Sega Genesis Controller Interfacing

Mason Strong, Stephen Just 2016-03-13

1 Introduction

The Sega Genesis was an old 16-bit game console that was released in North America in 1989. [1]

This console features support for two gamepads. Each gamepad has four directional buttons, a "Start" button, and either three or six action buttons, depending on model of controller. The three-button controller has a much simpler interface than the six-button controller, making use of a multiplexer and no other logic to access all of the buttons. This document focuses only on the three-button Genesis controller. The Genesis controllers also use a standard DB-9 connector, unlike most other game consoles which use proprietary connectors. [2]

Figure 1: Genesis Controller

2 Controller Interface

The Genesis controller uses a female DB-9 connector to interface with a Genesis console or other device. The pins on this connector are configured as follows:

Pin	Func (select low)	Func (select high)
1	up button	up button
2	down button	down button
3	logic low	left button
4	logic low	right button
5	Power (+5 volts)	Power (+5 Volts)
6	A button	B button
7	select signal	select signal
8	Ground	Ground
9	Start button	C button

While the controller was designed for +5 Volts for power, because of its simple design, it is possible to determine that it is actually capable of 2 - 6 Volts. This is possible because the controller only contains a single 74HC157 multiplexer chip inside, whose datasheet specifies that the device is operable within that range, with varying delay times. [3]

In order to read the buttons on a controller, the master device should apply a logic high or low to the select pin of the controller, and then query the state of each of the button pins. Then the master device can switch the state of the select pin, and then query the values for the other buttons. When a button is pressed, its value will be logic low. Buttons that are not pressed will appear as a logic high. Note that reading the up and down buttons of the controller are not affected by the select signal, as they are connected directly to the controller plug and not through the multiplexer.

3 Notes

On a real Genesis console, the controller's value is read once per video frame, or 60 times per second. That means that if you are trying to emulate a controller, the emulated buttons should remain pressed for at least 1/60th of a second, to ensure that the input is received by the console. Shorter button presses could be missed entirely.

Be aware that the six-button gamepad has a more complicated interface protocol. The extra buttons are accessed by toggling the select line on the controller three times in quick succession. If you want your application to tolerate six-button controllers, take care not to do this. The six-button controllers should not go in to this mode if you only toggle the select line once per frame. This appnote targets the standard three-button genesis controllers and corresponding protocol.

4 Project Setup

Ensure that you are in possession of a three-button genesis controller, and a corresponding adapter board to interface the DB9 connector with the appropri-

ate GPIO pins on the DE2. Should an adapter board be unavailable, a 40-pin ribbon cable breadboarded with a male DB9 connector according to the pinout described in Figure 2.

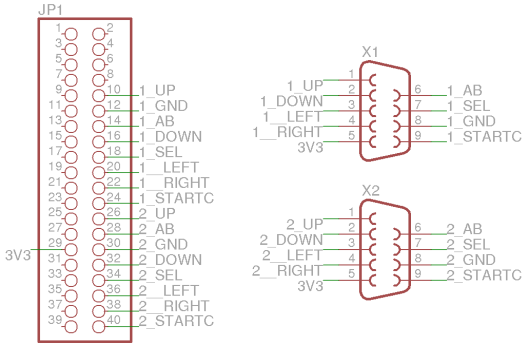


Figure 2: Connection Schematic

In a typical video game system, you might want to trigger the Genesis controller from a 60Hz pulse, such as a video VSync signal. However, to simplify this demo, there is a simple process in the VHDL top-level file to generate a pulse every 50ms to trigger the Genesis controller block.

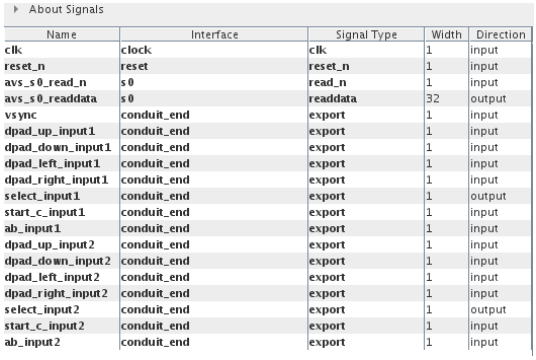
4.1 Qsys

In order to use a genesis controller, a custom component must be created in Qsys. This component is provided in the companion code to this application note, found in the `ip` directory. The following instructions detail how you might recreate this block, and set up a Qsys system to use it. The provided demo system is set up slightly differently, to use LEDs as an output.

1. Name the component `genesis`
2. Provide `custom_controller.vhd` as the synthesis file
3. Setup the signals as shown below in figure 3:
4. Configure the component interfaces as in figures 4, 5, and 6:
5. Add the custom component to your system. Connect `clock`, `s0`, and `reset` appropriately. Ensure that `conduit_end` is exported properly. An example of this configuration is shown in figure 7.
6. Regenerate base addresses in the event of conflicts.

4.2 Quartus

The Genesis controllers are connected to the DE2 via one of the GPIO ports. In our system, we used `GPIO_1`. To ensure correct controller behaviour, the GPIO



Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
reset_n	reset	reset_n	1	input
avs_s0_read_n	s0	read_n	1	input
avs_s0_readdata	s0	readdata	32	output
vsync	conduit_end	export	1	input
dpad_up_input1	conduit_end	export	1	input
dpad_down_input1	conduit_end	export	1	input
dpad_left_input1	conduit_end	export	1	input
dpad_right_input1	conduit_end	export	1	input
select_input1	conduit_end	export	1	output
start_c_input1	conduit_end	export	1	input
ab_input1	conduit_end	export	1	input
dpad_up_input2	conduit_end	export	1	input
dpad_down_input2	conduit_end	export	1	input
dpad_left_input2	conduit_end	export	1	input
dpad_right_input2	conduit_end	export	1	input
select_input2	conduit_end	export	1	output
start_c_input2	conduit_end	export	1	input
ab_input2	conduit_end	export	1	input

Figure 3: Genesis Component Signals

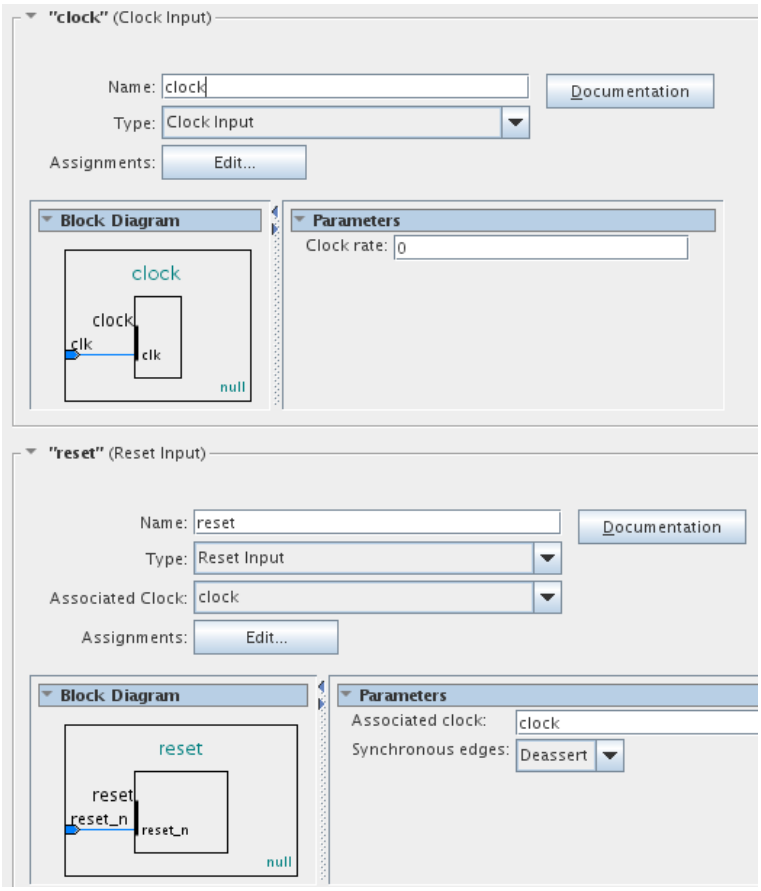


Figure 4: Genesis Component Clock and Reset Configuration

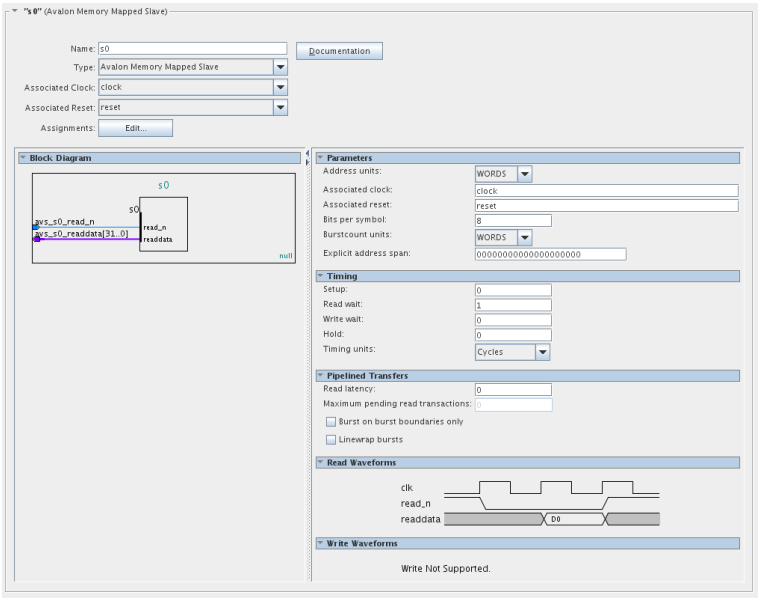


Figure 5: Genesis Component MM Slave Configuration

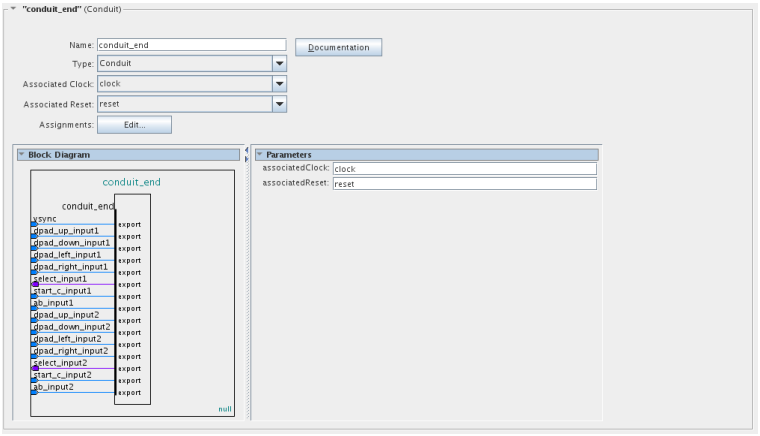


Figure 6: Genesis Component Conduit Configuration

connector should be configured as high-impedance by default, as is shown in Listing 1.

Listing 1: Entity Port for Genesis Controllers

```
VGA_VS : buffer std_logic;
GPIO_1 : inout std_logic_vector (35 downto 0) := (others => 'Z');
```

The Qsys system containing the Genesis controller interface component in-

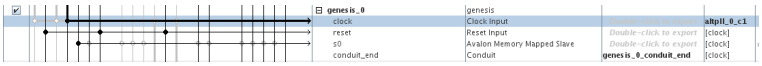


Figure 7: Genesis Component Qsys Connections

cludes the ports similar to the ones shown in Listing 2 within its component declaration.

Listing 2: Qsys System Ports for Genesis Controller

```
genesis_0_conduit_end_vsync  : in    std_logic  := 'X';
    -- trigger signal, usually vsync
genesis_0_conduit_end_dpad_up_input1            : in    std_logic  :=
     'X';             -- dpad_up_input1
genesis_0_conduit_end_dpad_down_input1          : in    std_logic  :=
     'X';             -- dpad_down_input1
genesis_0_conduit_end_dpad_left_input1          : in    std_logic  :=
     'X';             -- dpad_left_input1
genesis_0_conduit_end_dpad_right_input1         : in    std_logic  :=
     'X';             -- dpad_right_input1
genesis_0_conduit_end_select_input1             : out   std_logic;
                       -- select_input1
genesis_0_conduit_end_start_c_input1            : in    std_logic  :=
     'X';             -- start_c_input1
genesis_0_conduit_end_ab_input1                 : in    std_logic  :=
     'X';             -- ab_input1
genesis_0_conduit_end_dpad_up_input2            : in    std_logic  :=
     'X';             -- dpad_up_input2
genesis_0_conduit_end_dpad_down_input2          : in    std_logic  :=
     'X';             -- dpad_down_input2
genesis_0_conduit_end_dpad_left_input2          : in    std_logic  :=
     'X';             -- dpad_left_input2
genesis_0_conduit_end_dpad_right_input2         : in    std_logic  :=
     'X';             -- dpad_right_input2
genesis_0_conduit_end_select_input2             : out   std_logic;
                         -- select_input2
genesis_0_conduit_end_start_c_input2            : in    std_logic  :=
     'X';             -- start_c_input2
genesis_0_conduit_end_ab_input2                 : in    std_logic  :=
     'X';             -- ab_input2
```

The instance of the Qsys system defines which pins on the DE2's GPIO header are connected to the pins on the Genesis controllers. The configuration used for our interface PCB is shown in Listing 3.

Listing 3: Port Map Configuration

```
video_vga_controller_0_external_interface_VS => VGA_VS,
genesis_0_conduit_end_vsync  => genesis_trigger,
genesis_0_conduit_end_dpad_up_input1            => GPIO_1(9),
genesis_0_conduit_end_dpad_down_input1          => GPIO_1(13),
genesis_0_conduit_end_dpad_left_input1          => GPIO_1(17),
genesis_0_conduit_end_dpad_right_input1         => GPIO_1(19),
genesis_0_conduit_end_select_input1             => GPIO_1(15),
genesis_0_conduit_end_start_c_input1            => GPIO_1(21),
genesis_0_conduit_end_ab_input1                 => GPIO_1(11),
```

```
genesis_0_conduit_end_dpad_up_input2            => GPIO_1(23),
genesis_0_conduit_end_dpad_down_input2          => GPIO_1(27),
genesis_0_conduit_end_dpad_left_input2          => GPIO_1(31),
genesis_0_conduit_end_dpad_right_input2         => GPIO_1(33),
genesis_0_conduit_end_select_input2             => GPIO_1(29),
genesis_0_conduit_end_start_c_input2            => GPIO_1(35),
genesis_0_conduit_end_ab_input2                 => GPIO_1(25),
```

Compile the project to ensure it is working.

4.3 Eclipse

A sample workspace is included in the companion code to read the Genesis controllers connected to the system, and output to the red LEDs on the DE2. This is located in the `software` directory of the companion code package.

5 Reading from the Controllers

The current state for each button for either controller is stored in a 32-bit value, with 16 of those bits corresponding to a button on either controller. By reading the memory at `GENESIS_0_BASE`, one can examine a particular bit to determine which button is being pressed. The code below illustrates this simply:

Listing 4: Reading Genesis Controller State

```
int genesis_value = IORD_32DIRECT(GENESIS_0_BASE, 0);

if ((genesis_value)& (1 << 0)){
  printf("1␣Up␣was␣pressed\n");
}
if ((genesis_value)& (1 << 1)){
  printf("1␣Down␣was␣pressed\n");
}
if ((genesis_value)& (1 << 2)){
  printf("1␣Left␣was␣pressed\n");
}
if ((genesis_value)& (1 << 3)){
  printf("1␣Right␣was␣pressed\n");
}
if ((genesis_value)& (1 << 4)){
  printf("1␣A␣was␣pressed\n");
}
if ((genesis_value)& (1 << 5)){
  printf("1␣B␣was␣pressed\n");
}
if ((genesis_value)& (1 << 6)){
  printf("1␣C␣was␣pressed\n");
}
if ((genesis_value)& (1 << 7)){
  printf("1␣Start␣was␣pressed\n");
}

if ((genesis_value)& (1 << 10)){
  printf("2␣Up␣was␣pressed\n");
```

```
}
if ((genesis_value)& (1 << 11)){
  printf("2␣Down␣was␣pressed\n");
}
if ((genesis_value)& (1 << 12)){
  printf("2␣Left␣was␣pressed\n");
}
if ((genesis_value)& (1 << 13)){
  printf("2␣Right␣was␣pressed\n");
}
if ((genesis_value)& (1 << 14)){
  printf("2␣A␣was␣pressed\n");
}
if ((genesis_value)& (1 << 15)){
  printf("2␣B␣was␣pressed\n");
}
if ((genesis_value)& (1 << 16)){
  printf("2␣C␣was␣pressed\n");
}
if ((genesis_value)& (1 << 17)){
  printf("2␣Start␣was␣pressed\n");
}
```

References

[1] D. Cohen, "History of the Sega Genesis." http://classicgames.about.com/od/consoleandhandheldgames/p/History-Of-The-Sega-Genesis-Dawn-Of-The-16-Bit-Era.htm. Accessed: 2016-01-23.

[2] C. Rosenberg, "Sega six button controller hardware info." https://www.cs.cmu.edu/~chuck/infopg/segasix.txt. Accessed: 2016-01-18.

[3] Toshiba Corporation, "TC74HC157AP Datasheet." http://toshiba.semicon-storage.com/info/docget.jsp?did=10768&prodName=TC74HC157AP. Accessed: 2016-01-15.