl_readn = ~(
                                                    s_enable_u6 & A[6] &
A[5] &
WRn
)
```

hFC/FF PORT IO for controller read

s_enable_u6 Enable decoder

A[6:5] Address lines used for select lines.

WRn Select write or read.

DECODER INFORMATION FOR SUPER GAME MODULE

How address decoder is created for Super Game Module

SGM IO REG Clocked IO decoder for Super Game Module.

AS

```
assign AS = (
A[7:0]
 =
= 8'h50 & ~IORQn & ~WRn ? 1'b0 : 1'b1
)
```

h50 is the address select, when selected its in data mode

A[7:0] If address matches h50, enable

IORQn Active IO request, enable

WRn Z80 write is active, enable

AY_SND_ENABLEn

```
assign AY_SND_ENABLEn = (
A[7:1]
 =
= 7'b0101000 & ~IORQn & ~WRn ? 1'b0 : 1'b1
)
```

match both h50 and h51 by ignoring bit 0. Enable AY sound chip.

A[7:0] If address matches h50 or h51, enable

IORQn Active IO request, enable

WRn Z80 write is active, enable

AY_SND_ENABLEn

read cached register from previous write (AY emulation).

A[7:0] If address matches h52, enable

IORQn Active IO request, enable

RDn Z80 read is active, enable

CONTROLLER REGISTER READ

How to read controller inputs for player 1 and 2, works with roller and standard gamepads.

CP5_ARM

```
assign CP5_ARM = r_ctrl_arm
```

Activate ARM porition of controllers.

r_ctrl_arm See Also: r_ctrl_arm

CP8_FIRE

```
assign CP8_FIRE = r_ctrl_fire
```

Activate FIRE porition of controllers.

r_ctrl_fire See Also: r_ctrl_fire

D[0]

```
assign D[0] = (
                                                ~s_ctrl_readn & ~A[1] ?
C1P1
:
1'bz
)
```

Data bit zero for P1

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 0, read

D[1]

```
assign D[1] = (
                                            ~s_ctrl_readn & ~A[1] ?
C1P4
:
1'bz
)
```

Data bit one for P1

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 0, read

D[2]

```
assign D[2] = (
                                            ~s_ctrl_readn & ~A[1] ?
C1P2
:
1'bz
)
```

Data bit two for P1

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 0, read

D[3]

```
assign D[3] = (
                                            ~s_ctrl_readn & ~A[1] ?
C1P3
:
1'bz
)
```

Data bit three for P1

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 0, read

D[4]

```
assign D[4] = (
                                            ~s_ctrl_readn & ~A[1] ?
r_mono_p1
:
1'bz
)
```

Data bit one for P1

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 0, read

D[5]

```
assign D[5] = (
                                                    ~s_ctrl_readn & ~A[1] ?
C1P7
:
1'bz
)
```

Data bit five for P1

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 0, read

D[6]

```
assign D[6] = (
                                                    ~s_ctrl_readn & ~A[1] ?
C1P6
:
1'bz
)
```

Data bit six for P1

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 0, read

D[7]

```
assign D[7] = (
                                                    ~s_ctrl_readn & ~A[1] ?
s_int_p1
:
1'bz
)
```

Data bit seven for P1

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 0, read

s_int_p1

```
assign s_int_p1 = ~(
r_mono_p1 &
C1P9
)
```

generate interrupt for player one

r_mono_p1	See Also: r_mono_p1, RC TL emulation
C1P9	Input from controller port. Roller controller only.

D[0]

```
assign D[0] = (
                                                     ~s_ctrl_readn & A[1] ?
C2P1
:
1'bz
)
```

Data bit zero for P2

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 1, read

D[1]

```
assign D[1] = (
                                                     ~s_ctrl_readn & A[1] ?
C2P4
:
1'bz
)
```

Data bit one for P2

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 1, read

D[2]

```
assign D[2] = (
                                                     ~s_ctrl_readn & A[1] ?
C2P2
:
1'bz
)
```

Data bit two for P2

s_ctrl_readn	See Also: s_ctrl_readn, read when active low
A[1]	Address bit 1 is 1, read

D[3]

```
assign D[3] = (
                                                     ~s_ctrl_readn & A[1] ?
C2P3
:
1'bz
)
```

Data bit three for P2

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 1, read

D[4]

```
assign D[4] = (
                                        ~s_ctrl_readn & A[1] ?
r_mono_p2
:
1'bz
)
```

Data bit four for P2

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 1, read

D[5]

```
assign D[5] = (
                                        ~s_ctrl_readn & A[1] ?
C2P7
:
1'bz
)
```

Data bit five for P2

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 1, read

D[6]

```
assign D[6] = (
                                        ~s_ctrl_readn & A[1] ?
C2P6
:
1'bz
)
```

Data bit six for P2

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 1, read

D[7]

```
assign D[7] = (
                                        ~s_ctrl_readn & A[1] ?
s_int_p2
:
```

```
1'bz
)
```

Data bit seven for P2

s_ctrl_readn See Also: s_ctrl_readn, read when active low

A[1] Address bit 1 is 1, read

s_int_p2

```
assign s_int_p2 = ~(
r_mono_p2 &
C2P9
)
```

generate interrupt for player one

r_mono_p1 See Also: r_mono_p1, RC TL emulation

C2P9 Input from controller port. Roller controller only.

INTn

```
assign INTn = ~(
r_int_p1 |
r_int_p2
)
```

INTn is generated by monostable circuit based on NAND outputs.

r_int_p1 See Also: r_int_p1, RC TL emulation

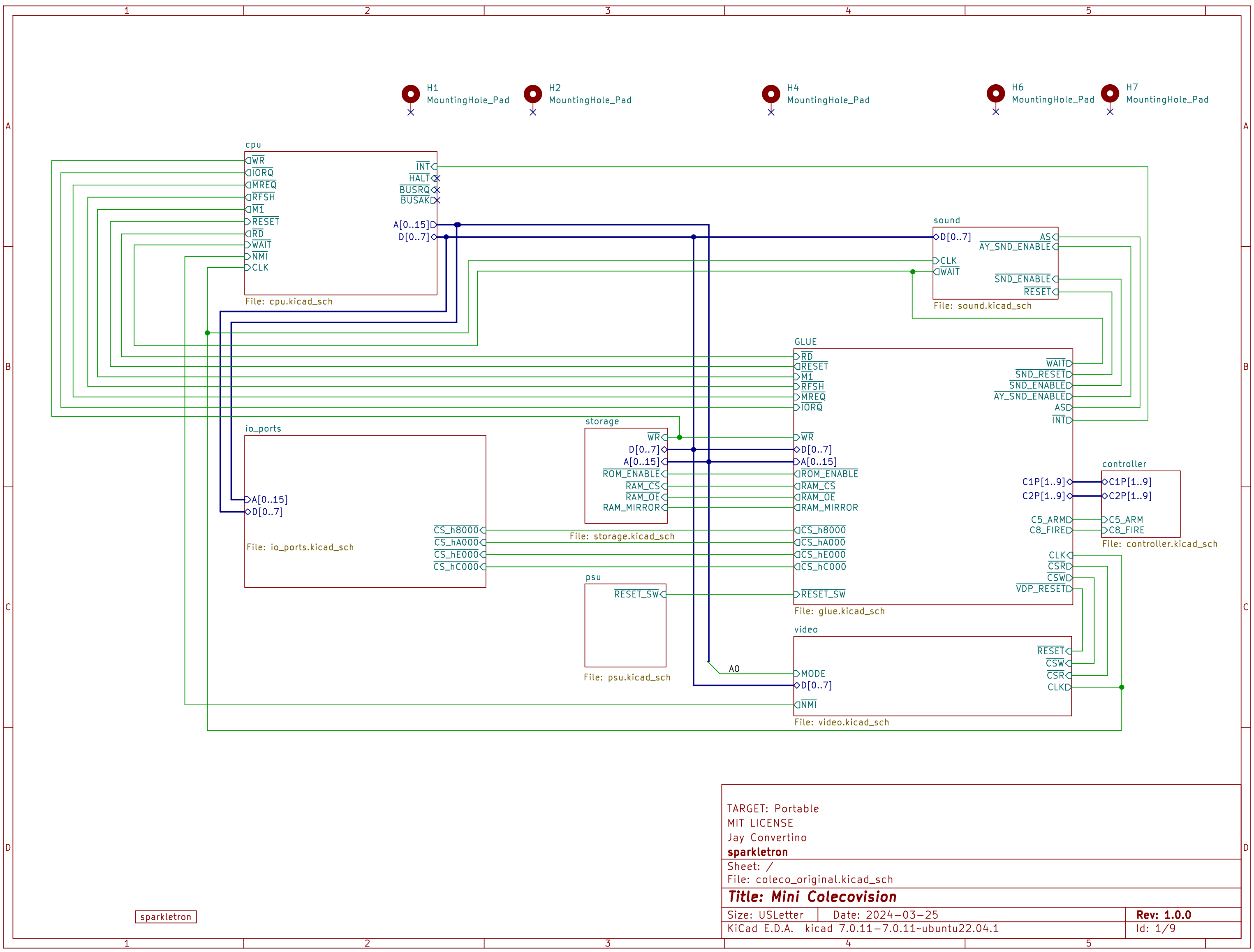
r_int_p2 See Also: r_int_p2, RC TL emulation

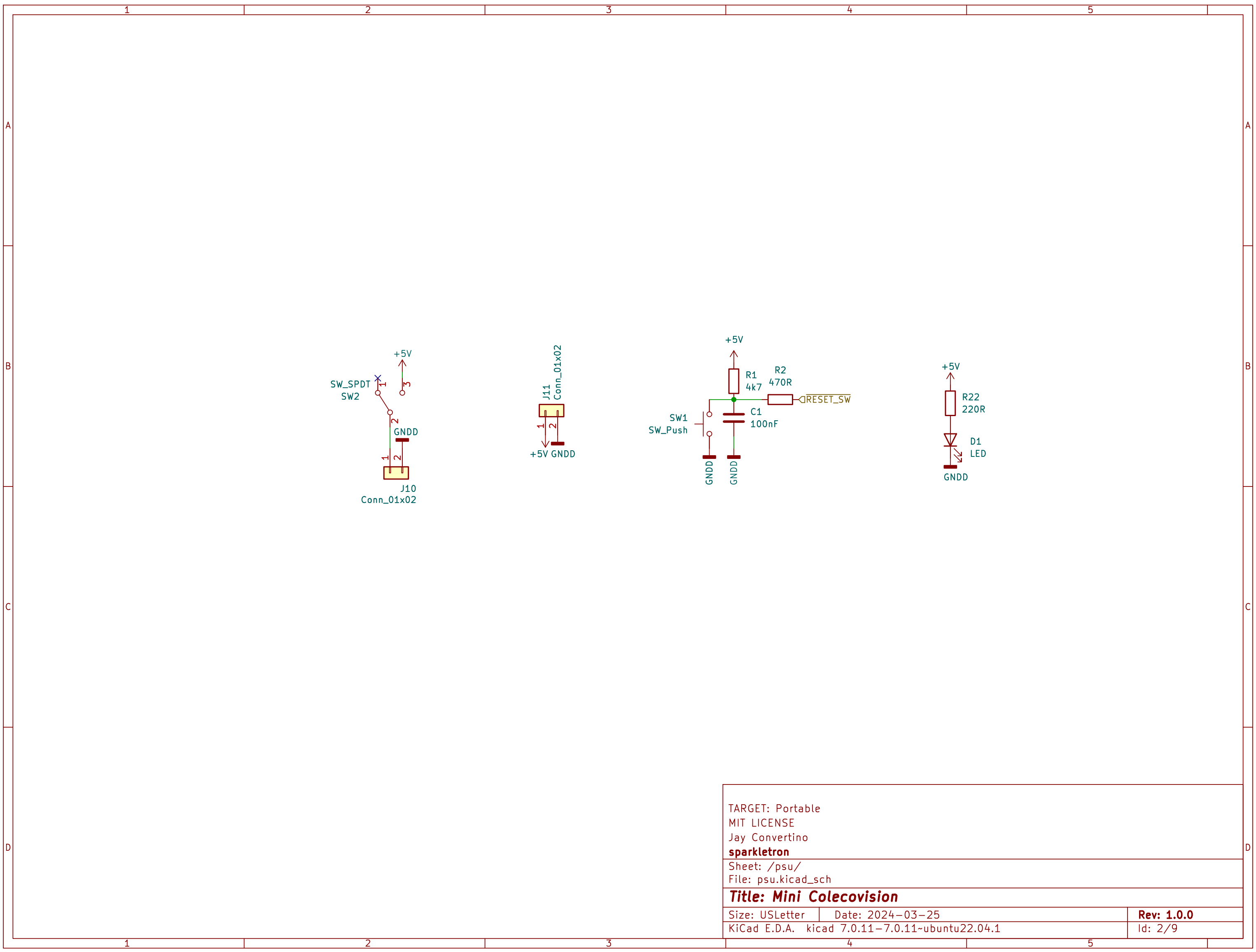
CIRCUIT EMULATION

Everything below emulates a part of the circuit that uses some sort of linear/non-linear components to perform its task. Things such as RC reset circuits, RC interrupts, IRQ and others. See this source file for details.

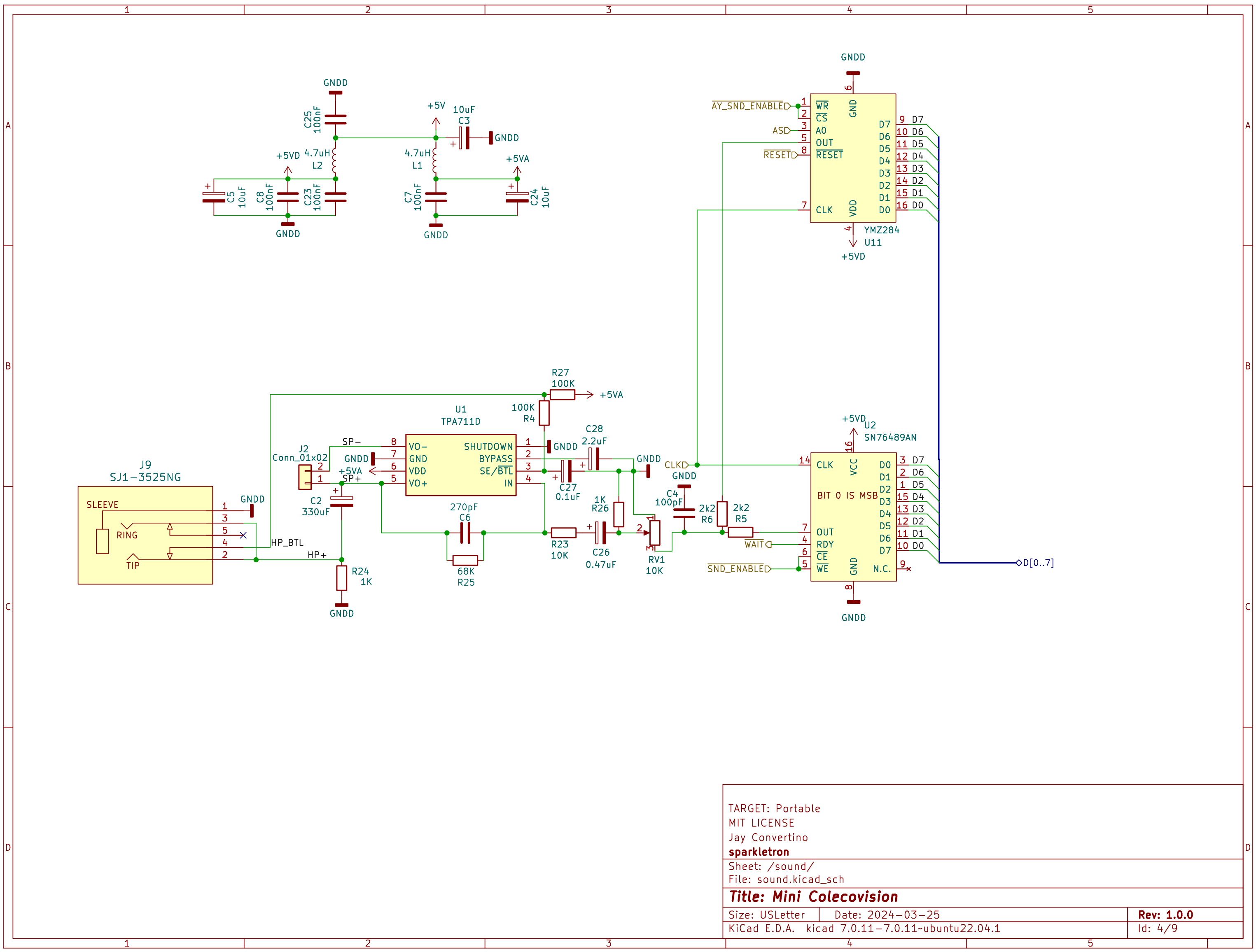
WAIT GENERATE	Generate wait states for the Z80 prosessor
RESET GENERATE	Generate a timed reset for the CPU/VDP/ETC.
TL RC RESET	Generate a interrupt for a monostable circuit that will trigger a 1 for a short duration.
CONTROLLER NAND	Controller NAND Latch FIRE/ARM emulation.
NAND IRQ PULSE	Controller bit 4 is a pulse that represents the spinner state.

5 Schematics

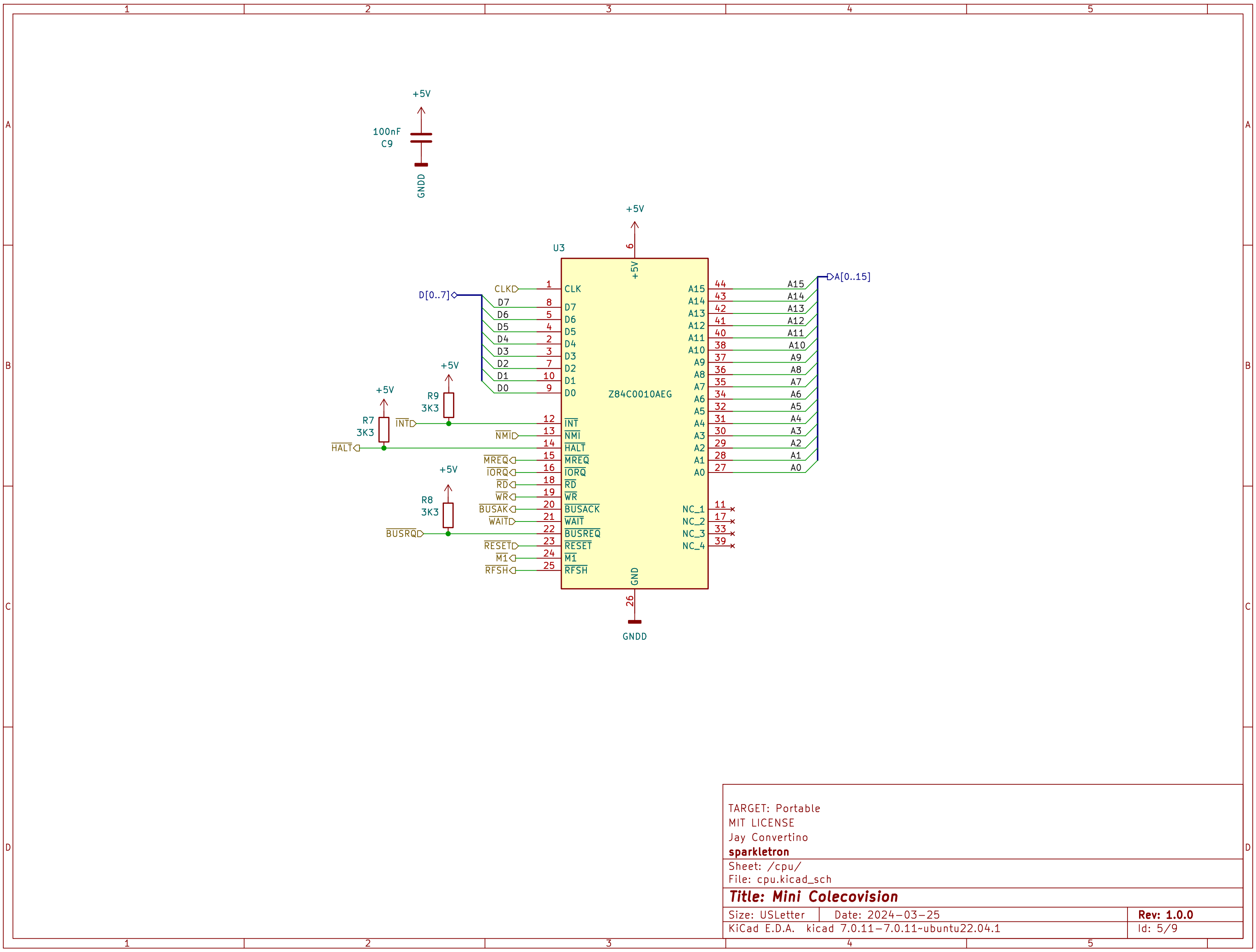

H1 MountingHole_Pad
H2 MountingHole_Pad
H4 MountingHole_Pad
H6 MountingHole_Pad
H7 MountingHole_Pad
cpu
WR IORQ MREQ RFSH M1 RESET RD WAIT NMI CLK
INT HALT BUSRQ BUSAK
A[0..15]
D[0..7]
File: cpu.kicad_sch
sound
D[0..7] CLK WAIT
AS AY_SND_ENABLE SND_ENABLE RESET
File: sound.kicad_sch
GLUE
RD RESET M1 RFSH MREQ IORQ WR D[0..7] A[0..15] ROM_ENABLE RAM_CS RAM_OE RAM_MIRROR CS_h8000 CS_hA000 CS_hE000 CS_hC000 RESET_SW
WAIT SND_RESET SND_ENABLE AY_SND_ENABLE AS INT C1P[1..9] C2P[1..9] C5_ARM C8_FIRE CLK CSR CSW VDP_RESET
File: glue.kicad_sch
controller
C1P[1..9] C2P[1..9] C5_ARM C8_FIRE
File: controller.kicad_sch
io_ports
A[0..15] D[0..7]
CS_h8000 CS_hA000 CS_hE000 CS_hC000
File: io_ports.kicad_sch
storage
WR D[0..7] A[0..15] ROM_ENABLE RAM_CS RAM_OE RAM_MIRROR
File: storage.kicad_sch
psu
RESET_SW
File: psu.kicad_sch
video
A0
MODE D[0..7] NMI
RESET CSW CSR CLK
File: video.kicad_sch
sparkletron
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /
File: coleco_original.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 1/9



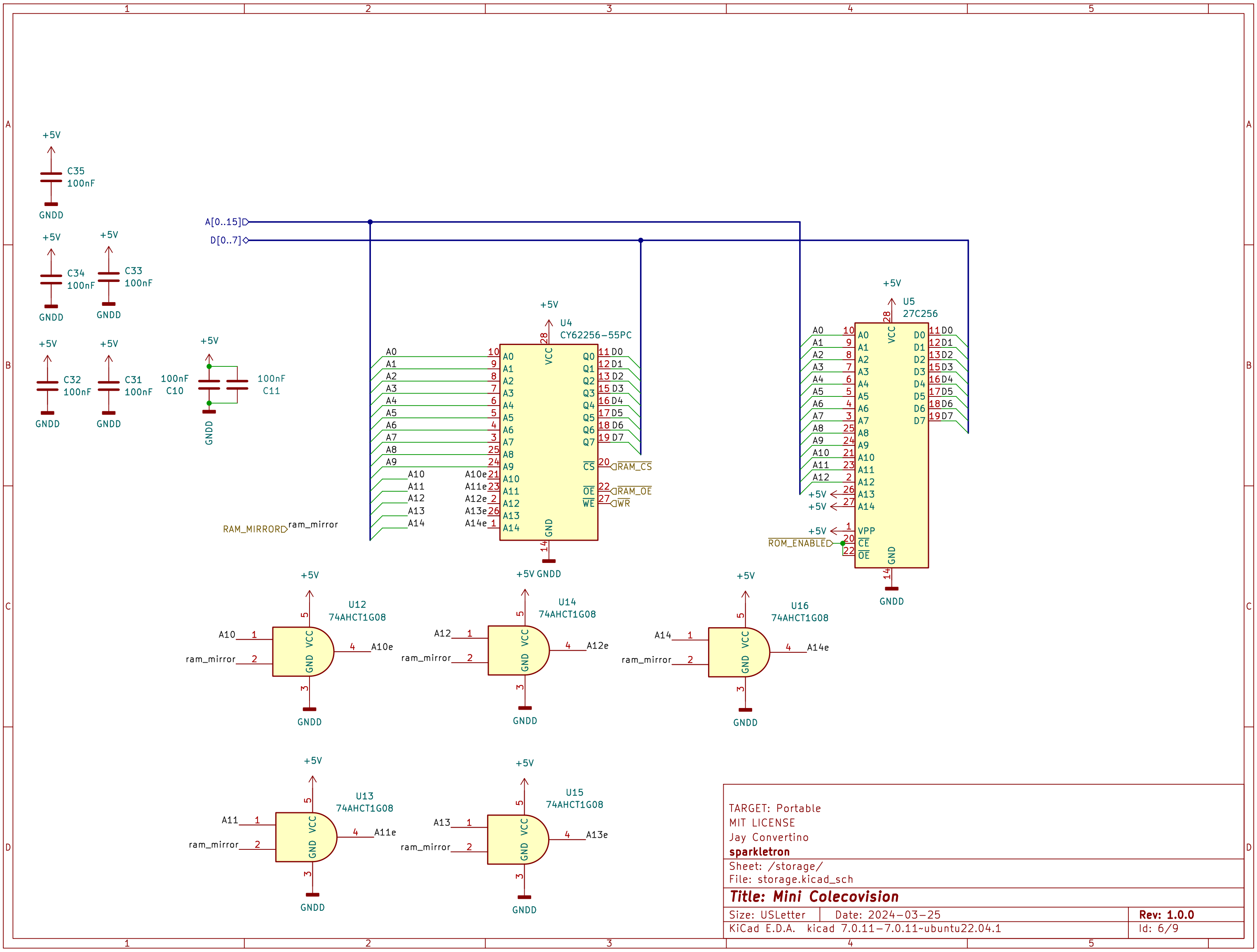
+5V
SW_SPDT
SW2
GNDD
J10
Conn_01x02
J11
Conn_01x02
+5V GNDD
+5V
R1
4k7
R2
470R
RESET_SW
SW1
SW_Push
C1
100nF
GNDD
GNDD
+5V
R22
220R
D1
LED
GNDD
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /psu/
File: psu.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 2/9

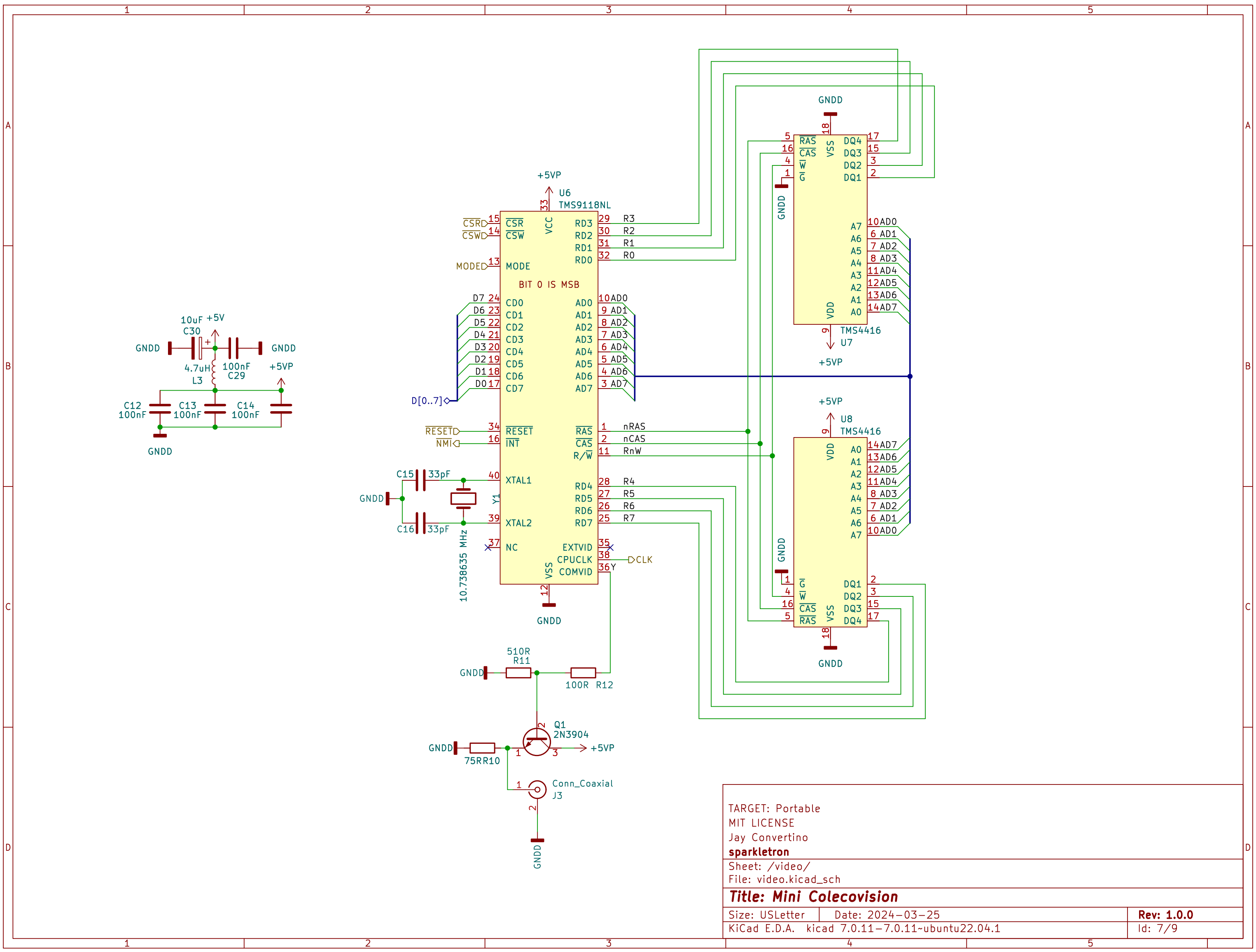


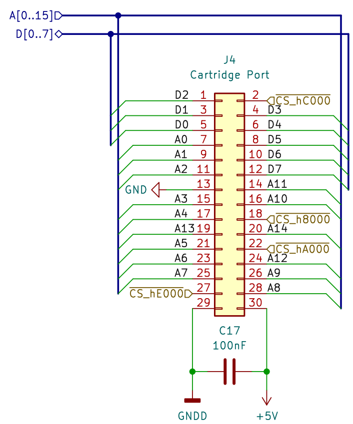
GNDD
C25
100nF
4.7uH
L2
+5VD
C5
10uF
C8
100nF
C23
100nF
+5V
10uF
C3
4.7uH
L1
+5VA
C7
100nF
C24
10uF
AY_SND_ENABLED
ASD
RESET
WR
CS
A0
OUT
RESET
CLK
GND
VDD
D7 D6 D5 D4 D3 D2 D1 D0
YMZ284
U11
+5VD
D[0..7]
R27
100K
+5VA
100K
R4
U1
TPA711D
VO−
GND
VDD
VO+
SHUTDOWN
BYPASS
SE/BTL
IN
C28
2.2uF
C27
0.1uF
1K
R26
C26
0.47uF
R23
10K
RV1
10K
C4
100pF
2k2
R6
2k2
R5
270pF
C6
68K
R25
J2
Conn_01x02
SP−
SP+
C2
330uF
J9
SJ1−3525NG
SLEEVE
RING
TIP
HP_BTL
HP+
R24
1K
CLK
U2
SN76489AN
BIT 0 IS MSB
VCC
OUT
RDY
CE
WE
N.C.
WAIT
SND_ENABLE
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /sound/
File: sound.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024−03−25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11−7.0.11−ubuntu22.04.1
Id: 4/9



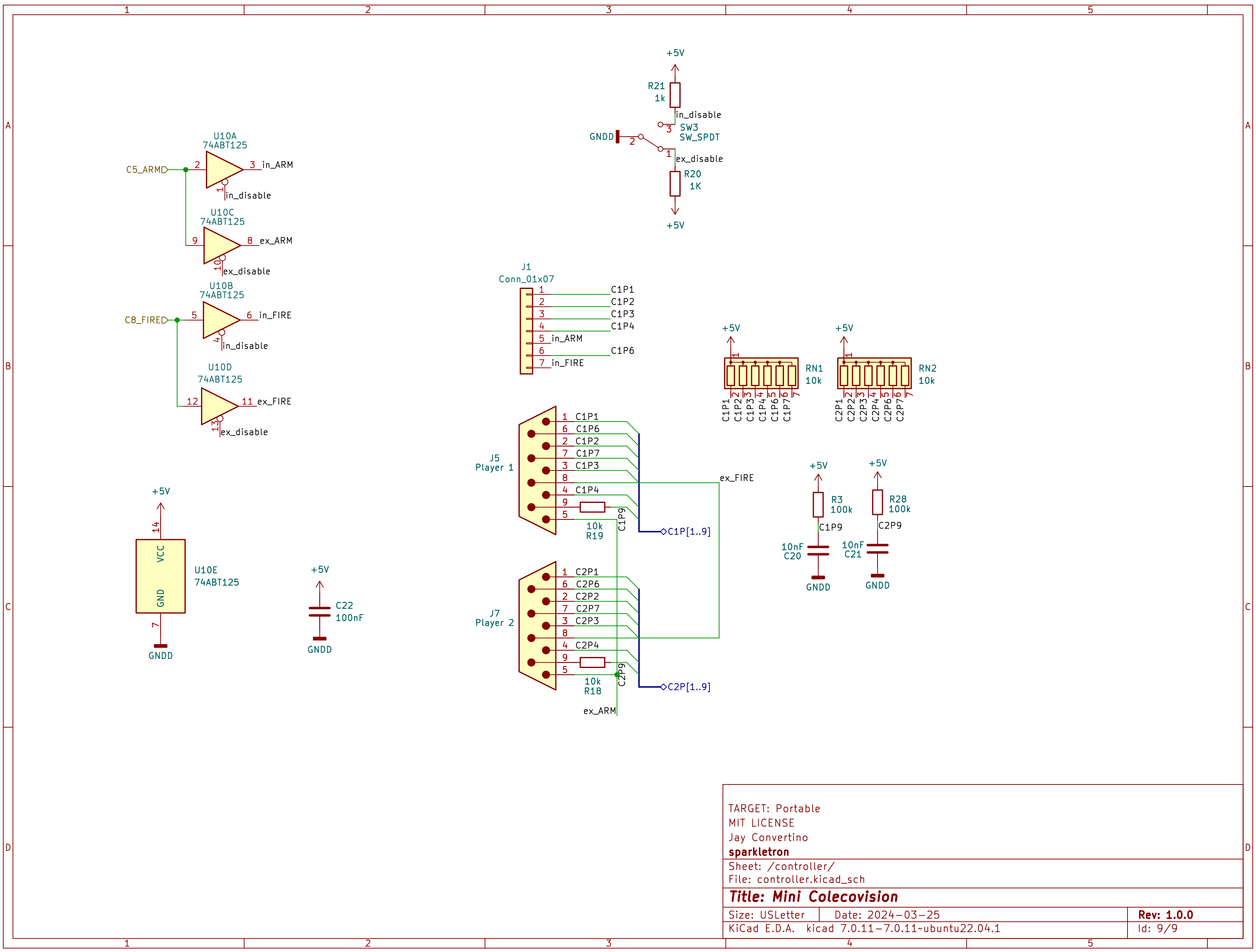
+5V
100nF
C9
GNDD
U3
Z84C0010AEG
D[0..7]
A[0..15]
CLK
D7
D6
D5
D4
D3
D2
D1
D0
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
R9
3K3
R7
3K3
R8
3K3
INT
NMI
HALT
MREQ
IORQ
RD
WR
BUSAK
BUSACK
WAIT
BUSRQ
BUSREQ
RESET
M1
RFSH
NC_1
NC_2
NC_3
NC_4
GND
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /cpu/
File: cpu.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 5/9

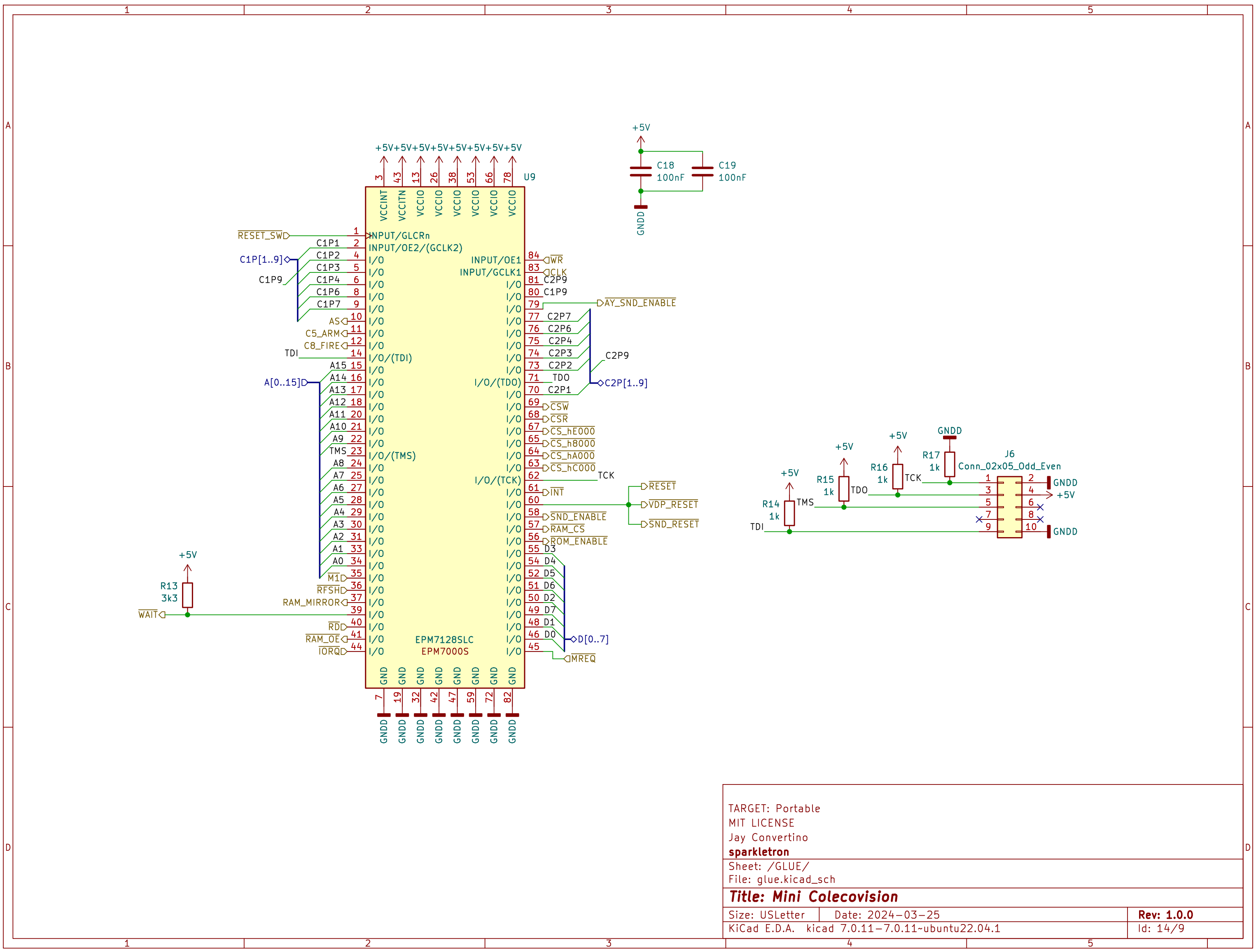

A[0..15]
D[0..7]
C35 100nF
C34 100nF
C33 100nF
C32 100nF
C31 100nF
100nF C10
100nF C11
+5V
GNDD
U4
CY62256-55PC
U5
27C256
RAM_CS
RAM_OE
WR
RAM_MIRROR
ram_mirror
ROM_ENABLED
U12
74AHCT1G08
U13
74AHCT1G08
U14
74AHCT1G08
U15
74AHCT1G08
U16
74AHCT1G08
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /storage/
File: storage.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 6/9


GNDD
+5VP
U6
TMS9118NL
CSR
CSW
MODE
BIT 0 IS MSB
VCC
RD3
RD2
RD1
RD0
R3
R2
R1
R0
CD0
CD1
CD2
CD3
CD4
CD5
CD6
CD7
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
D7
D6
D5
D4
D3
D2
D1
D0
D[0..7]
RESET
INT
NMI
RAS
CAS
R/W
nRAS
nCAS
RnW
XTAL1
XTAL2
C15
33pF
C16
33pF
Y1
10.738635 MHz
RD4
RD5
RD6
RD7
R4
R5
R6
R7
NC
EXTVID
CPUCLK
COMVID
CLK
VSS
TMS4416
U7
U8
DQ1
DQ2
DQ3
DQ4
A0
A1
A2
A3
A4
A5
A6
A7
VDD
10uF
C30
+5V
4.7uH
L3
100nF
C29
C12
100nF
C13
100nF
C14
100nF
510R
R11
100R
R12
Q1
2N3904
75R
R10
Conn_Coaxial
J3
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /video/
File: video.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11~ubuntu22.04.1
Id: 7/9

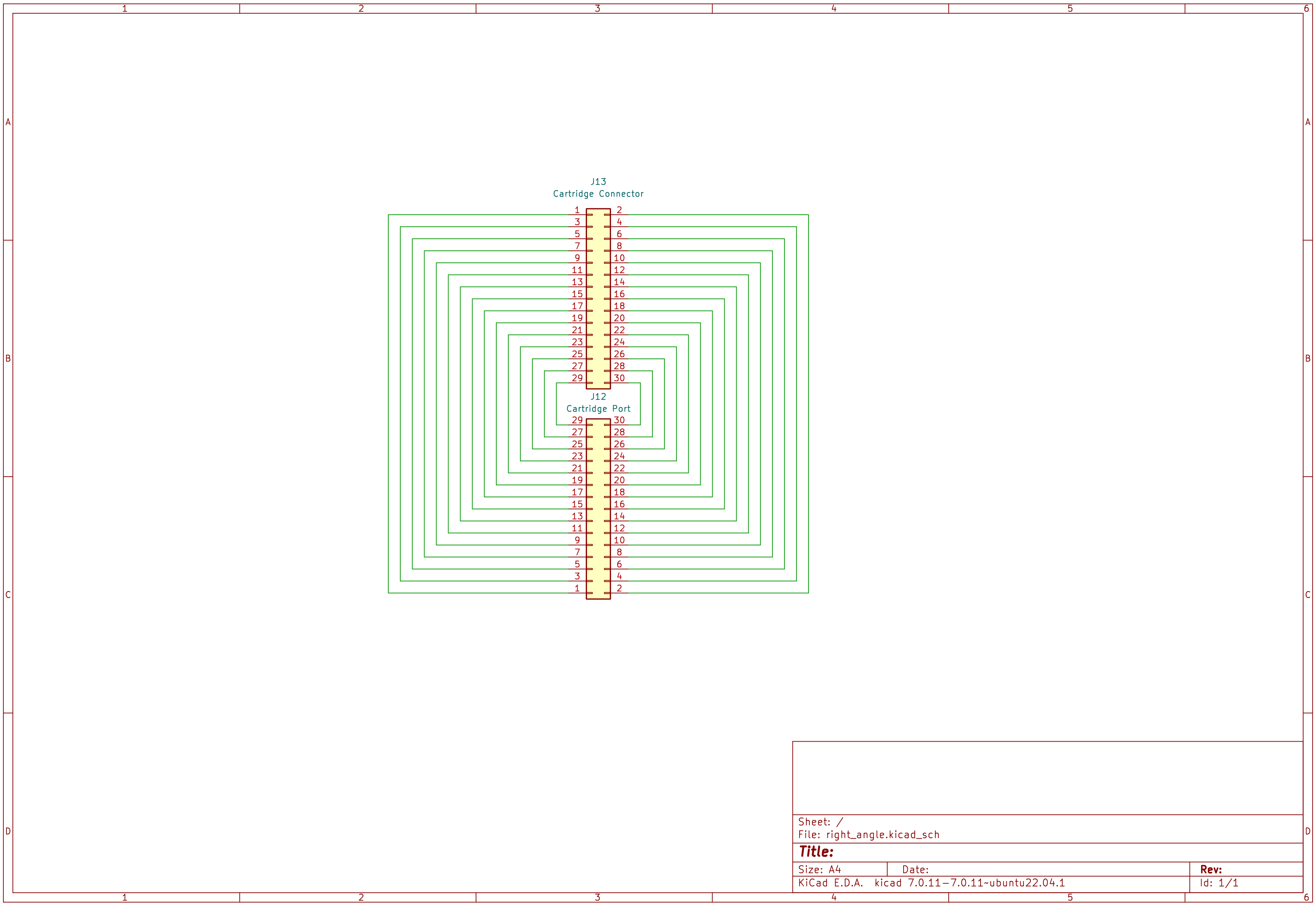


TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /io_ports/
File: io_ports.kicad_sch
Title: Mini Colecovision
Size: USLetter | Date: 2024-03-25 | **Rev: 1.0.0**
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1 | Id: 9/9


+5V
R21
1k
in_disable
SW3
SW_SPDT
GNDD
ex_disable
R20
1K
U10A
74ABT125
C5_ARMD
in_ARM
U10C
74ABT125
ex_ARM
U10B
74ABT125
C8_FIRED
in_FIRE
U10D
74ABT125
ex_FIRE
J1
Conn_01x07
C1P1
C1P2
C1P3
C1P4
C1P6
RN1
10k
RN2
10k
J5
Player 1
C1P[1..9]
10k
R19
J7
Player 2
C2P[1..9]
10k
R18
R3
100k
R28
100k
10nF
C20
10nF
C21
U10E
74ABT125
VCC
GND
C22
100nF
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /controller/
File: controller.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 9/9



U9
EPM7128SLC
EPM7000S
+5V
C18
100nF
C19
100nF
GNDD
RESET_SW
C1P[1..9]
C1P9
AS
C5_ARM
C8_FIRE
TDI
A[0..15]
M1
RFSH
RAM_MIRROR
WAIT
RD
RAM_OE
IORQ
R13
3k3
WR
CLK
AY_SND_ENABLE
C2P9
C2P[1..9]
TDO
CSW
CSR
CS_hE000
CS_h8000
CS_hA000
CS_hC000
TCK
INT
RESET
VDP_RESET
SND_RESET
SND_ENABLE
RAM_CS
ROM_ENABLE
D[0..7]
MREQ
R14
R15
R16
R17
1k
TMS
J6
Conn_02x05_Odd_Even
TARGET: Portable
MIT LICENSE
Jay Convertino
sparkletron
Sheet: /GLUE/
File: glue.kicad_sch
Title: Mini Colecovision
Size: USLetter
Date: 2024-03-25
Rev: 1.0.0
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 14/9



J13
Cartridge Connector
J12
Cartridge Port
Sheet: /
File: right_angle.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 7.0.11-7.0.11-ubuntu22.04.1
Id: 1/1