Single-Wire Controller for Digital Temperature Sensors

Reference Design

FPGA-RD-02099-1.1

December 2019

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

1. Introduction

A single-wire interface can be used for serial protocol applications, such as I2C and SPI buses. It provides a smallfootprint communication channel between a controller and low-cost components on the board such as temperature sensors. Such communication channels usually do not require high speed. The protocol should be simple; using few resources in the controller.

As its name suggests, this bus protocol uses one wire to communicate with a master controller, usually a microprocessor. This reference design implements the Single-Wire Controller in a CPLD. The initialization procedure and all data transfers are implemented in the CPLD to offload the tasks of the microprocessor. The task of the microprocessor is reduced to reading and writing of the registers in the master controller through the WISHBONE bus. The Single-Wire Controller can be used in various applications. This design uses a single-wire interface to read/write a digital-output temperature sensor.

2. Features

- One-wire communication protocol compliant design
- Configurable command/data transfers
- Standard one-wire communication speed (15.4 Kbps with 60 us data slot)
- Configurable process clock frequency
- Provides control for most one-wire temperature sensors
- Initialization procedure and data transfer are triggered by configuring the control register
- WISHBONE compliant reference design

3. Functional Description

The Single-Wire Controller is used to control one or multiple one-wire slave components such as temperature sensors. It is a master controller located between the microprocessor and the slave devices.

The one-wire data communication protocol is composed of two procedures, an initialization procedure and a data/command transfer procedure. All data/command transfers are byte-oriented except for the Search-ROM command. This design provides a third procedure to complete the Search-ROM command. Figure 3.1. is a block diagram of the design.

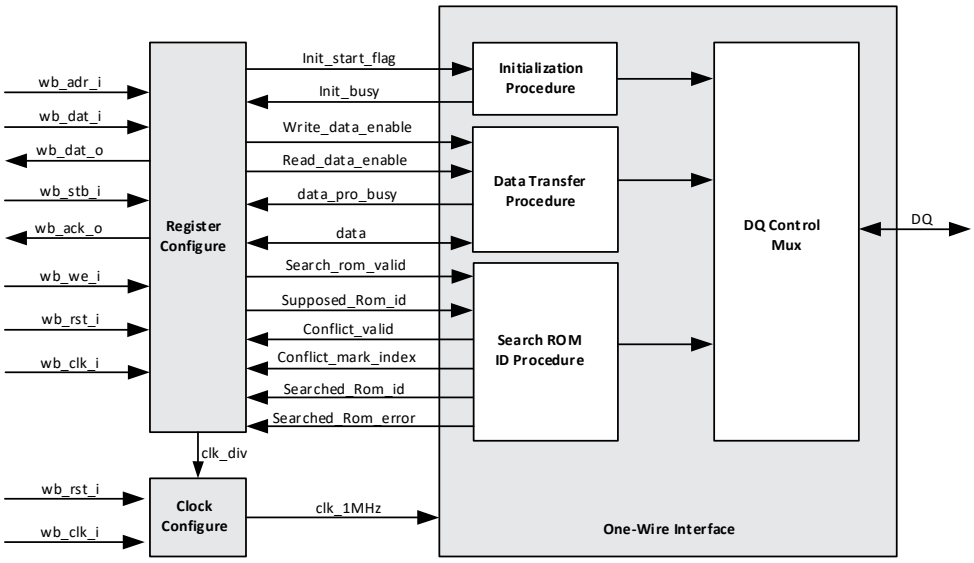

Figure 3.1. Block Diagram

Table 3.1. provides the pin descriptions for this reference design.

Table 3.1. Pin Descriptions

Port	Direction	Width	Description
WISHBONE Interface			
wb_clk_i	Input	1	WISHBONE clock input
wb_rst_i	Input	1	WISHBONE reset input, active high level, synchronous reset signal
wb_adr_i	Input	5	WISHBONE input lower address bits
wb_dat_i	Input	8	WISHBONE input data towards the core
wb_dat_o	Output	8	WISHBONE output data from the core
wb_we_i	Input	1	WISHBONE input write enable signal
wb_stb_i	Input	1	WISHBONE input strobe signal
wb_ack_o	Output	1	WISHBONE output bus cycle acknowledge output
One-wire Interface			
DQ	Input	1	One-wire device interface, default is tri-state

4. One-wire Communication Protocol

One-wire communication is a strict data transfer protocol made up of an initialization procedure, command procedure and data procedure. The initialization procedure is used to detect the presence of the slave devices and to reset them. The command procedure includes the ROM command procedure and function command procedure. The ROM command is used to identify the slave device. The function command is used to read/write the internal registers of a slave device (e.g. to trigger the temperature conversion and to get the voltage supply mode). The data procedure works with the function command to transfer or read data to and from the slave device.

When the one-wire bus is in idle state, it keeps the wire in a high logic level until the master pulls the bus to the logic low level. All the data transfers are initiated by the master start by sending the reset pulse. The initialization procedure is comprised of one reset pulse and one presence pulse. The reset pulse is transmitted by the master and the presence pulse is the acknowledge signal from the slave device. The initialization procedure timing is described in Figure 4.1.

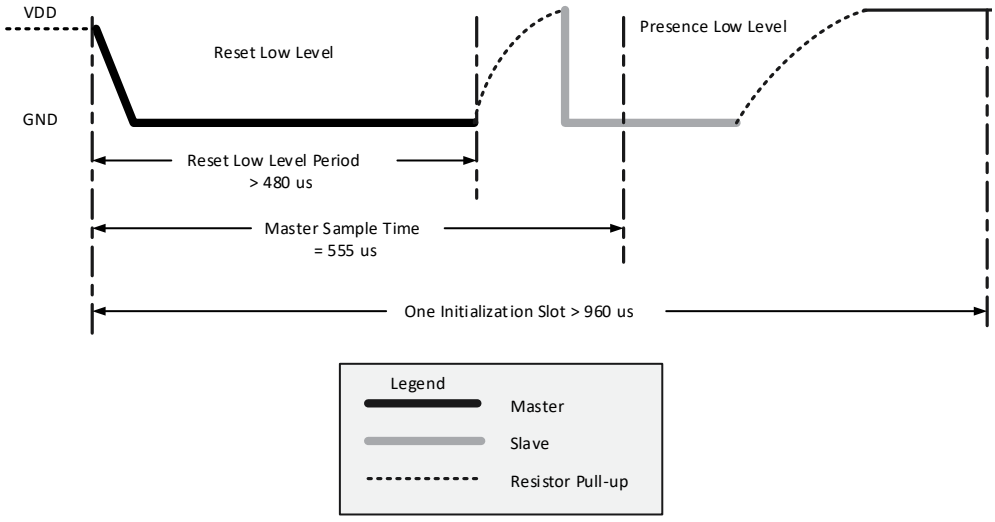

Figure 4.1. Initialization Timing Diagram

Once the master receives the presence pulse, it knows there is at least one slave device on the bus. It can now transmit a ROM command to identify the slave devices. By transmitting a ROM command, the master determines the ROM ID of the slave device and how many slave devices are available on the one-wire bus. The ROM command can be summarized by reading the ROM, MATCH ROM, SEARCH_ROM, ALARM_SEARCH and SKIP_ROM comands. The COMMAND/DATA register can be written to send the ROM command to the slave devices.

The function command is used for accessing the specified device after executing a ROM command. Depending on the functional requirements of the slave device, the master can send different command codes to the targeted slave device. The function commands include triggering temperature conversion, reading and writing the scratchpad register, and determining the power supply mode. The same COMMAND/DATA register can be used to send the function command to the slave devices.

For every data transfer operation, the master must first pull down the DQ bus for 1us. If the operation is a WRITE command/data procedure, the master can trigger the write '0' slot for bit '0', or the write '1' slot for bit '1'. If the operation is a READ procedure, the master samples the DQ bus as the read value from the slave device at 15 us. Data transfer can last from 60 to 120 us.

The timing for the write and read slots is shown in Figure 4.2.

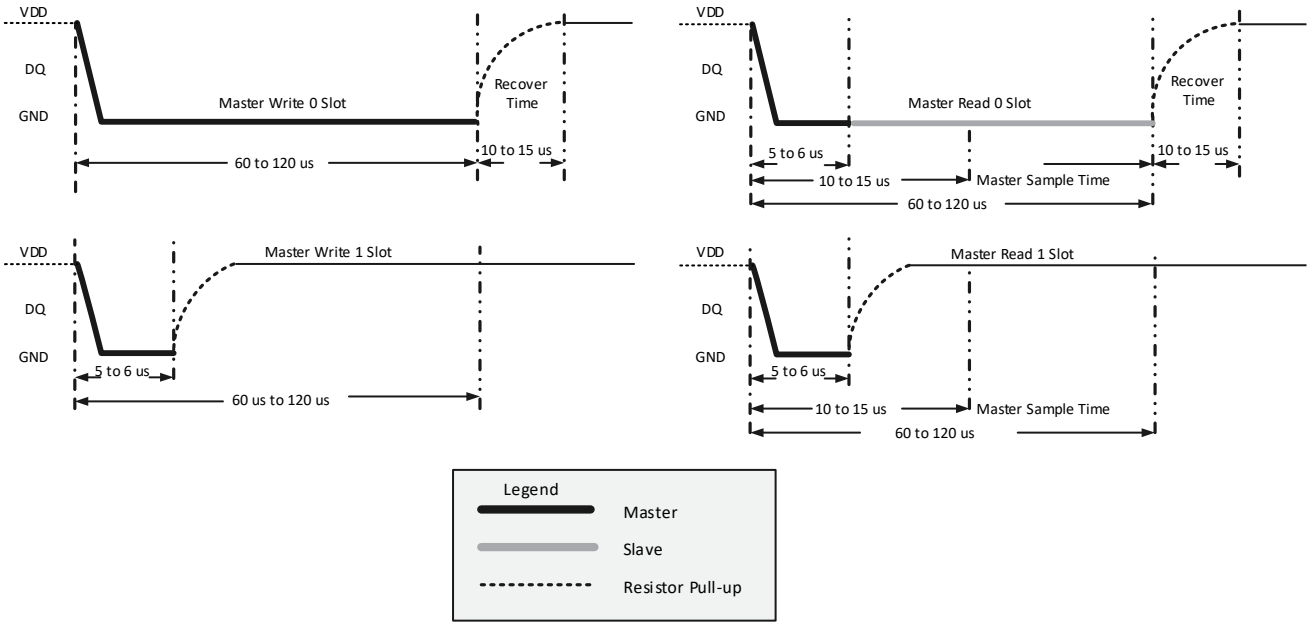

Figure 4.2. Write and Read Slot Timing

5. Design Module Description

This design can be divided into three sub-modules: the clock_divider, register_configure and one_wire interface modules.

5.1. Clock_divider Module

This module is used to generate one 1MHz clock. The CLK_DIV register can be configured to obtain the required clock frequency based on the process clock frequency. This reference design uses clock_1MHz to generate the timing parameters defined in the one-wire communication protocol.

5.2. Register_configure Module

This module implements the slave WISHBONE interface and read/write functions of all user registers. Based on the contents of the CONTROL register, the module generates different control signals to trigger different flows. The STATUS register stores the internal status of the controller. The COMMAND/DATA register is used to store the command codes and data. The READ_DATA register stores the contents read back from the slave device. For the search ROM_ID command, this reference design has a presumed Rom_ID number which is stored at addresses 8'h2 to 8'h9. In each search process, it records the conflict bit position and the conflict may occur many times. The conflict_mark register records the latest conflict position. After each search, the master acquires one slave device's ROM ID which is registered in the ROM_ID registers.

This reference design includes 22 user registers, as listed in Table 5.1.

Table 5.1. User Register Definitions

Name	Address	Width	Access	Description
Control	0x00	8	Read/Write	Control register
Write_DATA	0x01	8	Read/Write	Command/data configure register
Rom_ID_DATA	0x02,0x03 0x04,0x05, 0x06,0x07 0x08,0x09	64	Read/Write	Presumed ROM ID number, the default value is 64'b0. The value can be modified before each search ROM_ID procedure.
Rom_ID	0x0A,0x0B 0x0C,0x0D, 0x0E,0x0F, 0x10,0x11	64	Read	Received slave ROM_ID after each Search ROM command
Status	0x12	8	Read	Status register, records each trigger enable signal and monitors the state machine
Read_data	0x13	8	Read	Acquired data after executing function command
Clk_div	0x14	8	Read/Write	Clock configuration register
Conflict_mark	0x15	8	Read	Conflict mark register records the last conflict bit position

5.3. Control Registers

Table 5.2. Control Register Definitions

Bit #	Access	Description
7:4	R/W	Reserved
3	R/W	search_rom_valid signal – Triggers the search ROM procedure
2	R/W	read_data_enable signal – Triggers the read data procedure
1	R/W	write_data_enable signal – Triggers the write data procedure
0	R/W	init_start_flag signal – Triggers the initialization procedure

The control register is cleared after the busy_pro signal is valid. The default value is 8'h0.

5.4. Status Registers

Table 5.3. Status Register Definitions

Bit #	Access	Description
7	R	busy_pro signal – Or'ed by init_busy, data_busy and search_rom busy
6	R	no_slave_device_flag – Indicates no slave device on the one-wire bus
5	R	error flag – Records the error in the Search ROM procedure
4	R	conflict_mark_valid – Indicates one conflict occurs in the Search ROM procedure
3	R	search_rom_valid signal – Monitors the search ROM state machine
2	R	read_data_enable signal – Monitors the read operation state machine
1	R	Write_data_enable signal – Monitors the write operation state machine
0	R	init_start_flag signal – Monitors the initialization state machine

The status register is used to monitor the controller. The default value is 8'h0.

5.5. One-wire Interface Module

Based on the one-wire data communication protocol, the data transfer procedure uses two timing models, one for the initialization procedure and the other for the data transfer procedure. These timing modes have different timing parameters. This module uses two state machines to build the corresponding timing model. A third state machine is used to complete the Search ROM command to simplify the data transfer state machine.

The three state machines include:

- init_state_machine
- data_transfer_state_machine
- search_rom_id_state_machine

5.6. init_state_machine

According to the one-wire communication protocol, the initialization procedure can be divided into seven states, as shown in the state transfer diagram in Figure 5.1.

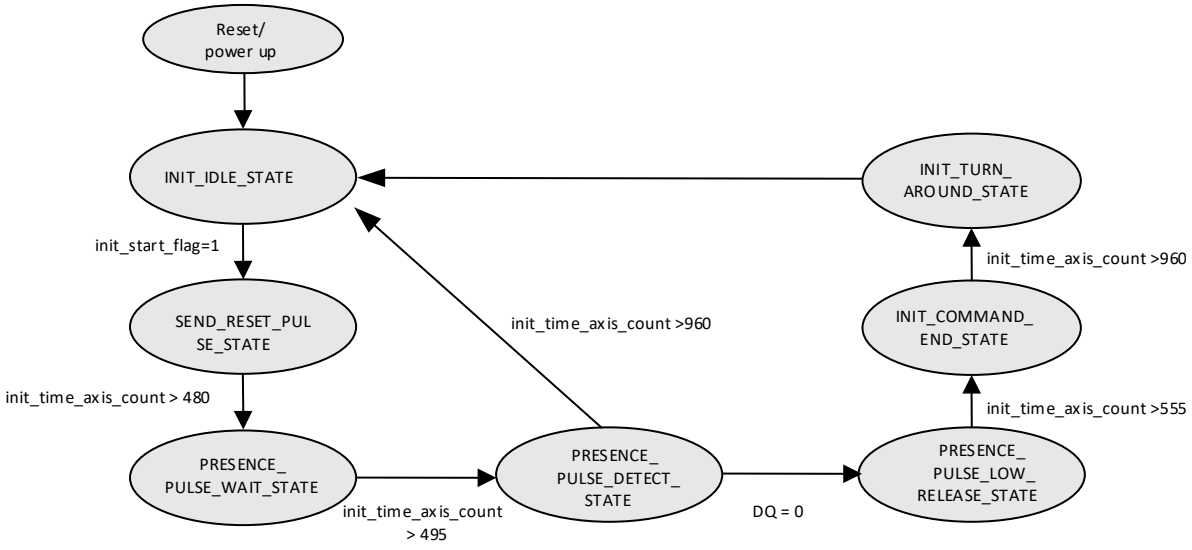

Figure 5.1. Initialization Procedure State Machine

In the initial stage, the state machine is in idle state and the DQ bus is in weak pull-up state. If the user wishes to operate a device, the first task is to write the control register to generate the init_start_flag signal. After the init_start_flag signal is received, the master pulls down the DQ bus for 480 us. All slave devices respond to the master's reset pulse by generating one presence pulse. The presence pulse is the slave device pulling down the DQ bus for more than 60 us. As soon as the master detects the low state of the DQ bus, the master knows there is a slave device response to the past reset pulse.

Sometimes the bus may not be pulled low. The master waits for 480 us and concludes that there is no slave device on the DQ bus. In this case, it generates a no_slave_device_flag signal to report this case.

5.7. data_transfer_state_machine

Data transfer is made up of the ROM command write, function command write, data write, status signal read, ROM-ID data read and Search ROM command operations. Except for the Search ROM command operation, the other command operations are byte-oriented. Since the one-wire bus uses only one line to transfer data, it needs to transfer the 8-bit data or command eight times.

Each data transfer is comprised of the write '0', write '1', read '0' and read '1' slots. This state machine is used to implement an 8-time read slot or write slot according to the master's configuration contents. If the operation is a read operation, the state machine enters the reading flow. If the operation is a write operation, it enters a write flow.

The data transfer flow can be divided into nine states as shown in Figure 5.2.

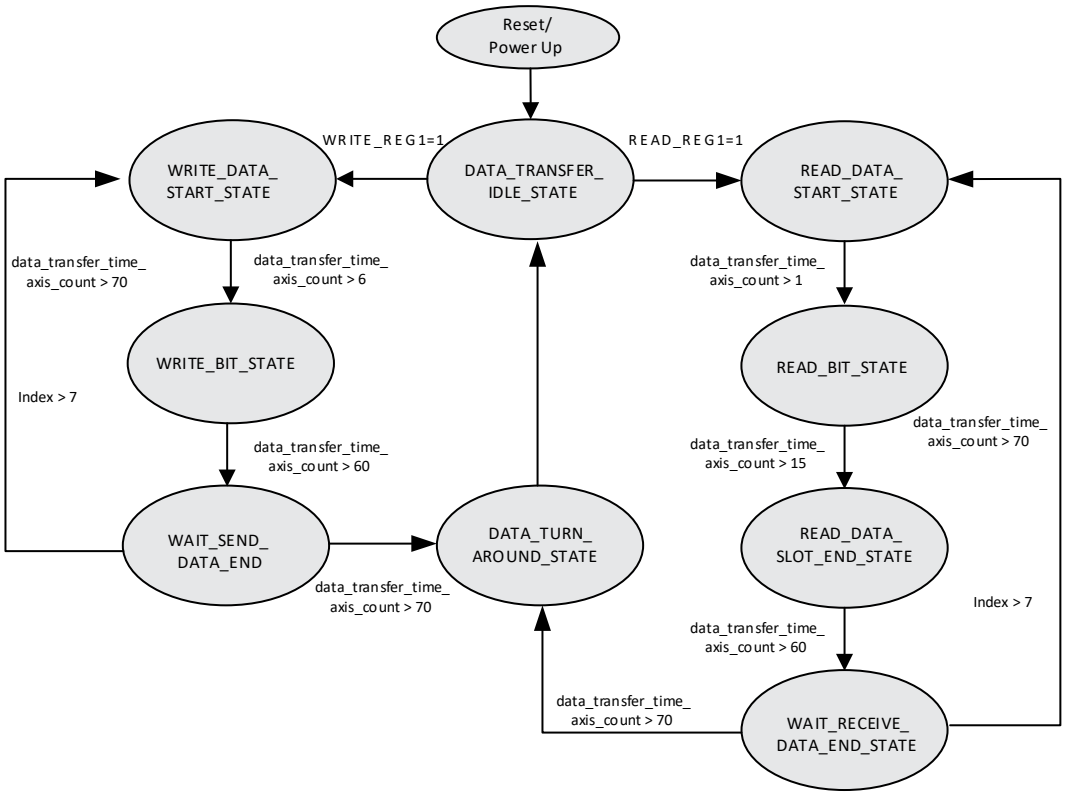

Figure 5.2. Data Transfer State Machine

If only one slave device is on the DQ bus, the master can read or write the data directly to the slave device. Usually the master does not know the number of slave devices and the ROM ID. For this reason, the master needs to send a SEARCH_ROM command to get information from the slave device.

For each data transfer, the master writes the command code to the COMMAND/DATA register first. Then it writes the write_data_enable bit of the control register. Once the write enable signal is valid, the controller can transfer a bit every 70 us. After transferring the command code, the master determines the next step. If more data must be written, the master writes the control register to activate the write-enable signal again. If data must be read, the master writes the control register to activate the read-enable signal. The controller is designed to be byte-oriented. Therefore, it reads one time if the feedback data is one byte. If the feedback data is 64 bits, the master will read the data byte eight times and the read enable signal will be written eight times.

5.8. search_rom_id_state_machine

When the serial numbers of the slave devices on the one-wire bus are unknown, the master must send a SEARCH_ROM command code to acquire the information from the slave devices. Users should read the reference document to understand the algorithm. This section only describes the implementation flow of the algorithm. For every search, it can read one 64-bit ROM-ID from the slave device.

Before the SEARCH_ROM command is executed, the master goes through the following steps:

1. For the SEARCH_ROM flow, the master sends the 8'hF0 command code first.
2. Before each search, the master writes the presumed ROM ID to the ROM_ID_DATA register. The 64'b0 is the default presumed ROM ID. Once a search is finished, the master gets the last conflict bit index and one slave device's ROM ID. The conflict_valid of the status register indicates that more than one slave device is connected to the one-wire bus. To search the second slave device, the ROM_ID_DATA register is written with the latest acquired ROM ID and the changed bit in the conflict bit index. For example, if the latest acquired ROM ID is 64'h0123_4567_89ab_cdef and the conflict bit index is 6'd32, next ROM_ID_DATA register is 64'h0123_4566_89ab_cdef.
3. After writing to the ROM_ID_DATA register, the master writes the control register to trigger the Search_ROM_ID flow. Once the search_rom_valid is active, the search ROM ID state machine will be in busy state. The state machine can be refreshed to idle state when an error occurs. The busy status signal is cleared when the search is complete.

The search ROM procedure has the same timing as the data transfer procedure. The difference is that the search ROM ID algorithm reads two bits from the slave device first and then writes to the slave device one bit at a time. This operation repeats 64 times in the search procedure. The data transfer procedure is byte-oriented; therefore this operation is repeated eight times. Figure 5.3. shows the Search ROM ID state machine.

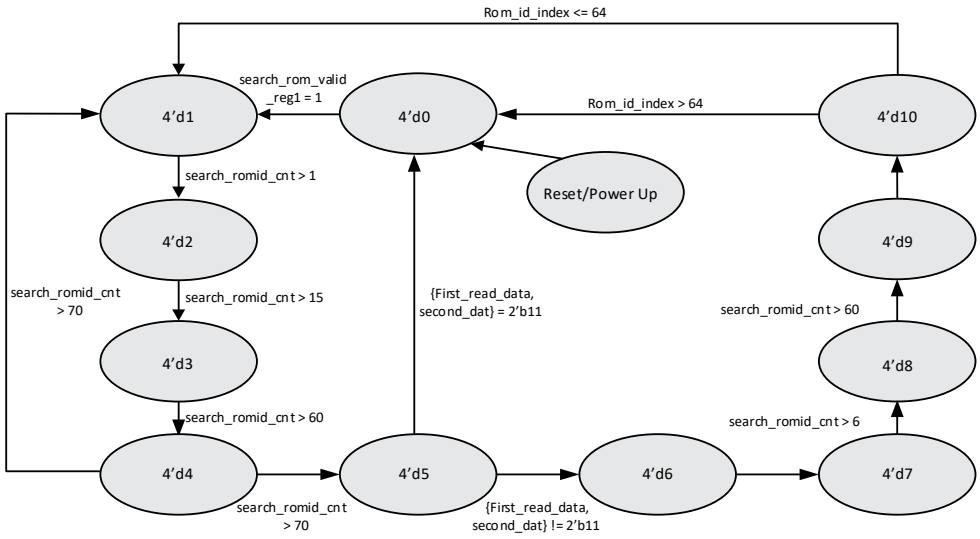

Figure 5.3. Search ROM ID State Machine

6. Operation Sequence

The sequences of common operations are described below.

To read the ROM ID:

1. Write the clock divider register at address 8'h14 with 8'hA.
2. Read the status register address 8'h12. If the controller is idle, proceed to Step 3. Otherwise, wait for the controller to be idle.
3. Write the control register at address 8'h00 with 8'h01 to trigger the initialization operation.
4. Read the status register at address 8'h12. If the controller is idle, proceed to Step 5. Otherwise, wait for the controller to be idle.
5. Write the command/data register at address 8'h01 with 8'h33.
6. Write the control register at address 8'h00 with 8'h02 to trigger write enable signal.
7. Read the status register at address 8'h12. If the controller is idle, proceed to Step 8.
8. Wait for the controller to be idle.
9. Write the control register at address 8'h00 with 8'h04 to trigger read-enable signal.
10. Read the reading data register at address 8'h13. Proceed to Step 8 until one ROM ID is read.
11. Proceed to Step 1 for the next operation.

To search the ROM ID:

1. Write the clock divider register at address 8'h14 with 8'hA.
2. Read the status register address 8'h12. If the controller is idle, proceed to Step 3. Otherwise wait for the controller to be idle.
3. Write the control register at address 8'h00 with 8'h01 to trigger the initialization operation.
4. Read the status register at address 8'h12. If the controller is idle, proceed to Step 5. Otherwise, wait for the controller to be idle.
5. Write the command/data register at address 8'h01 with 8'hF0.
6. Write the control register at address 8'h00 with 8'h02 to trigger a write enable signal.
7. Read the status register at address 8'h12. If the controller is idle, proceed to Step 8.
8. Wait for the controller to be idle.
9. Write the ROM_ID_DATA register at address 8'h2 to 8'h9 with the presumed ROM ID data.
10. Write the control register at address 8'h0 with 8'h08 to trigger the Search ROM flow.
11. Read the status register at address 8'h12. If the controller is idle, proceed to Step 12.
12. Wait for the controller to be idle.
13. Read the ROM_ID register at address 8'ha to 8'h11 to get the searched ROM ID.
14. Proceed to Step 1 for the next search operation.

7. Timing Specifications

The simulation result is based on the functional simulation of the design.

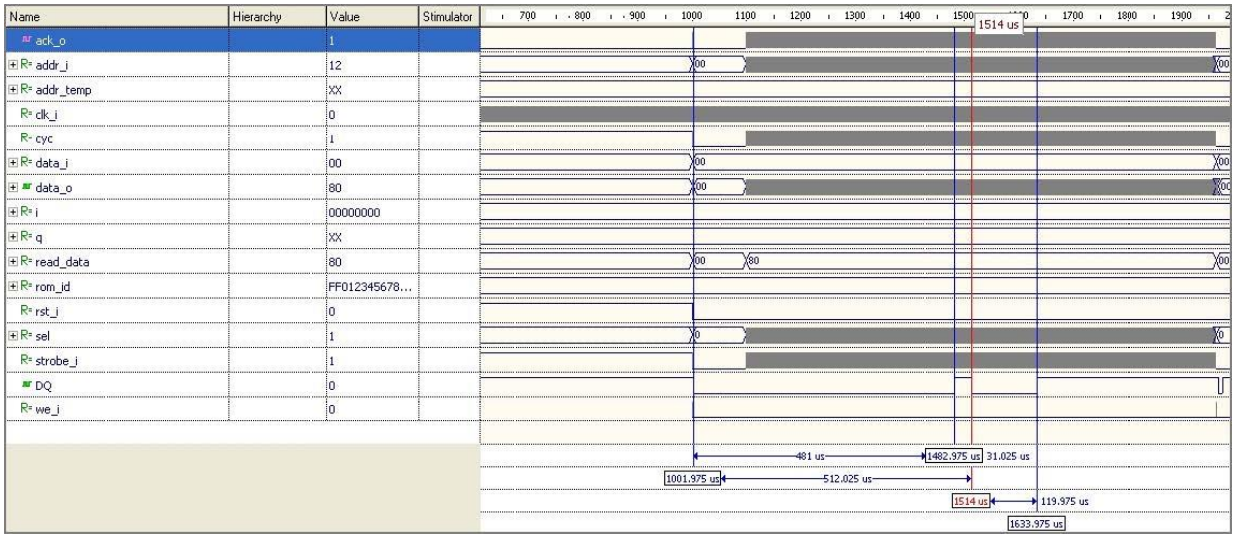

Figure 7.1. Initialization Procedures

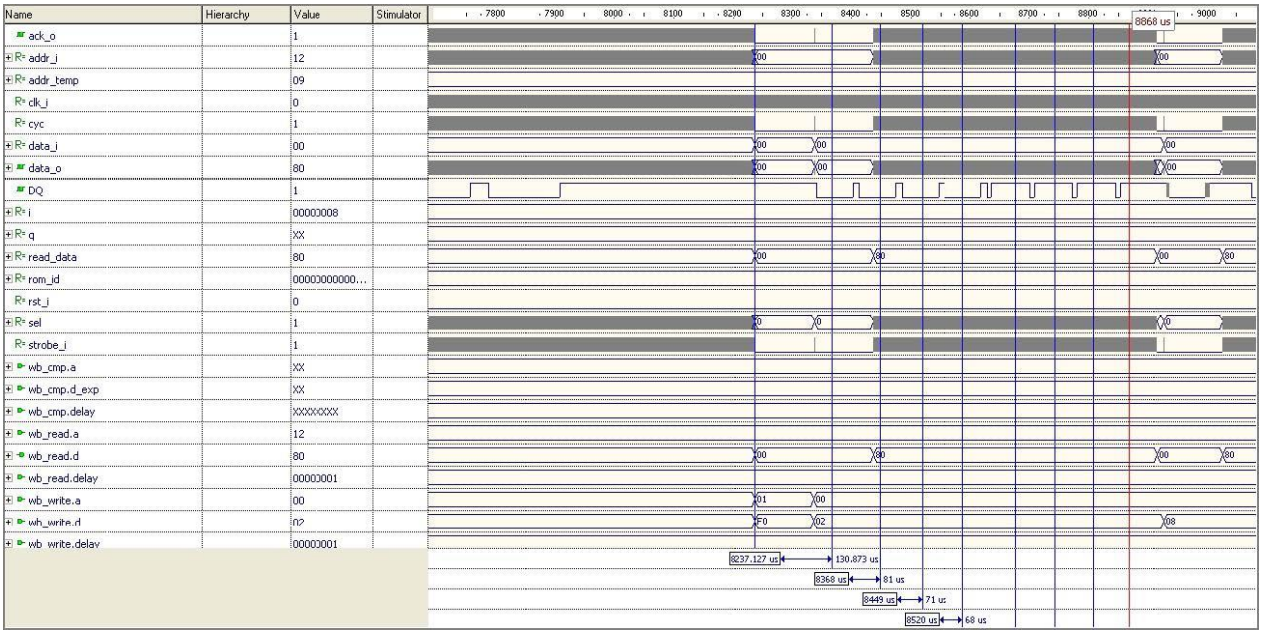

Figure 7.2. Command Transfer Waveform

1. The master writes the command 8'hF0 to the address 8'h01.
2. The master writes the control register with 8'h02 to trigger the writing operation.
3. The data_o signal specifies that the state machine in busy state.
4. The data 8'hF0 is transferred on the DQ bus.

8. Implementation

This design is implemented in the Verilog language with a VHDL wrapper supported.

Table 8.1. Performance and Resource Utilization

Device Family	Software	Speed Grade	Utilization (LUTs)	f_{MAX} (MHz)	I/Os	Architecture Resources
MachXO2™ [1]	ispLEVER®	-4	524	>20	27	N/A
	Lattice Diamond™	-4	524	>20	27	N/A
MachXO™ [2]	ispLEVER	-3	511	>20	27	N/A
	Lattice Diamond	-3	511	>20	27	N/A

Notes:

1. Performance and utilization characteristics are generated using LCMXO2-1200HC-4TG100CES, with Lattice ispLEVER 8.1 SP1 and Lattice Diamond 1.1. When using this design in a different device, density, speed, or grade, performance and utilization may vary.
2. Performance and utilization characteristics are generated using LCMXO2280C-3T100C, with Lattice ispLEVER 8.1 SP1 and Lattice Diamond 1.1 design software. When using this design in a different device, density, speed, or grade, performance and utilization may vary.

References

- Dallas Semiconductor, DS1821 – Programmable Digital Thermostat and Thermometer Data Sheet
- Dallas Semiconductor, DS18S20 – 1-wire Digital Thermometer Data Sheet
- MAXIM Semiconductor, DS1WM – Synthesizable 1-Wire Bus Master Data Sheet

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.1, December 2019

Section	Change Summary
All	• Changed document number from RD1099 to FPGA-RD-02099. • Updated document template.
Disclaimers	Added this section.

Revision 1.0, November 2010

Section	Change Summary
All	Initial release.

www.latticesemi.com