

# Open Game Module

## SPARKLETRON

May 21, 2025

Jay Convertino

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Fix for opcode games . . . . .	2
1.2	Specifications . . . . .	3
1.3	Parts List . . . . .	3
1.3.1	electronics . . . . .	3
1.3.2	hardware . . . . .	3
<b>2</b>	<b>Building</b>	<b>3</b>
2.1	Dependencies . . . . .	4
2.1.1	Open Game Module Glue File List . . . . .	4
2.1.2	Fusesoc . . . . .	4
2.1.3	Open Game Module Glue Targets . . . . .	4
2.1.4	Quartus . . . . .	5
2.2	PCB . . . . .	5
2.3	3D Printed Case . . . . .	6
2.4	Programming . . . . .	6
2.4.1	CPLD . . . . .	6
<b>3</b>	<b>Usage</b>	<b>7</b>
3.1	Directory Guide . . . . .	7
<b>4</b>	<b>Module Documentation</b>	<b>8</b>
4.1	Open Game Module . . . . .	9
<b>5</b>	<b>Schematics</b>	<b>16</b>
5.1	Open Game Module . . . . .	17

# 1 Introduction

Open Game Module is a open source expansion card for the Colecovision. This module is Super Game Module (SGM) compatable. This means all SGM games will run when this module is used with a Colecovision. The sound is expanded with a Yamaha YMZ284 instead of the GI AY chip.

## 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

- 32 KB of RAM
- YMZ284 Sound Chip
- MAX7000S CPLD (EPM7064SLC)
- PCB, two layer

## 1.3 Parts List

### 1.3.1 electronics

Item	Qty	Reference(s)	Value
1	1	J1	Connector, 60 pin
2	4	C1, C2, C3, C4	100pF
3	1	U1	EPM7064SLC-10
4	1	U2	YMZ284
5	1	U3	HM62256BLP
6	1	R5	470R
7	1	R6	4k7
8	4	R1, R2, R3, R4	1k
9	1	J2	Pin Header, 10 pin

### 1.3.2 hardware

Item	Qty	Reference(s)	Value
1	4	S1	#2-32 x 1/4 Fastenal 0148209, screw
2	1	TOP	top.stl 100.00
3	1	BOTTOM	bottom.stl 100.00

## 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 PCB from schematic/gerber/open\_game\_module.zip
- Populate 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 v8.X

### 2.1.1 Open Game Module Glue File List

- src
  - 'src/open\_game\_module.v': 'file\_type': 'verilogSource'
- constr
  - 'constr/open\_game\_module.sdc': 'file\_type': 'SDC'
- tb
  - 'tb/tb\_open\_game\_module.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 Open Game Module Glue Targets

- default
  - Info: Default IP target for future tool intergration.
  - src
  - constr
- sim
  - Info: Simulation target for basic test bench.
  - src
  - tb

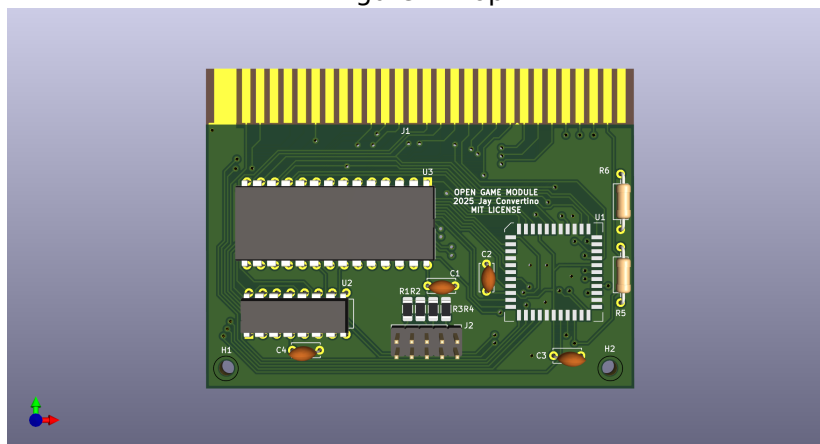
### 2.1.4 Quartus

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

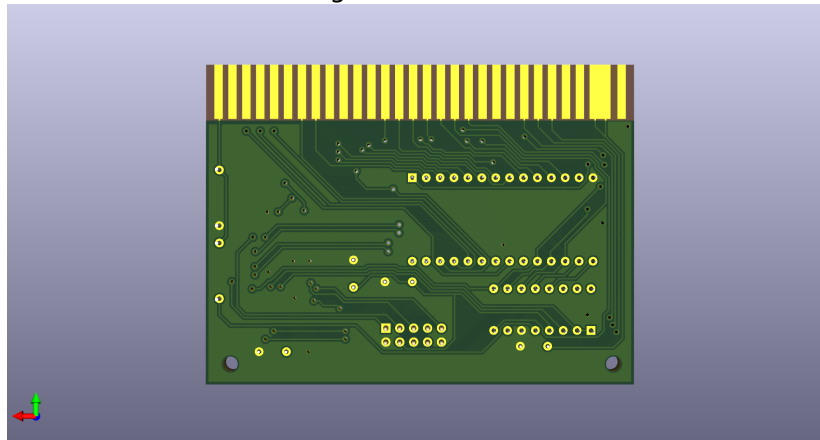
The top has all the components of the circuit.

Figure 1: Top



The bottom has no components.

Figure 2: Bottom



## **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. Bottom, sit pcb aligned to screw holes.
2. Top, sit top half on top and install screws.

## **2.4 Programming**

There is one device that need to be programmed which is the CPLD (complex programmable logic device). This is programmed with quartus using the JTAG header to upload the bitfile.

### **2.4.1 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

The expansion module must be firmly inserted into the expansion connector of the Colecovision. If you have issues, be sure to clean the connectors.

1. **Open Expansion Cover** 
2. **Insert Open Game Module** 

### 3.1 Directory Guide

Below highlights important folders from the root of the open\_game\_module.

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 v8.X schematic and PCB designs
  - **gerber** Contains gerber files and archives for production.
  - **pdf** PDF schematic
4. **src** CPLD firmware source
  - **open\_game\_module** Contains verilog source code and constraints
  - **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

---

Copyright 2025 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## open\_game\_module

---

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

input [

```

    input
    RESETn,
    input
    RDn,

    7:0]
    D,
    output
    RAM_CSn,
    output
    RAM_OEn,
    output
    AY_CSn,
    output
    AY_AS
)

```

inout [

Colecovision Super Game Module Glue Logic

## Ports

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

## REGISTER INFORMATION

Core has 3 registers at the IO addresses that follow.

<b>SOUND_ADDR_CACHE</b>	h50
<b>SOUND_CACHE</b>	h51
<b>RAM_24K_ENABLE</b>	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 register 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\_selIn

```

assign s_y0_sel0 = ~(
                                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'bzzzzz
)
```

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**

