# Open Game Module


May 24, 2025

Jay Convertino

# Contents

# 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 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 constraits
    - **quartus13sp01** Quartus project used to generate firmware.

## 2.2 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

### 2.2.1 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'

### 2.2.2 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.

### 2.2.3 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

### 2.2.4 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.

## 2.3 PCB

The top has all the components of the circuit.
The bottom has no components.

## 2.4 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.

Figure 1: Top

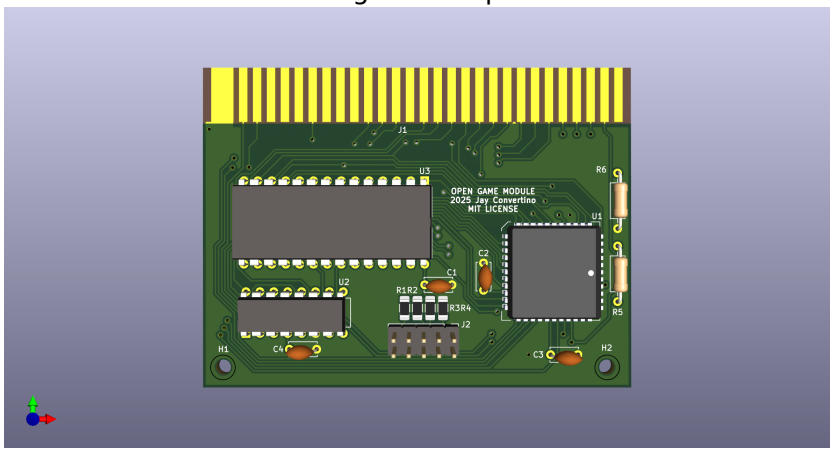
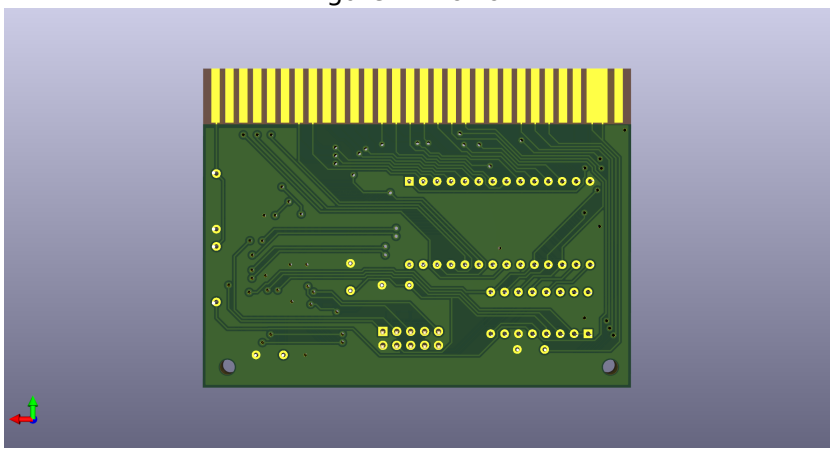



Figure 2: Bottom



## 2.5 CPLD 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.

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 Console

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** 


## 3.2 Software Programming

Module Documentation has indepth details for using the open game module and register bits. You could do this in many different ways, but having the SDCC compiler know ahead of time that the RAM is there works well for me. There are different stragies that could be done for this but I feel this is the simplest way. Other methods can provide better detection of the OGM. The psudo code steps are as follows.

1. Setup expansion RAM, (NOTE COULD CHECK IF REGISTER VALUE IS EQUAL TO THE DEFAULT TO VERIFY SGM)
2. Setup IRQS
3. Copy IRQs to RAM where ROM jumps used to be.
4. Setup Sound CHIP, (READ BACK TO CHECK EXISTANCE)

### 3.2.1 Assembly Code For RAM Init

This brief assembly code doesn't actually check for the SGM it simply enables the ram registers, sets the irqs, and starts main. This code is taken from RODAC and isn't complete, see RODAC coleco_sgm arch crt0.s for complete code. I does give a decent idea on how to set the SGM registers for RAM.

```
_SGM_RAM_ENA_PORT       .equ 0x0053
_SGM_BIOS_SWAP_PORT     .equ 0x007F
_NMI_SIZE               .equ (_irq_nmi_end - _irq_nmi + 1)
_SPIN_SIZE              .equ (_irq_spin_end - _irq_spin + 1)
;setup for super game module.
ld a,#0x01
out (_SGM_RAM_ENA_PORT), a
ld a,#0x0D
out (_SGM_BIOS_SWAP_PORT), a
```

```
;setup ram locations with irq vectors?, 0x8 etc.
;reti
ld a,#0xED
ld (0x08),a
ld (0x10),a
ld (0x18),a
ld (0x20),a
ld (0x28),a
ld (0x30),a
ld a,#0x4D
ld (0x09),a
ld (0x11),a
ld (0x19),a
ld (0x21),a
ld (0x29),a
ld (0x31),a
;copy code to ram
;nmi first
ld bc, #_NMI_SIZE
ld hl, #_irq_nmi
ld de, #0x0066
ldir
;spin
ld bc, #_SPIN_SIZE
ld hl, #_irq_spin
ld de, #0x0038
ldir
ld      sp, #0x8000
call  gsinit
call   _main
rst    0x0
```

### 3.2.2 C Code for SDCC

The code below is based on using SDCC 4.X.X as the C complier with a custom crt0.s. Essentially for the sound chip you need to communicate with it by writing to the ports a address, then the data. You can also read from the ports to verify the chip exists and that the OGM is working. Though for the OGM its chip does not have a read function. It is emulated for registers 0, 2, and 4 (0=1, 2=3, and 4=5) in the CPLD so all opcode/team pixelboy games will startup correctly.

```
#define GI_SND_CP_ADDR      0x50
#define GI_SND_WDATA_ADDR 0x51
#define GI_SND_RDATA_ADDR 0x52
```

```
__sfr __at(GI_SND_CP_ADDR)    GI_SND_CP_PORT;
__sfr __at(GI_SND_WDATA_ADDR) GI_SND_WDATA_PORT;
__sfr __at(GI_SND_RDATA_ADDR) GI_SND_RDATA_PORT;

/*** send address to chip ***/
inline void sendAddr(uint8_t addr)
{
  di();

  GI_SND_CP_PORT = addr;

  ei();
}

/*** send data to chip ***/
inline void sendData(uint8_t data)
{
  di();

  GI_SND_WDATA_PORT = data;

  ei();
}

/*** send data to chip ***/
inline void readData(uint8_t data)
{
  di();

  data = GI_SND_RDATA_PORT;

  ei();
}
```

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