Open Game Module

May 21, 2025

Jay Convertino

Contents

1 Introduction

Mini Colecovision Compatible Portable 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 Fix for opcode games

Opcode super game module mega cart games have a particular check for the sound chip in there startup routine.

1. Set AY register address to 0x00
2. Write the value 0xAA to AY (register 0x00).
3. Set AY register address to 0x02
4. Write the value 0x55 to AY (register 0x02).
5. Set AY register address to 0x00
6. Read the value from AY (register 0x00).
7. Compare to value originally written (0xAA), fail if not matching
8. Set AY register address to 0x02
9. Read the value from AY (register 0x02).
10. Compare to value originally written (0x55), fail if not matching.

The hack fix for my setup is to only have 4 8 bit regiters for 0,1,2,3,4,5,6, and 7. The mapping is AY Software Register ADDRESS => Cache Register ADDRESS

- 0 => 0
- 1 => 0
- 2 => 1
- 3 => 1
- 4 => 2
- 5 => 2
- 6 => 3
- 7 => 3

1.2 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

1.3 Parts List

1.3.1 main pcb

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

1.3.2 right angle adaptor

Item	Qty	Reference(s)	Value
1	1	J	Cartridge Port

1.3.3 misc electronics

Item	Qty	Reference(s)	Value
1	1	C1	Colecovision Controller, github.com/sparkletron/Colecovision_Controller
2	1	LS1	LCD, LQ035NC111
3	1	LC1	LCD Controller, VSD1612N3 type with SS0101 controller chip or BIT1612
4	1	Battery	3 Cell 10050 mAh Adafruit 5035
5	1	Battery Charger	500 mA charger Adafruit 1944
6	1	Speaker	CMS404928S

1.3.4 hardware

Item	Qty	Reference(s)	Value
1	22	S1	#4-24 x 1/4 Fastenal 0142972, screw
2	9	S2	#4-24 x 1/2 Fastenal 0173613, screw
3	4	S3	#2-32 x 1/4 Fastenal 0148209, screw
4	8	S4	#4-24 x 1/2 Fastenal 0149046, screw black

1.3.5 3D printed hardware

Item	Qty	Reference(s)	Value
1	1	P1	back.stl 100.5
2	4	P2	button.stl 100
3	1	P3	cart_back.stl 100.5
4	1	P4	dpad_large.stl 100
5	2	P5	firebutton.stl 100
6	1	P6	front.stl 100.5
7	1	P7	lcd_hold_mount.stl 100.5
8	12	P8	numpad.stl 100
9	4	P9	serial_strip.stl 100
10	1	P10	speaker_mount.stl 100.5
11	1	P11	speaker_spacer.stl 100

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 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.

2.1.2 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

The 3D printed case has been tested on two different printers. It has only been tested with ABS filament. The parts list for the 3D printed case has the STL file name in the value and the XXX.X value is the scale size. In general I recommend the following steps for assembly.

1. Front, populate side controls w/buttons. PCB will need to be cut down to fit height wise.
2. Front, populate LCD screen.
3. Front, Insert Power LED
4. Front, Route LCD screen wires under gamepad. Glue LED wires to side of case to keep out of DPAD.
5. Front, Mount colecovision_controller with buttons.
6. Front, Insert speaker with spacer.
7. Front, Glue or use speaker_mount to hold speaker.
8. Front, Mount USB battery charger.

9. Cart_back, insert battery, check for wire crimping.
10. Cart_back, mount right angle adaptor.
11. Back, mount cart_back to back.
12. Back, run battery wire to charger
13. Back, run wires from front to main PCB
14. Back, hot glue any wires that seem like they may come off.
15. Back, Mount PCB by inserting the top with the power switch first. Then press it in and watch the reset switch and cart port for binding.
16. Back, Mount serial strips over PCB ports.
17. Finish, mate the back to the front. It should fold together and not give much resistance.

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 used to generate firmware.

4 Module Documentation

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

open_game_module.v

AUTHORS

JAY CONVERTINO

DATES

2025/04/19

INFORMATION

Brief

Colecovision Emulation of the Super Game Module using a YMZ284 for audio.

License MIT

open_game_module

```
module open_game_module (
input
clk,
                                                                input [
10:0]
A,
input
MREQn,
input
RFSHn,
input
IORQn,
input
WRn,
```

```
 input
 RESETn,
 input
 RDn,
                                                          inout [
 7:0]
 D,
 output
 RAM_CSn,
 output
 RAM_OEn,
 output
 AY_CSn,
 output
 AY_AS
)
```

Colecovision Super Game Module Glue Logic

Ports

clk input	Clock for all devices in the core
A input[10: 0]	Address input bus from Z80
MREQn input	Z80 memory request input, active low
RFSHn input	Refresh request, active low
IORQn input	Z80 IO request input, active low
WRn input	Z80 Write to bus, active low
RESETn input	Input for reset, active low
RDn input	Z80 Read from bus, active low
D inout[7: 0]	Z80 8 bit data bus, tristate IN/OUT
RAM_CSn output	RAM chip select, active low
RAM_OEn output	RAM Ouput enable, active low
RAM_WEn	RAM write enable, active low
AY_CSn output	AY sound chip chip select
AY_AS output	AY data or register select

REGISTER INFORMATION

Core has 3 registers at the IO addresses that follow.

SOUND_ADDR_CACHE	h50
SOUND_CACHE	h51
RAM_24K_ENABLE	h53

SWAP_BIOS_TO_RAM h7F

SOUND_ADDR_CACHE

```
localparam SOUND_ADDR_CACHE = 8'h50
```

Defines the address of r_snd_addr_cache

SOUND ADDR CACHE REGISTER
1:0
CACHE LAST ADDRESS WRITE TO AY SOUND CHIP, BITS 2:1

Setup an address for cache the sound address so each write will be to a proper address (opcode games need to write multiple and read multiple addresses) The resister is only 4 bits since there are only 16 registers max.

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_snd_addr_cache

```
reg [ 1:0] r_snd_addr_cache = 0
```

register for SOUND_ADDR_CACHE See Also: SOUND_ADDR_CACHE

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[3:0]
```

register for SOUND_CACHE See Also: SOUND_CACHE

ASSIGNMENT INFORMATION

How signals are created

s_enable

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

Decided to keep the same method used internally as the coleco. This emualtes the original ttl chip logic.

s_y0_seln

```
assign s_y0_seln = ~(
                                                    s_enable & ~A[10] & ~
A[9] &
                                                                       ~
A[8]
)
```

Address h0000, ROM/RAM

s_enable Enable decoder

A[10:8] Address lines used for select lines (actually lines A[15:13]).

s_ram2_csn

```
assign s_ram2_csn = ~(
                                                    s_enable & ~A[10] & ~
A[9] &
A[8]
)
```

Address h2000, RAM

s_enable Enable decoder

A[10:8] Address lines used for select lines (actually lines A[15:13]).

s_ram1_csn

```
assign s_ram1_csn = ~(
                                                     s_enable & ~A[10] &
A[9] &
                                                                       ~
A[8]
)
```

Address h4000, RAM

s_enable Enable decoder

A[10:8] Address lines used for select lines (actually lines A[15:13]).

s_ram0_csn

```
assign s_ram0_csn = ~(
                                                     s_enable & ~A[10] &
A[9] &
A[8]
)
```

Address h6000, RAM

s_enable Enable decoder

A[10:8] Address lines used for select lines (actually lines A[15:13]).

s_ram_csn

```
assign s_ram_csn = (
(s_y0_seln | r_swap_ena[1]) & (s_ram2_csn | ~r_24k_ena[0]) & (s_ram1_csn |
                                                                    (
s_ram0_csn | ~r_24k_ena[0])
)
```

RAM Chip select when address is requested (active low). When the 24k is not enabled, use internal memory.

(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 | ~r_24k_ena[0]) address range starting at h6000, 24k enable bit from register.

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

DECODER INFORMATION FOR SUPER GAME MODULE

How address decoder is created for Super Game Module, using a YMZ284.

SGM IO REG Clocked IO decoder for Super Game Module.

AY_AS

```
assign AY_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

s_ay_sound_csn

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

D

```
assign D = (
A[7:0]
 =
= 8'h52 & ~IORQn & ~RDn ? r_snd_cache[{2'b00, r_snd_addr_cache}] : 8'bzzzz
)
```

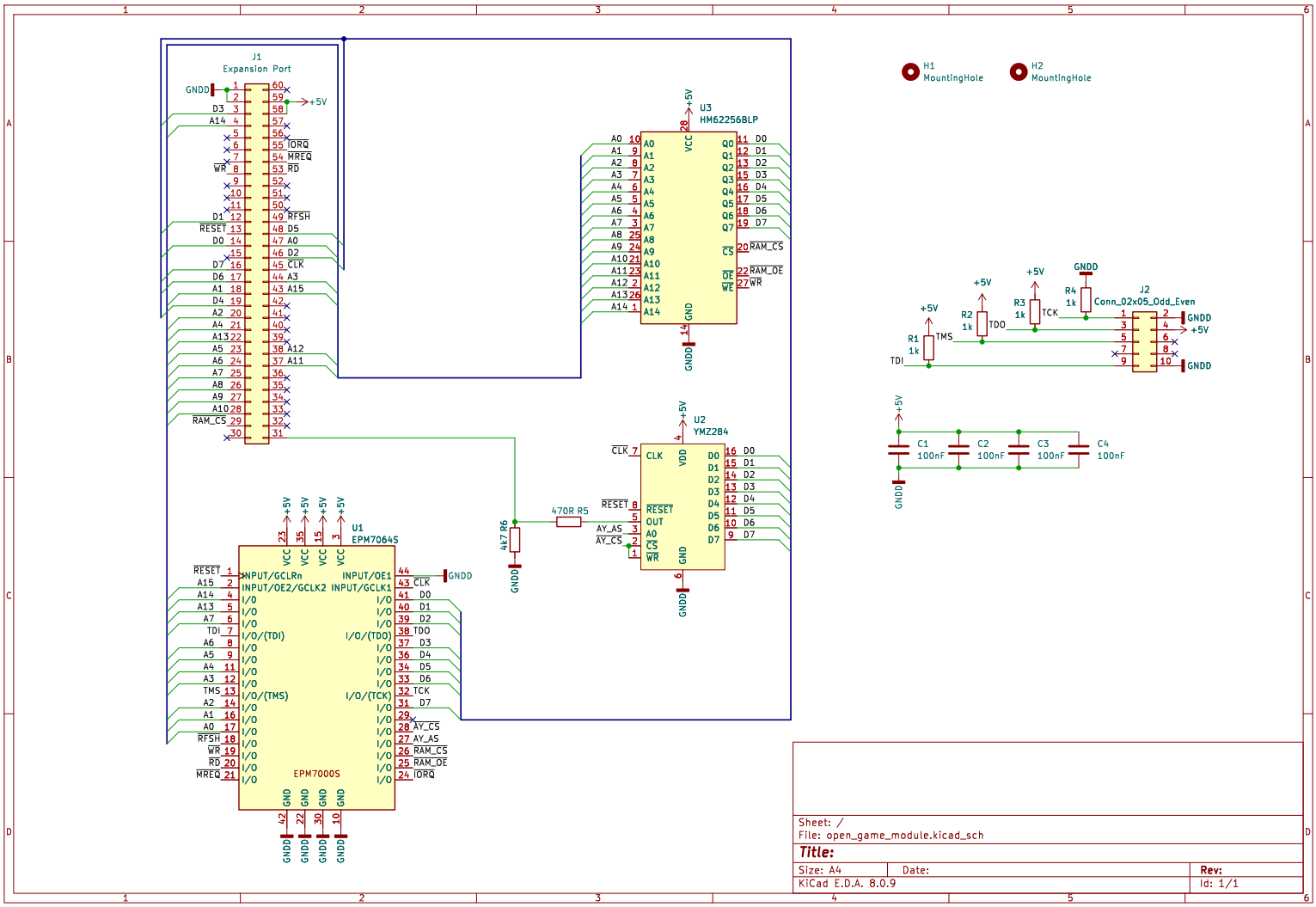
read cached register from previous write (AY emulation), at set address location.

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

IORQn Active IO request, enable

RDn Z80 read is active, enable

5 Schematics


J1
Expansion Port
U3
HM62256BLP
U2
YMZ284
U1
EPM7064S
EPM7000S
H1
MountingHole
H2
MountingHole
J2
Conn_02x05_Odd_Even
R1 1k
R2 1k
R3 1k
R4 1k
470R R5
4k7 R6
C1 100nF
C2 100nF
C3 100nF
C4 100nF
+5V
GNDD
TMS
TDI
TDO
TCK
RAM_CS
RAM_OE
AY_AS
AY_CS
Sheet: /
File: open_game_module.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. 8.0.9
Id: 1/1