Seven Segment Decoder v1.0

IP Documentation

November 2022

Licensing Notice

Seven Segment Decoder v1.0 is a fully tested, portable, configurable, and synthesisable soft IP core provided by **Chip**munk **Logic**™. The IP source files are complied with IEEE VHDL/Verilog/System-Verilog standards. All the source codes are open-source licensed and hence may be used, modified, and shared without any restrictions or conflicts of interest with the original developer.

Chipmunk **Logic**™ shall not be liable or held accountable for any loss or damage (direct or indirect) resulting from the use of any product, as the designs are not intended to be fail-safe or for use in any application requiring fail-safe performance. Hence, the user shall assume sole risk and liability for the use of any of our products in any of their applications.

Table of Contents

1. Seven Segment Decoder 4
2. Features 4
3. Overview 5
4. SPI Interface 6
5. Configuration Parameters 7
6. Top-level Ports/Interfaces 8
7. Register Map 9
8. Designing with the IP 10
   8.1 Clocking 10
   8.2 Reset 10
   8.3 Configuring the IP 10
   8.4 Refresh Rate Control 10
   8.5 Motion Rate Control 11
   8.6 Interrupt 12
9. Appendix 13
   9.1 Test Application 13
   9.2 Performance and Resource Utilization on FPGA 13

1. Seven Segment Decoder

Seven Segment Decoder is a configurable soft IP which can decode and drive segment display panels.

2. Features

- ✓ Configurable display panel size, max. of 8 digits supported.
- ✓ Run-time configurability through SPI interface.
- ✓ Configurable data and enable pin polarities to support both common anode and common cathode displays.
- ✓ Motion enable/disable feature – horizontal scrolling effect.
- ✓ Supports wide range of core clock (1-100 MHz) at configurable refresh rates > 60 Hz.

3. Overview

Fig 3.1 shows the top-level block diagram of Seven Segment Decoder.

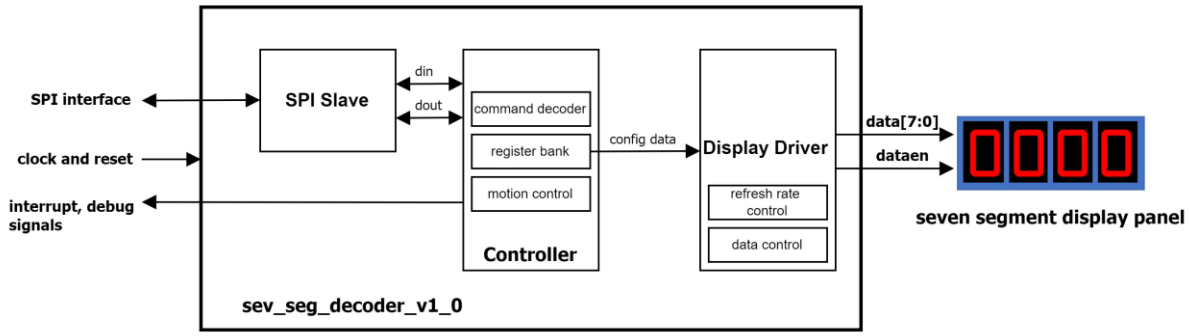


Fig 3.1: Seven Segment Decoder – Block Diagram

The core has set of peripheral registers to control the display. These registers are programmable through commands via SPI bus. *SPI Slave* converts the serial data to parallel data (and vice versa) for *Controller*.

Controller is responsible for decoding the commands, configuring the register bank, and dynamically control the data flow for motion control.

The configuration data from *Controller* is read by *Display Driver* aka *Decoder*. *Decoder* controls the refresh rate of displays by dynamically shifting and controlling the enable (common anode/cathode) pin and muxing the data to display panel at appropriate time intervals.

Interrupt and debug signals are also generated by the core.

4. SPI Interface

The core registers (refer here) are programmed via SPI. The SPI Interface supports the operational mode: [CPOL = 0, CPHA = 1]. The idle state of $\overline{SCLK}$ is low. Data is shifted out at rising edge, and sampled at falling edge of $\overline{SCLK}$. MSb is always the first bit in the byte transaction. Timing at the SPI Interface is shown below.

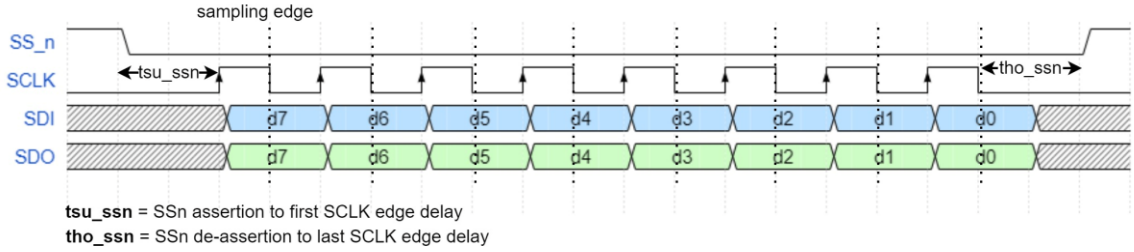


Figure 4.1: SPI Interface Timing

Pulling $\overline{SS}$ low selects the slave and enables shifting in and out of serial data through SDI and SDO. Pulling $\overline{SS}$ high will reset the serial interface and internal buffers, and hence it is expected to be asserted stable throughout the byte transfer for reliability. The timing delays `tsu_ssn` and `tho_ssn` are required to be at least 1 SCLK cycle. The SPI slave requires no interbyte transmission delay.

$\overline{SDO}$ is tri-stated to support multi-slave systems. Hence, pull-up is required on $\overline{SDO}$. No other lines require pull-up.

SCLK considerations are discussed here.

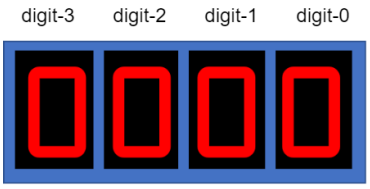
5. Configuration Parameters

The core supports following compile-time configuration parameters.

Parameter	Type	Default	Description
N	integer	**4**	Max. no of digits in seven segment display panel (xN). Range: [1, 8], recommended: {1, 2, 4, 8}

Table 5.1: Configuration Parameters

For e.g., core when configured for N = 4 supports the following display panel:



Display panel with N = 4

6. Top-level Ports/Interfaces

Table 6.1 lists all top-level I/O ports/interfaces of Seven Segment Decoder.

Signal Name	Direction	Width	Description
Clock and Reset Interface			
clk	input	1	Core Clock
resetn	input	1	Core Reset; fully synchronous, active-low
Debug Interface			
o_err_sts	output	1	Error in last command received; invalid opcode.
o_wait_sts	output	1	Waiting for data byte during write command
SPI Slave Interface			
i_sclk	input	1	Serial Clock
i_sdi	input	1	Serial Data In
o_sdo	output	1	Serial Data Out
i_ssn	input	1	Slave Select; active-low
Interrupt Interface			
o_intr	output	2	Interrupt status; for more info refer here. *bit [0]* – interrupt *bit [1]* – interrupt source; '0' if front buffer interrupt '1' if back buffer interrupt
Display Interface			
o_data	output	8	Data pins: {a, b, c, d, e, f, g, dp}
o_en	output	*N*	Enable pins; common anode/cathode pins

Table 6.1: Top-level I/O Ports/Interfaces

7. Register Map

Table 7.1 lists all peripheral registers in the core with address map. These registers are written/read via SPI. All registers are 8-bit and have reset value= 0x00 unless mentioned otherwise.

Address	Register Name	Access	Description
0x00	*control*	RW	Control register *bit [0]* – Display panel enable pin polarity *bit [1]* – Display panel data pin polarity *bit [3]* – Enable motion
0x01	*refresh_rate*	RW	Refresh rate of the display; for more info refer here.
0x02	*motion_rate*	RW	Motion rate on the display; for more info refer here.
0x03	*data_enable*	RW	Enable/Disable each digit in the display panel. If disabled, the digit will not be displayed.
0x04 to 0x<*N*-1+4>	*data_bbuf_0* to *data_bbuf_<N-1>*	RW	Back buffer of size *N* *data_bbuf_0* → digit-0 *data_bbuf_0* → digit-<*N*-1> If motion is enabled, this buffer should be loaded the next set of data to be displayed on the panel when interrupt is generated with source = '1'. If interrupt is disabled, this buffer should be loaded the second set of data to be displayed on the panel. If motion is disabled, this buffer is never used. These registers have reset value = 0x02
0x<*N*+4> to 0x<2*N*+3>	*data_fbuf_0* to *data_fbuf_<N-1>*	RW	Front buffer of size *N* *data_fbuf_0* → digit-0 *data_fbuf_0* → digit-<*N*-1> If motion is enabled, this buffer should be loaded the next set of data to be displayed on the panel when interrupt is generated with source = '0'. If interrupt is disabled, this buffer should be loaded the first set of data to be displayed on the panel. If motion is disabled, this buffer should be used to load the set of data to be statically displayed on the panel. These registers have reset value = 0x02

Table 7.1: Register Map

8. Designing with the IP

This chapter discusses the guidelines, constraints, and limitations while using Seven Segment Decoder.

8.1 Clocking

The IP has core clock, `clk`. The system clock should be at least four times faster than that of SCLK. The choice of system clock frequency also affects the range of motion and refresh rates achievable at display panel (refer below). The clock range supported is 1-100 MHz for no observable flickering on displays.

8.2 Reset

The external reset should be fully synchronous to clock. Min. pulse width of reset = 1 core clock cycle.

8.3 Configuring the IP

The core is configurable at run-time via SPI interface. The list of configurable registers is here. Writing and reading registers are performed by sending command bytes via SPI. Following are the commands supported by the core.

Command	Format	Description
`Write`	<3'b101+address><write-data>	2-byte command to write to a register <address> – 5-bit address of register <data> – Byte to be written to the register Response: NA
`Read`	<3'b100+address><0x00><0x00>	3-byte command to read a register <address> – 5-bit address of register <0x00> – Dummy byte Response: <read-data>

Table 8.1: Commands

The response to `Read` command i.e., read-data is received via SDO.

8.4 Refresh Rate Control

The refresh rate of the display can be controlled by configuring `refresh_rate` register. The expression to calculate the value r to be configured is:

$$r = \frac{core\ clock\ frequency}{required\ refresh\ rate\ in\ Hz * 8192}$$

To avoid flickering, the refresh rate should be > 60 Hz. To not significantly reduce the brightness of the display, refresh rate should be < 1 kHz.

Range of refresh rates supported for the range of core clock (1-100 MHz) is shown in below table.

Core clock (in MHz)	refresh_rate range	refresh_rate configured	Refresh rate on display (in Hz)	Refresh rate range (in Hz)
1	[1-2]	2	61	[61-122]
		1	122	
10	[2-20]	20	61	[61-610]
		2	610	
50	[7-101]	101	60	[60-872]
		7	872	
100	[13-203]	203	60	[60-939]
		13	939	

Table 8.2: Refresh Rates supported

8.5 Motion Rate Control

The core supports motion enable/disable. The front/back buffer data horizontally scrolls in and out on the digits of the panel when motion is enabled. If motion is disabled, front buffer data is statically displayed.

The motion rate of the display can be controlled by configuring `motion_rate` register. The expression to calculate the value m to be configured is:

$$r = \frac{core\ clock\ frequency}{required\ motion\ rate\ in\ Hz * 262144}$$

Range of motion rates supported for the range of core clock (1-100 MHz) is shown in below table.

Core clock (in MHz)	motion_rate range	motion_rate configured	Motion rate on display (in Hz)	Motion rate (in ms or sec)
1	[1-255]	255	0.015	66 sec
		1	3.8	263 ms
10	[1-255]	255	0.15	6.6 sec
		1	38.14	26.3 ms
50	[1-255]	255	0.748	1.34 sec
		1	190.73	5.24 ms
100	[1-255]	255	1.496	670 ms
		1	381.46	2.62 ms

Table 8.3: Motion Rates supported

1 Hz means every 1 second, a digit scrolls.

8.6 Interrupt

The core supports interrupt only if motion is enabled. The interrupt can be enabled or disabled via *control* register. The interrupt is generated when all digits in either front or back buffer have been displayed completely and scrolled over and out on motion. This is an active-high level interrupt.

If front buffer has scrolled over and out, front buffer interrupt is generated with interrupt status 2'b01. This interrupt should be acknowledged by writing new set of data to front buffer registers. If not acknowledged, same data scrolls back in next time to display. If the buffer data need not be updated, interrupt can be acknowledged by simply sending a dummy byte 0x00 via SPI.

If back buffer has scrolled over and out, back buffer interrupt is generated with interrupt status 2'b11. This interrupt should be acknowledged by writing new set of data to back buffer registers. If not acknowledged, same data scrolls back in next time to display. If the buffer data need not be updated, interrupt can be acknowledged by simply sending a dummy byte 0x00 via SPI.

9. Appendix

9.1 Test Application

The IP package comes with a test application in embedded C to test ***Seven Segment Decoder v1.0*** on board. The test environment consists of a micro-controller configuring the IP on FPGA through SPI and then enables motion and interrupt in the core. The core drives a 4-digit seven segment display.

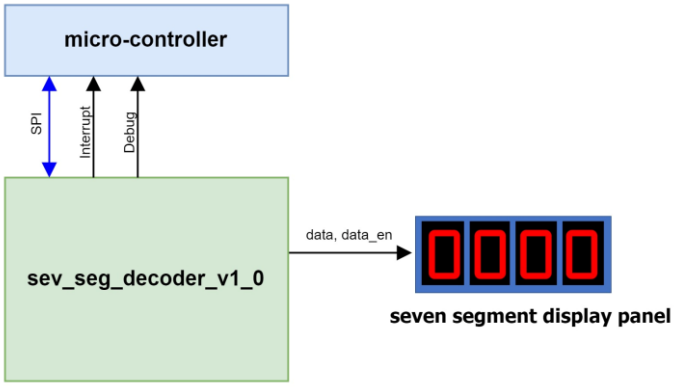


Figure 9.1: Seven Segment Decoder – Test Setup

The micro-controller updates front/back buffers in real-time on receiving interrupt and seamlessly displays the moving string configured in the application.

The test application is configurable. The default test setup is:

Micro-controller	ATmega-128; Arduino UNO
FPGA	Xilinx Artix-7; Basys-3
Core Clock	100 MHz
SPI Clock	1 MHz
Display panel	x4, data and enable polarities = '0'
Features enabled	Motion, Interrupt
String displayed	hEY thErE…this is JuST For Fun….

9.2 Performance and Resource Utilization on FPGA

The following is an estimate of timing and resource utilization of ***Seven Segment Decoder v1.0*** configured to support up to x4 display panel, when targeted on Xilinx Artix-7 FPGA.

IP Configuration	$N = 4$
FPGA Targeted	Xilinx Basys-3 Board (XC7A-35T-CPG236-1)

Synthesiser	Vivado 2018.3
Targeted Clock Frequency	50 MHz, SCLK = 20 MHz
LUTs	193
Registers	288

Seven Segment Decoder v1.0

An open-source licensed IP core

Developer : **Mitu Raj**
Vendor : **Chip**munk **Logic**™, *chip@chipmunklogic.com*
Website : **chipmunklogic.com**