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



Figure 1: Genesis Controller

controllers also use a standard DB-9 connector, unlike most other game consoles which use proprietary connectors. [2]

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 $(+5 \text{ volts})$	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 sixbutton 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

	Qsy	rs - ge	enesi	s_demo_syst	tem.qsys (/ho	ome/stephe	n/school/ECE49	2/docs/app	notes/ger	nesis_appnote	/genesis_d	emo_system.qsy	s)			×
<u>E</u> lle <u>E</u> dit <u>S</u> ystem <u>∨</u> lew <u>T</u> ools <u>H</u> elp																
Component Library	Syst	em Cont	tents	Address Map	Clock Settings	Project Settin	igs 👔 Instance Param	eters Syste	m Inspector	HDL Example	Generation					
	-	Use		Connections	Nan	18	Descript	ion		Export	Clock	Base		End		IRQ
		×			⊡ clk_0	0	Clock Source									
Project	-	V			⊟ altpll_0	1	Avalon ALTPLL									
- New Component			<u> </u>		inclk_inter	face 0	Clock Input		Double	e-click to export	clk_0					
 Genesis Controller Interface 				• •	inclk_inter	face_reset F	Reset Input		Double	e-click to export	[inclk_interfa					
o∽ System				+ • · · · ·	pl_slave	4	Avalon Memory Mappe	d Slave	Double	e-click to export	[inclk_interfa			0x0100_903f		
Library Pridane					< c0	0	Clock Output		altpll_0_c	:0	altplI_0_c0					
Clock and Reset			Πč		< c1	0	Clock Output		Double	e-click to export	altplI_0_c1					
 Configuration & Programming 				-	areset_co	nduit G	Conduit		Double	e-click to export						
• DSP				-	c2_condui	t c	Conduit		Double	e-click to export						
 Embedded Processors 				-	locked_co	nduit G	Conduit		Double	e-click to export						
• Interface Protocols	191				phasedon	e_conduit 0	Conduit		Double	e-click to export						
Memories and Memory Controller		×			nios2_qsy	s_0 P	lios Il Processor									
 Merin Components Moreopartrolar Deviatorale 			°-+	+ + +	> clk	0	Clock Input				altpll_0_c1					
Peripherals					reset_n	F	Reset Input				[clk]					
o∽ PLL					< data_mast	er /	Avalon Memory Mappe	d Master			[clk]		IRQ O		IRQ 31	
 Qsys Interconnect 					< instruction	_master /	Avalon Memory Mappe	d Master			[clk]					
SLS					< jtag_debuj	g_module_reF	Reset Output				[clk]					
 University Program 				++	itag_debui	2_module /	Avalon Memory Mappe	d Slave			[clk]	© 0x0100_8800		0x0100_8fff		
 Verification Mandaux Bridge 				×	< custom_in	struction_m 0	Custom Instruction Mas	ter	Double	e-click to export						
- vwridow bridge		r	° +	*** :	→	emory2_0	On-Chip Memory (RAM	or ROM)			altpll_0_c1	r 0x0100_4000		0×0100_7fff		
		r	0-+	***	→ 🗄 sdram_0	9	SDRAM Controller				altpll_0_c1	© 0x0080_0000		0×00ff_ffff		
		V		† ° † :	→ timer_0	- P	nterval Timer				altpli_0_c1	r 0x0100_9000		0×0100_901f	P	-9
		V			🖃 genesis_0	(Genesis Controller Inter	face								
			1 T		CIOCK		Jock Input				aftpil_0_c1					
					reset		reset input				[CIOC8]					
				T I I I I I I I I I I I I I I I I I I I	* SU		Avalon Memory Mappe	a Stave			[CIOCK]	■ 0x0100_9050		0x0100_9053		
					conduit_er	na c	conduit		genesis_	0	[CIOC8]					4
			î T	III :	• 🖽 jtag_uart_0		ITAG UART		_		aftpll_0_c1	0x0100_9048		0x0100_904T	2	-0
			0	TYT	• E sysid_qsy	8_0	system ID Peripheral				aftpil_0_c1	# 0x0100_9040		0x0100_9047		
			11		⊢ red_leds_	,	NO (Parallel I/O)									
			0		CIK		Jock Input		Double	e-click to export	aftpil_0_c1					
					reset		reset input		Double	e-click to export	[CIK]					
				• •	* \$1		Avalon Memory Mappe	3 Słave	Doubh	e-crick to export	[CIK]	# 0x0100_9020		0X0100_902T		
New Edit 🕂 Add					external_c	connection p	conduit Endpoint		red_leds	1						
Messages																
				Descriptio	n							Path				
🗢 <u> </u> 4 Warnings																
🗢 📵 3 info Messages																
0 Errors, 4 Warnings																

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			
(
C	locks		
CLOC	:K_50 :	in	std_logic;
S	DRAM on boa	ard	
DRAM	i_ADDR :	out	<pre>std_logic_vector (11 downto 0);</pre>
DRAM	i_BA_0 :	out	std_logic;
DRAM	1_BA_1 :	out	<pre>std_logic;</pre>
DRAM	i_CAS_N :	out	<pre>std_logic;</pre>
DRAM	i_CKE :	out	<pre>std_logic;</pre>
DRAM	i_Clk :	out	std_logic;
DRAM	i_CS_N :	out	std_logic;
DRAM	i_DQ :	inout	<pre>std_logic_vector (15 downto 0);</pre>
DRAM	i_ldqm :	out	<pre>std_logic;</pre>
DRAM	i_udqm :	out	<pre>std_logic;</pre>
DRAM	i_RAS_N :	out	<pre>std_logic;</pre>
DRAM	1_WE_N :	out	std_logic;
I	nput switch	nes and bu	ittons
KEY		: in	<pre>std_logic_vector (3 downto 0);</pre>
I	ndicator LI	EDs	
LEDF	:	out	<pre>std_logic_vector (17 downto 0);</pre>
@	PIO Port 1		
GPIC	_1 : inout	std_logid	c_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]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[9]
set_instance_assignment	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[10]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[11]
set_instance_assignment	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[12]
<pre>set_instance_assignment</pre>	-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]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[17]
<pre>set_instance_assignment</pre>	-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	GPTO 1[20]

<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[21]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[22]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[23]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[24]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[25]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[26]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[27]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[28]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[29]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[30]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[31]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[32]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[33]
<pre>set_instance_assignment</pre>	-name	WEAK_PULL_UP_RESISTOR	ON	-to	GPIO_1[34]
<pre>set_instance_assignment</pre>	-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	<pre>std_logic;</pre>	
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	<pre>std_logic;</pre>	
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

- 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.