# Sega Genesis Controller Interfacing

Mason Strong, Stephen Just 2016-04-02

## 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 albeit 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, and the supplied code examples have only been verified to handle three-button controllers.

To simplify the example project, there is a simple process in the supplied VHDL top-level file to generate a pulse every 50ms to trigger the Genesis controller block, as opposed to connecting the block to a VSync signal.

# 4 System Requirements

## 4.1 Hardware

Ensure that you are in possession of a three-button genesis controller, and a corresponding adapter board to interface the DB9 connector with the appropriate GPIO pins on the DE2. Should an adapter board be unavailable, a 40-pin header can be connected to two male DB9 connectors on a breadboard using the pinout described by the schematic in Figure 2. Then, you can use a ribbon cable to connect your breadboard to your DE2.

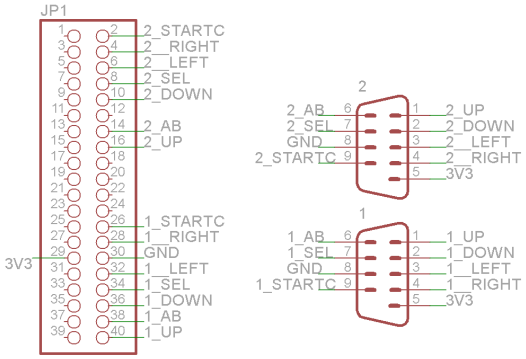


Figure 2: Connection Schematic

An Eagle project with a custom PCB design is included in `pcb_design.tar.gz`, found with this document.

## 4.2 Qsys

In order to interface with Genesis controllers, you may use the Qsys-compatible IP Core supplied with the example code in the `ip` directory. This IP core is also available standalone in the `genesis_ipcore.tar.gz` archive supplied with this document.

With the Genesis IP Core included in your project, you can connect it to your Qsys system. Figure 3 shows how the Genesis Controller Interface can be connected to your system.

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

As well, the FPGA's internal weak pull-up resistors should be enabled to ensure that the inputs are not floating when controllers are not plugged in to

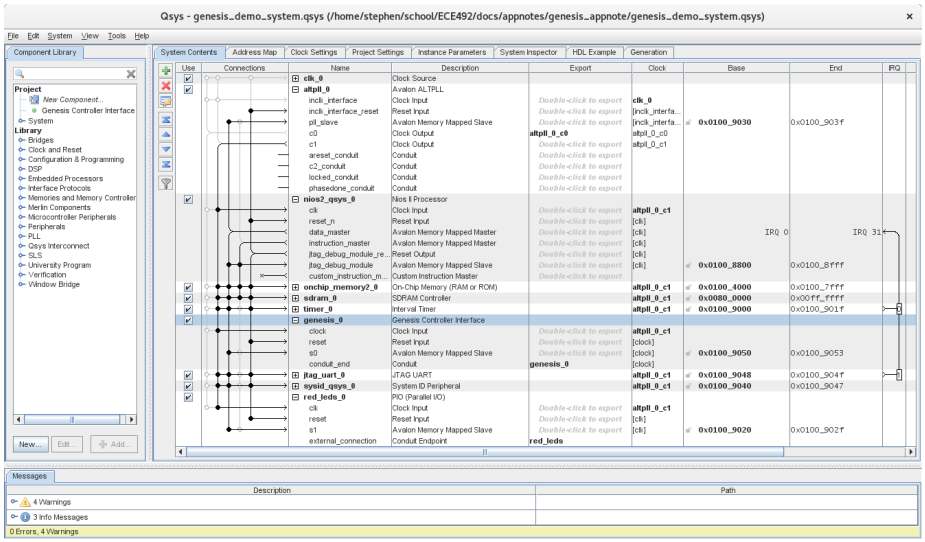


Figure 3: Qsys System Connections

the system. This can be done by adding the lines in Listing 2 to your project's QSF file.

Listing 1: Entity for Top-Level

```
entity genesis_demo is
        port
        (
                -- Clocks
                CLOCK_50     : in       std_logic;

                -- SDRAM on board
                DRAM_ADDR    : out      std_logic_vector (11 downto 0);
                DRAM_BA_0    : out      std_logic;
                DRAM_BA_1    : out      std_logic;
                DRAM_CAS_N   : out      std_logic;
                DRAM_CKE     : out      std_logic;
                DRAM_CLK     : out      std_logic;
                DRAM_CS_N    : out      std_logic;
                DRAM_DQ      : inout    std_logic_vector (15 downto 0);
                DRAM_LDQM    : out      std_logic;
                DRAM_UDQM    : out      std_logic;
                DRAM_RAS_N   : out      std_logic;
                DRAM_WE_N    : out      std_logic;

                -- Input switches and buttons
                KEY          : in       std_logic_vector (3 downto 0);

                -- Indicator LEDs
                LEDR         : out      std_logic_vector (17 downto 0);

                -- GPIO Port 1
                GPIO_1 : inout std_logic_vector (35 downto 0) := (
                    others => 'Z')
        );
end genesis_demo;
```

Listing 2: Commands to Enable GPIO Pull-Up Resistors

```
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[0]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[1]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[2]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[3]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[4]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[5]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[6]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[7]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[8]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[9]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[10]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[11]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[12]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[13]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[14]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[15]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[16]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[17]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[18]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[19]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[20]
```

```
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[21]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[22]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[23]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[24]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[25]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[26]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[27]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[28]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[29]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[30]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[31]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[32]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[33]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[34]
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to GPIO_1[35]
```

The Qsys system containing the Genesis controller interface component includes the ports similar to the ones shown in Listing 3 within its component declaration.

Listing 3: Qsys System Ports for Genesis Controller

```
genesis_0_trigger                 : in    std_logic   := 'X';
genesis_0_dpad_up_input1          : in    std_logic   := 'X';
genesis_0_dpad_down_input1        : in    std_logic   := 'X';
genesis_0_dpad_left_input1        : in    std_logic   := 'X';
genesis_0_dpad_right_input1       : in    std_logic   := 'X';
genesis_0_select_input1           : out   std_logic;
genesis_0_start_c_input1          : in    std_logic  := 'X';
genesis_0_ab_input1               : in    std_logic  := 'X';
genesis_0_dpad_up_input2          : in    std_logic  := 'X';
genesis_0_dpad_down_input2        : in    std_logic  := 'X';
genesis_0_dpad_left_input2        : in    std_logic  := 'X';
genesis_0_dpad_right_input2       : in    std_logic  := 'X';
genesis_0_select_input2           : out   std_logic;
genesis_0_start_c_input2          : in    std_logic  := 'X';
genesis_0_ab_input2               : in    std_logic  := 'X';
```

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 4.

Listing 4: Port Map Configuration

```
genesis_0_trigger                 => genesis_poll_trigger,
genesis_0_dpad_up_input1          => GPIO_1(35),
genesis_0_dpad_down_input1        => GPIO_1(31),
genesis_0_dpad_left_input1        => GPIO_1(27),
genesis_0_dpad_right_input1       => GPIO_1(25),
genesis_0_select_input1           => GPIO_1(29),
genesis_0_start_c_input1          => GPIO_1(23),
genesis_0_ab_input1               => GPIO_1(33),
genesis_0_dpad_up_input2          => GPIO_1(13),
genesis_0_dpad_down_input2        => GPIO_1(9),
genesis_0_dpad_left_input2        => GPIO_1(5),
genesis_0_dpad_right_input2       => GPIO_1(3),
genesis_0_select_input2           => GPIO_1(7),
```

```
genesis_0_start_c_input2                => GPIO_1(1),
genesis_0_ab_input2                     => GPIO_1(11),
```

### 4.4 Nios II SBT

A sample project 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. Import the project and its BSP into a new workspace.

## 5 C Driver API

A driver is provided with the Genesis IP Core. To make use of this driver, include `genesis.h` into your project. `genesis.h` is automatically added to your BSP when you include the Genesis IP Core in your system. Listing 5 provides an example of how you might read the state of two Genesis controllers in software.

Listing 5: Reading Genesis Controller Example

```
#include <stdio.h>
#include <system.h>
#include <genesis.h>

int main(void)
{
  // Initialize the Genesis controller interface
  genesis_open_dev(GENESIS_0_NAME);

  genesis_controller_t player1, player2;
  while (1)
  {
    // Poll the status of each Genesis controller
    player1 = genesis_get(GENESIS_PLAYER_1);
    player2 = genesis_get(GENESIS_PLAYER_2);

    // Check which buttons are pressed on controller 1
    if (player1.up){
      printf("1␣Up␣was␣pressed\n");
    }
    if (player1.down){
      printf("1␣Down␣was␣pressed\n");
    }
    if (player1.left){
      printf("1␣Left␣was␣pressed\n");
    }
    if (player1.right){
      printf("1␣Right␣was␣pressed\n");
    }
    if (player1.a){
      printf("1␣A␣was␣pressed\n");
    }
```

```
    if (player1.b){
      printf("1 B was pressed\n");
    }
    if (player1.c){
      printf("1 C was pressed\n");
    }
    if (player1.start){
      printf("1 Start was pressed\n");
    }

    // Check which buttons are pressed on controller 2
    if (player2.up){
      printf("2 Up was pressed\n");
    }
    if (player2.down){
      printf("2 Down was pressed\n");
    }
    if (player2.left){
      printf("2 Left was pressed\n");
    }
    if (player2.right){
      printf("2 Right was pressed\n");
    }
    if (player2.a){
      printf("2 A was pressed\n");
    }
    if (player2.b){
      printf("2 B was pressed\n");
    }
    if (player2.c){
      printf("2 C was pressed\n");
    }
    if (player2.start){
      printf("2 Start was pressed\n");
    }
    }

  return 0;
}
```

# 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.