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 procressor

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.