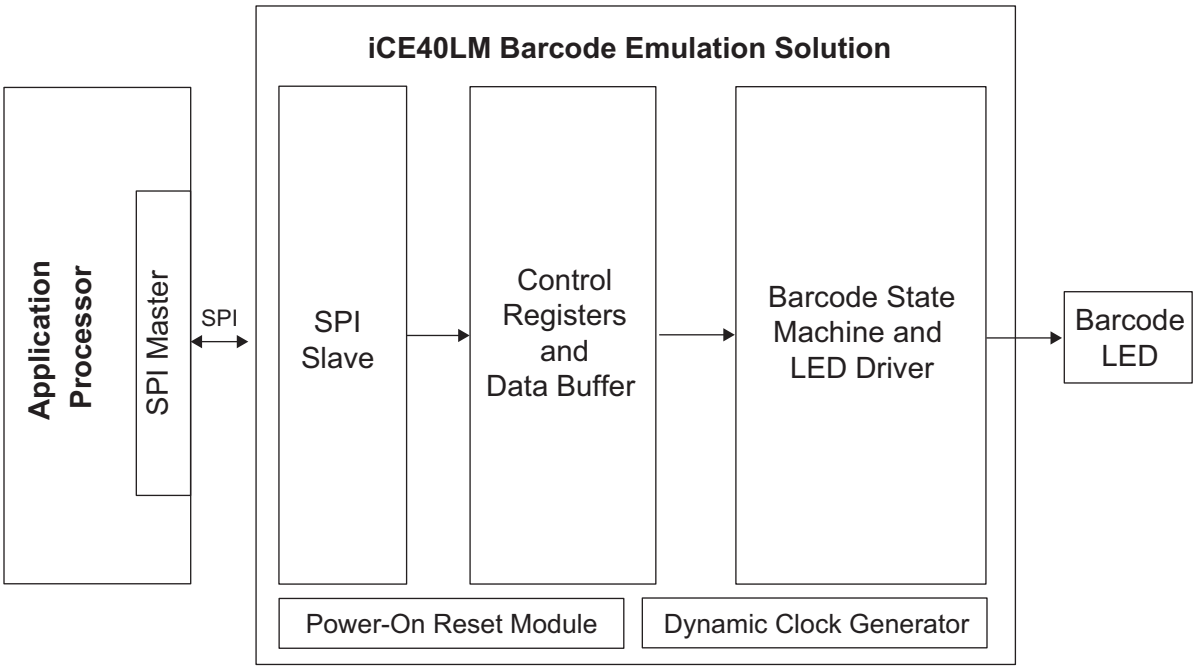
iCE40LM Barcode Emulation Reference Design

General Description

The iCE40LM Barcode Emulation Reference Design allows any device with an LED to be able to transmit bar codes to laser based bar code readers. Laser based bar code readers can not detect a bar code that is displayed on a smart phone screen for example. The laser bounces off the screen and is not readable, but with the Barcode Emulation reference design, one can use a LED to transmit the barcode so a laser can read it. This reference design is ideal for mobile devices but it can be incorporated into any product with an LED. The Barcode Emulation Reference Design allows one to easily transfer the bar code data from the application processor to the Barcode LED. This reference design acts as data control and buffer between the Barcode LED and the application processor. The Barcode Emulation Reference Design is a configurable solution, available as either standalone off the shelf or fully customizable solution.

Figure 1. System Block Diagram



As a standalone solution, the iCE40LM Barcode Emulation Reference Design connects to the application processor's Serial Peripheral Interface Bus (SPI) with clock frequency set to 10.8MHz. This enables a fast communication speed to/from the processor. The iCE40LM Barcode Emulation standalone solution prepares and sends the data from the application processor to a Barcode LED. The data is sent to the Barcode LED with the correct frequency, duration, and interval as required by Code 39 type of barcode encoding. Correct frequency is achieved by using the Dynamic Clock Generator, which sweeps the frequency at which the Barcode LED emits the signal, making the solution a barcode reader agnostic solution.

The Barcode Emulation standalone solution has a system operating frequency of 27MHz and SPI bus frequency to application processor of 10.8MHz. The SPI bus is configured to have a voltage of 1.8V, and the Barcode LED is driven by a 3.3V I/O.

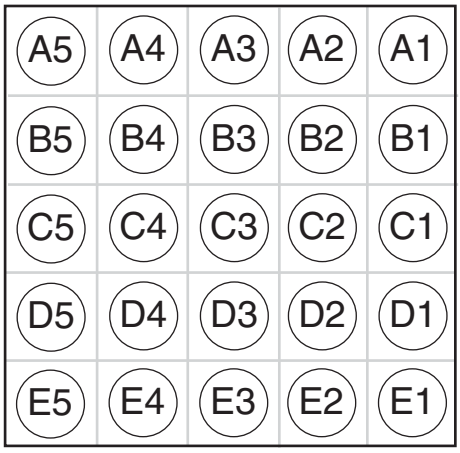
The iCE40 FPGA based architecture allows this solution to be fully customizable. This capability is ideal, but not limited to users who would like to include additional Barcode Emulation standards, change the data acquisition FIFO depth, create an I/O bridge between Barcode LEDs and processor, or create additional custom logic.

The iCE40LM Barcode Emulation reference design consumes only 356 LUTs. This allows it to fit in a device as small as a iCE40LM1K.

Regardless of the solution type, when the iCE40LM is used to implement the Barcode Emulation solution, it has a core voltage of 1.2V. It is available in a very small form-factor 25-pin WLCSP package. The package has 0.35mm ball pitch, making the overall package size to be 1.71mm x 1.71mm that easily fit into a number of mobile devices such as smart phones. Other packages include .4mm ball pitch with 36 balls (2.5x2.5mm) or 49 balls (3x3mm). The solution operates at industrial temperature range of -40C to 100C.

As a standalone solution, user simply obtains the solution which includes the device, Diamond Programmer software, and ready-for-download bitstream. As a fully customizable solution, user will obtain the device, the iCEcube2 design software, the programming software, and the source code of the Barcode Emulation solution.

Figure 2. Package Diagram (Balls Up)



Features

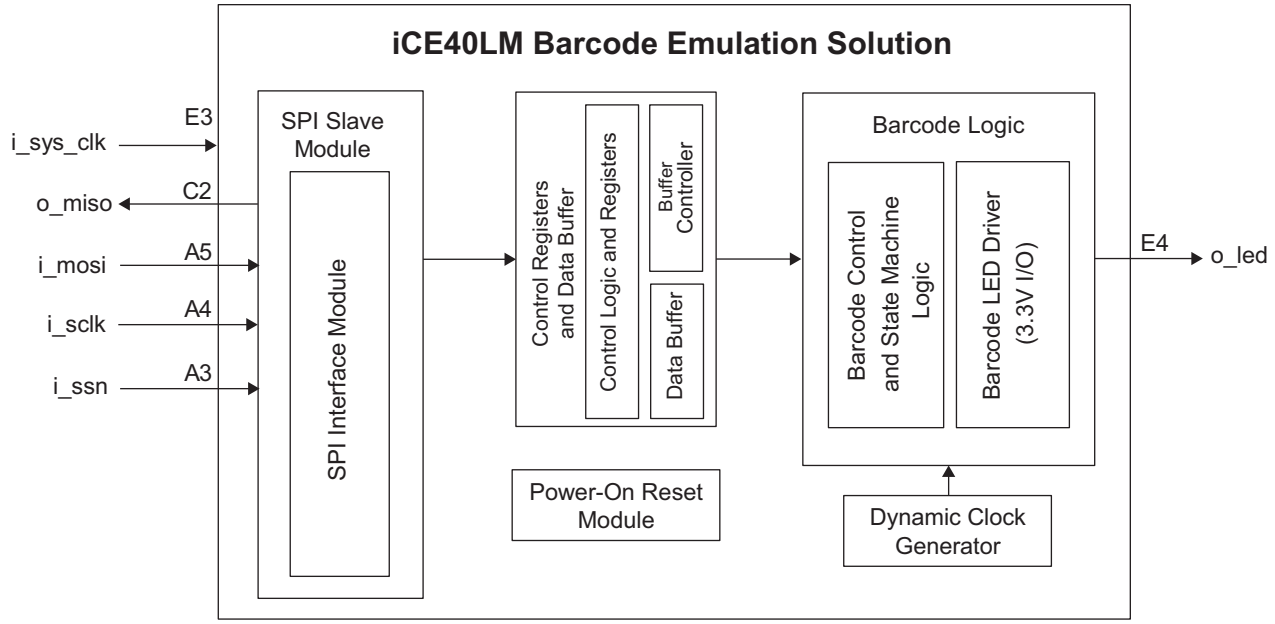
- Configurable Barcode Emulation for Mobile Devices
  - Configured to Code 39 type of barcode encoding
  - Barcode frequency sweep capability
  - Default System frequency of 27MHz
  - Power-On Reset capability
- Serial Peripheral Interface (SPI) Bus connection to Application Processor with the following Default settings:
  - Interface frequency of 10.8MHz
  - Interface voltage of 1.8V
  - Solution is a “slave” of the Application Processor
  - SPI slave mode CPOL = 1 and CPHA = 1 (mode “3”)
  - SPI slave features MSB first
- General
  - Core voltage of 1.2V
  - I/O voltages of 1.8V and 3.3V
  - 25-pin WLCS at 1.69mm x 1.69mm with 0.35mm pitch
  - Industrial (-40C to 100C) Grade

Applications

- Notebook PCs
- Smart Phones
- Tablets

Functional Block Diagram

Figure 3. Functional Block Diagram



Specifications

Recommended Operating Conditions

Table 1. Recommended Operating Conditions

Symbol	Parameter	Min.	Typ.	Max.	Units
V_{CC}	Core Supply Voltage	See DS1045, iCE40LM Family Data Sheet.	1.2	See DS1045, iCE40LM Family Data Sheet.	V
$V_{CCIOVB1}$[1]	Bank 1 I/O Driver Supply Voltage	1.71	1.8	1.89	V
$V_{CCIOVB2}$[1]	Bank 2 I/O Driver Supply Voltage	3.14	3.3	3.46	V
t_{JUND}	Junction Temperature Operation	See DS1045, iCE40LM Family Data Sheet.	-	See DS1045, iCE40LM Family Data Sheet.	°C

1. Assumes operating under “off-the-shelf standalone1 solution”.

Power Supply Ramp Rates

See DS1045, iCE40LM Family Data Sheet.

Power-On-Reset Voltage Levels

See DS1045, iCE40LM Family Data Sheet.

ESD Performance

See DS1045, iCE40LM Family Data Sheet.

DC Electrical Characteristics

See DS1045, iCE40LM Family Data Sheet. Set the $_{VCCIO}$ values to the values stated in the Recommended Operating Conditions table.

Power Supply Current

See DS1045, iCE40LM Family Data Sheet.

Absolute Maximum Ratings

See DS1045, iCE40LM Family Data Sheet.

Performance Characteristics

Table 2. Performance Characteristics[1, 3]

Symbol	Parameter	Min.	Typ.	Max.	Units
$F_{coremax}$	System Frequency			27	MHz
Tcoremaxdcd	Maximum duty cycle distortion for System Clock				%
F_{spimax}	SPI Bus Frequency			10.8	MHz
Tsuspi	SPI setup time				ns
Thdspi	SPI hold time				ns
Tcospi	SPI clock to out time				ns
Tscsn	SPI chip-select setup time				ns
Thdcsn	SPI chip-select hold time				ns
Tcooled	o_led clock to out time				ns
Tpor	Power-On Reset duration[2]		192		Cycles

1. Assumes operating under “off-the-shelf standalone1 solution”
2. Relative to System Frequency
3. All values are based on iCEcube2’s Timing Analyzer’s results. The design is not validated by test engineering.

FPGA Characteristics

See DS1045, iCE40LM Family Data Sheet. Note that once customization is performed, the values in “Performance Characteristics” may not be the same.

Pin Configuration and Function Descriptions

Figure 4. Bottom View of iCE40LM4K-SWG25TR (Balls Up)

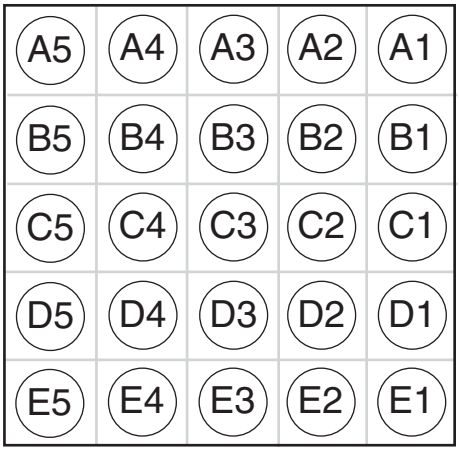


Table 3. Pin Function Description[1]

Pad Name	Port Name	Port Direction	Description
A1	General Purpose I/O	Input/Output	1.8V I/O for user interface
A2	VCCIOVB1	Input	I/O Power Supply
A3	i_ssn	Input	SPI bus slave select (Active Low)
A4	i_sclk	Input	SPI bus serial clock
A5	i_mosi	Input	SPI bus serial data in to slave
B1	General Purpose I/O	Input/Output	1.8V I/O for user interface
B2	GND	Input	Ground
B3	CRESET	Input	Configuration Reset (Active Low). See Datasheet
B4	VCC	Input	Core Power Supply
B5	General Purpose I/O	Input/Output	1.8V I/O for user interface
C1	ice_SI	Output	Configuration Output to external SPI Memory
C2	o_miso	Output	SPI bus serial data out from slave
C3	CDONE	Output	Configuration Done. See Datasheet
C4	General Purpose I/O	Input/Output	3.3V I/O for user interface
C5	General Purpose I/O	Input/Output	3.3V I/O for user interface
D1	flsh_sclk	Input	Configuration Clock
D2	ice_SO	Input	Configuration Input from external SPI Memory
D3	General Purpose I/O	Input/Output	3.3V I/O for user interface
D4	GND	Input	Ground
D5	General Purpose I/O	Input/Output	3.3V I/O for user interface
E1	flsh_cs	Input	Configuration Chip Select (Active Low)
E2	VCCIOVB2	Input	I/O Power Supply
E3	i_sys_clk	Input	System Clock
E4	o_led	Output	Barcode LED Output Driver (3.3V)
E5	General Purpose I/O	Input/Output	3.3V I/O for user interface

1. Assumes operating under “off-the-shelf standalone1 solution”.

Theory of Operations

The Barcode Emulation solution interfaces between an application processor and a Barcode LED. It receives Code 39 type of barcode encoding data and control from the processor through the SPI bus. The received data is then formatted and stored into a data buffer. When the control commands (with START asserted) are received and processed, the barcode logic reads the buffer content and converts the data into serial signals that drives the Barcode LED. The Barcode LED data rate frequency is swept through the use of the Dynamic Clock Generator so as to account for different frequencies of different barcode readers. It continues to drive the barcode LED until control commands (with STOP asserted) are received. The whole process begins again when the next set of received data and control are received.

Functional Descriptions

This sub-section describes the function of each sub-block in inside the Barcode Emulation solution. Many of these blocks have HDL module associated with them.

Barcode Emulation Top Level

The Barcode Emulation Top Level is found in barcode_fsm_top. This module contains the SPI Slave to Application Processor, the Control Registers and Data Buffer, and the Barcode Logic. It also contains a Power-On Reset (POR) module. The POR module initiates a system reset upon power up for Tpor number of cycles. The iCE40LM Barcode Emulation Solution operates after system reset has been completed. There is also the FIRMWARE_VERSION register and a counter that creates different clock frequencies as required by the Barcode Logic. The purpose of this counter is to sweep the frequency at which the Barcode LED emits signal, making the solution a barcode reader agnostic solution.

SPI Interface to Application Processor

This module is used to interface between the Barcode Emulation solution and the application processor. It is found in spi_slave module. This module waits until an interrupt is received from the processor. When a read command is received from the application processor, this module sends commands to the desired read registers. This module then receives the read data and sends them to the application processor. When a write command is received from the application processor, this module decodes the commands and sends the appropriate data to the desired register locations.

Control Registers and Data Buffer

This module stores the data for the barcode LED (from the application processor) in a buffer, and it decodes the control signals (also from processor) to initiate/stop data transmission to the barcode LED. The control signals processed are Code Delay Rate (CDR), Code Bit Rate (CBR), number of cycles, start, and stop. It also contains additional control logic for writing and reading to the data buffer. This module is found in SPI_Slave_Registers.

Barcode Logic

The Barcode Logic contains the state machine to transmit data to the barcode LED. It is controlled by CDR, CBR, number of cycles, start, and stop signals generated by the Control Registers and Data Buffer module. Upon receiving the control signals, this module reads the data buffer starting from address 0 and sends the read data serially to the barcode LED. The module also controls the duration of each serial data. This module uses the clock from the Dynamic Clock Generator so as to sweep the frequency of the Barcode LED signal emitting rate.

Block Descriptions

The purpose of this section is to provide detailed descriptions of each block of the iCE40LM Barcode Emulation Solution so as to assist users who want to use this solution using alternative sensors. Codes in this section are taken directly from the HDL file. Note that in most cases, the topics in each paragraph below are presented in the order in which they appear in the HDL code.

Barcode Emulation Top Level (barcode_fsm_top)

This module contains the SPI Slave to Application Processor, the Control Registers and Data Buffer, and the Barcode Logic. The code starts with the Power On Reset (POR) Module, which initiates a system reset upon power up for Tpor number of cycles.

The code then continues to a set of registers and a frequency counter (Dynamic Clock Generator) controlled by cycles_elapsed, which is generated by the Barcode Logic. The frequency counter is then used to generate various divided clock which is used by the Barcode Logic for sweeping the data transmission. There is also a FIRMWARE_VERSION register that can be read by the application processor.

The SPI Slave to Application Processor (spi_slave) provides communication to/from the processor. The Control Registers and Data Buffer (SPI_Slave_Registers) stores the data for the barcode LED (from the application processor) in a buffer, and it decodes the control signals (also from processor) to initiate data transmission to the barcode LED. The Barcode Logic (barcode_fsm) contains the state machine to transmit data to the barcode LED.

SPI Slave to Application Processor (spi_slave)

The SPI Slave to Application Processor Module is found in the spi_slave file. It is used to provide connection between the Barcode Emulation Solution to the application processor via SPI interface (spi_slave). This module contains the hard SPI module called “SB_SPI”. It contains logic that determines whether the command is write or read, and state machine to process the SPI master commands so as to prepare data for the backend interface.

Table 4 summarizes the ports to/from the SPI Slave to Application Processor Module

Table 4. Ports To/From the SPI Slave to Application Processor Module

Port Name	Direction	Description
i_sys_clk	Input	System Clock
i_sys_rst	Input	System Reset - Connected to POR
i_miso_byte[7:0]	Input	Data to send to Application Processor
i_miso_byte_valid	Input	Determines if data to send is valid
o_miso_byte_req	Output	Determines whether the received command is write (Active HIGH) or read (Active LOW)
o_mosi_byte[7:0]	Output	Data received from Application Processor
o_mosi_byte_valid	Output	Determines if received data is valid (Active HIGH)
o_cmd_byte	Output	Determines if received data is a command byte (Active HIGH)
o_miso	Output	SPI interface (connected to o_miso pin)
i_mosi	Input	SPI interface (connected to o_mosi pin)
i_csn	Input	SPI interface (connected to i_csn pin)
i_sclk	Input	SPI interface (connected to i_sclk pin)

Control Registers and Data Buffer (SPI_Slave_Registers)

This module stores the data for the barcode LED (from the application processor) in a buffer, and it decodes the control signals (also from processor) to initiate data transmission to the barcode LED.

Table 5 summarizes the ports to/from the Control Registers and Data Buffer Module.

Table 5. Ports To/From the Control Registers and Data Buffer Module

Port Name	Direction	Description
o_FsmRdData[7:0]	Output	Data Buffer Read Data
i_SPI_WrData[7:0]	Input	Data received from Application Processor
i_spi_busy	Input	Indicates whether SPI interface is busy (Assert HIGH) - connected to NOT of i_ssn
i_txn_busy	Input	Barcode Transmit Busy (Active High)
o_txn_start	Output	Transmit Start (Active High)
o_txn_stop	Output	Transmit Stop (Active High)
o_cbr_reg[7:0]	Output	Code Bit Rate (CBR) value
o_cdr_reg[7:0]	Output	Code Delay Rate (CDR) value
o_txn_cycles[5:0]	Output	Number of Cycles
o_DataCount[5:0]	Output	Tracks the number of data received from the Application Processor
i_sys_clk	Input	System Clock
i_sys_rst	Input	System Reset - Connected to POR
i_FsmRdEn	Input	Data Buffer Read Enable (Active High)
i_FsmRdAddr[5:0]	Input	Data Buffer Read Address
i_RegWrEn	Input	Data Buffer Write Enable (Active High) - Connected to o_mosi_byte_valid

The code starts with a set of registers to store delayed values of various control signals: i_FsmRdEn, i_txn_busy, and the i_spi_busy. The purpose of these delayed registers is to synchronize the Barcode Logic state machine to the Control Register and Data Buffer logic. Note the conditions of the data buffer read logic (FsmRdEn_pulse_i) and the SPI busy signal (d#_spi_cs_n).

Next a read_byte_count counter is implemented to keep track the number of bytes of data received from the application processor. The number of bytes of data received and the content of the data received is then used to determine whether the incoming data is read instruction (RegRdEn_i logic), and write instruction (i_RegAddr logic). Note reg_wr_data_i logic is not used.

When write instruction is received, the i_RegAddr is then checked whether it is between 0x10 and 0x4F. If it were so, then the control signal data_buf_sel_i is asserted to indicate that incoming data will be written into the data buffer. This logic is found under the following comments:

// Tx/Rx Buffer Enable, read and Write control signals from SPI

Along with the i_RegWrEn input to this module, data_bufsel_i and RegRdEn_i act as control signals for the data buffer. These signals determines whether the data buffer operation is a multi read/write (multi_rw_op_i), prepares the write data for the buffer, prepares the data buffer write address, and asserts the write enable for the data buffer. These operations are found under the following comments:

//Buffer read and write address/enable for multi-byte operation

Under the comments “// Register set”, user can see the registers used to control the Barcode Logic. These registers are control_reg, o_cbr_reg, and o_cdr_reg. The control_reg contains the Transmit Start (bit 7), Transmit Stop (bit 6), and Number of Cycles (bit 5 to 0). By default, it is set to 0x9F. On the other hand o_cbr_reg and o_cdr_reg contain CBR and CDR respectively. By default, CBR is set to 0x20, and CDR is set to 0x50. These registers are

written when i_RegAddr is set to their respective values. When these registers are set, the Barcode Logic begins the transmission process. Note that user is responsible for initiating both the start by asserting Transmit Start and the stop by asserting Transmit Stop. Do not assert both Transmit Start and Transmit Stop.

As part of the transmission process, the Barcode Logic reads the data buffer. The Barcode Logic will need to provide the data buffer read address and the data buffer read enable. The data buffer will then provide the read data and the number of data in the buffer. Note that the data buffer is a 64x8.

Table 6 summarizes the ports summarizes the ports to/from the Data Buffer.

Table 6. Ports To/From the Data Buffer

Port Name	Direction	Description
i_clk	Input	System Clock
i_rst	Input	System Reset - Connected to POR
o_RdData[7:0]	Output	Data Read from buffer
o_DataCount[5:0]	Output	Number of data in the buffer
i_FsmRdEn	Input	Buffer read enable from barcode FSM (Active High)
i_RdEn	Input	Buffer read enable (Active High) - Not used and connected to 0
i_FsmRdAddr[5:0]	Input	Buffer read address from barcode FSM
i_RdAddr[5:0]	Input	Buffer read address - Not used and connected to 0
i_WrEn	Input	Buffer write enable for data from Application Processor (Active High)
i_WrAddr[5:0]	Input	Buffer write address from Application Processor
i_WrData[5:0]	Input	Buffer write data from Application Processor

Barcode Logic (barcode_fsm)

The Barcode Logic contains the state machine to transmit data to the barcode LED. It is controlled by CDR, CBR, number of cycles, start, and stop signals generated by the Control Registers and Data Buffer module. Upon receiving the control signals, this module reads the data buffer starting from address 0 and sends the read data serially to the barcode LED. This module also controls the duration of each serial data.

Table 7 summarizes the ports to/from the Barcode Logic module

Table 7. Ports To/From the Barcode Logic Module

Port Name	Direction	Description
i_sys_clk	**Input**	System Clock
i_rst	Input	System Reset - Connected to POR
o_txn_busy	Output	Indicates if Barcode State Machine is busy (Active High)
o_buf_rd_en	Output	Data Buffer Read Enable (Active High)
o_led	Output	Barcode LED driver port
o_rd_addr[5:0]	Output	Data Buffer Read Address
o_cycles_elapsed	Output	Cycles elapsed (Active High) - used to control divided clock frequency
i_txn_start	Input	Transmit Start (Active High)
i_txn_stop	Input	Transmit Stop (Active High)
i_cdr_reg[7:0]	Input	Code Delay Rate (CDR) value
i_cbr_reg[7:0]	Input	Code Bit Rate (CBR) value
i_txn_cycles[5:0]	Input	Number of Cycles
i_byte_count[5:0]	Input	Number of data in the buffer
i_buf_rd_data[7:0]	Input	Data Buffer Read Data

The code starts with CBR register value correction and flags to indicate whether CDR and CBR registers have value greater than 0 (cdr_gt0_i and cbr_gt0_i). CDR register value is multiplied by 38 to account for 10us (logic found under "// CDR reg value multiplied by a factor(38) to account for 10us."

The code then analyze whether i_txn_start, cdr_gt0_i, and cbr_gt0_i are true, and whether i_txn_cycles is greater than 0. If all these conditions are met then the transmission process begins. As the state machine moves across states, it controls counters for cycles_count, byte_count, bit_count, cbr_count, and cdr_count. The counters control cycles_elapsed, byte_elapsed, bits_elapsed, and cdr_elapsed. Note that cycles_elapsed is used to control the divided clock frequency to be used by Barcode Logic.

Logic to indicate that the Barcode Logic is in transmitting state is implemented under "// Txn FSM busy when not in Idle State". This logic generates o_txn_busy signal.

The Barcode Logic is also responsible to generate the read enable and read address signals for the data buffer in the Control Registers and Data Buffer Module. Note that the start read address signal is 0. The logic for these functions are implemented under: // Buffer read and address generator.

The read data are then serialized and sent to the barcode LED. The logic for these functions are under the following comments:

- // Loading shift register value from buffer as well as shifting register
- // LED output also depends upon CBR register value. Other counters keep

Finally, the state machine code is implemented.

Note that this module uses the Dynamic Clock Generator as its clock input so that frequency sweeping of the Barcode LED transmission rate is performed.

Design Considerations

SPI Interface

This section describes the SPI interface between Barcode Emulation and the Application Processor

The Application Processor obtains sensor data over SPI lines through the spi_slave module. This module expects SPI in mode “3” format, i.e. CPHA = 1 and CPOL = 1, and MSB first while transmitting a byte of data over the bus.

The following timing diagrams show various read/write access patterns. Multi byte transaction is supported only for write operation.

Figure 5. Single Byte Read Operation (Read FIRMWARE_VERSION)

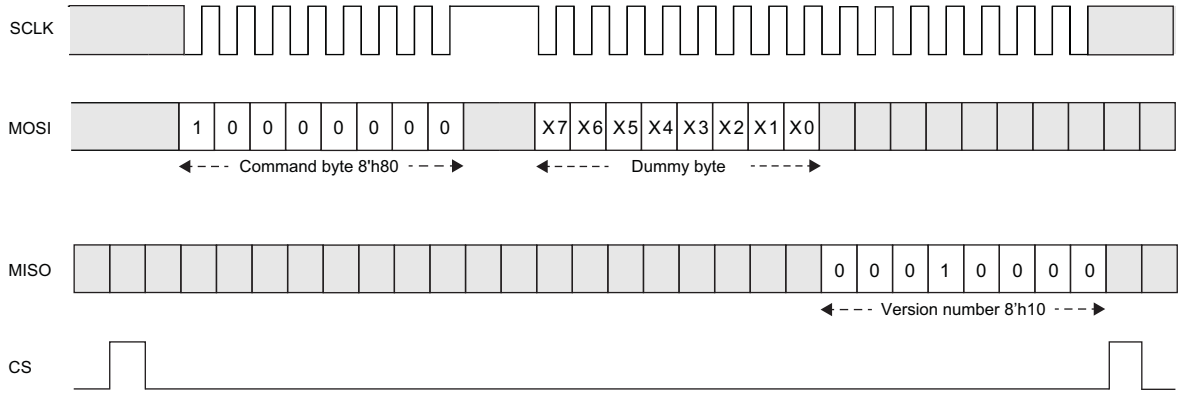
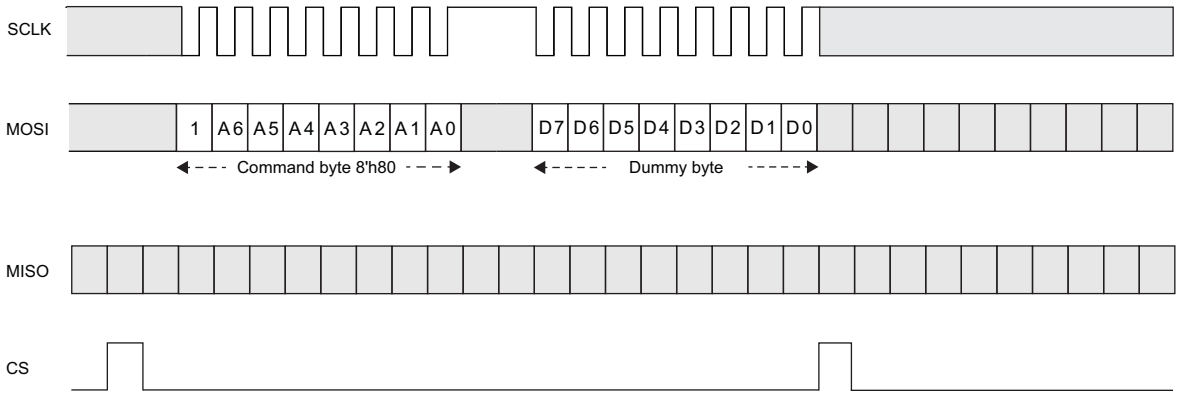


Figure 6. Single Byte Write Operation



Notes:

- In the above timing diagrams, Single Byte Read Operation is only available for FIRMWARE_VERSION register
- A6 to A0 in the Command byte indicate address of the register. See “SPI Register Descriptions” for more details.
- For a read operation from processor, MSB of command byte is always 1.
- For a write operation from processor, MSB of command byte is always 0.
- CS must not be asserted until all the bytes are read in case of multiple bytes read
- Multiple byte write operation is available only for DATA registers. Bits A6 to A0 must have a valid start address for DATA registers. After sending the command byte, consecutive data can be sent via SPI bus.

SPI Registers Description

The Barcode Emulation contains registers that can be accessed by SPI interface. These registers are accessed by A6 to A0 bits of the Command byte. The following table describes registers accessed by the A6 to A0 bits.

Table 8. Register Map for A6 to A0 Bits

Address (A6 to A0) as 7-bit hex)	Register Name	Access Type	Description
0x00	FIRMWARE_VERSION	R	Indicates the firmware version
0x02	CONTROL	W	Control register
0x03	CBR	W	CBR register
0x04	CDR	W	CDR register
0x10 to 0x4F	DATA	W	Data registers

Table 9. FIRMWARE_VERSION Register Bit Description

7	6	5	4	3	2	1	0
Firmware Version							

Table 10. CONTROL Register Bit Description

7	6	5	4	3	2	1	0
START	STOP	CYCLES					

Table 11. CBR Register Bit Description

7	6	5	4	3	2	1	0
Code Bit Rate (bit rate x250ns)							

Table 12. CDR Register Bit Description

7	6	5	4	3	2	1	0
Code Delay Rate (delay between transmission x10us)							

Table 13. DATA Registers Bit Description[1]

7	6	5	4	3	2	1	0
Code 39 type of barcode encoding							

1. DATA registers are part of Data Buffer.

Complete SPI Registers Location

Table 14 lists the first byte to be transmitted from SPI master (AP) to iCE40 on MOSI Line. This is combination of register address listed in SPI Registers Description section and also the control signal values listed after the SPI Timing diagram (under Notes).

Table 14. First Byte from SPI Master (AP) to iCE40 on MOSI Line

First Byte for SPI Read (1, A6 to A0 as 8-bit hex)	First Byte for SPI Write (0,A6 to A0 as 8-bit hex)	Register Description
0x00	-	Indicates the firmware version
-	0x02	Control register
-	0x03	CBR register
-	0x04	CDR register
-	0x10 to 0x4F	Data registers

Pseudo Code Example for Application Processor

The following code illustrates how an Application Processor could process the interrupt received from the IR Rx Solution to obtain the IR received data set.

```
Write Code 39 type of barcode encoding data to DATA Registers
Write data to CBR Register
Write data to CDR Register
Write commands to CONTROL Register (assert Transmit Start, deassert Transmit Stop)
// After some elapsed time
Write commands to CONTROL Register (deassert Transmit Start, assert Transmit Stop)
```

Design Customization Considerations

Since this is an FPGA based solution, user can customize this solution by changing the source code of the Barcode Emulation solution or add additional functions to this solution. Note that when customization is performed, the “Performance Characteristics” values might change.

Programming Solutions

Due to the FPGA nature of this solution, the solution requires FPGA programming. The programming solutions include, but not limited to programming via FTDI chip, programming via SPI Flash, or programming via application processor. For more information on programming solutions, please refer to “iCE40 Configuration Solutions Guide”.

Power Supplies

Please refer to FPGA board design guide.

Layout Guidelines

Please refer to FPGA board design guide.

Heatsink Selection

Please refer to FPGA board design guide.

Software Requirement

For standalone solution, Diamond Programmer and “barcode_fsm_top_bitmap.hex” file. The following steps are required to program the device:

1. Create a new project
2. Set to SPI Programming

For fully customizable solution, iCEcube2, Diamond Programmer, and IR Rx HDL source files are required. For more information on iCEcube2, please refer to the iCEcube2 webpage.

Resource Utilization

LUTs	Registers	PLBs	BRAMs	I/Os	I2Cs	SPIs
356	244	70	2	6	0	1

Typical Application Circuits

Figure 7. Barcode Emulation Solution with Pre-programmed SPI Flash

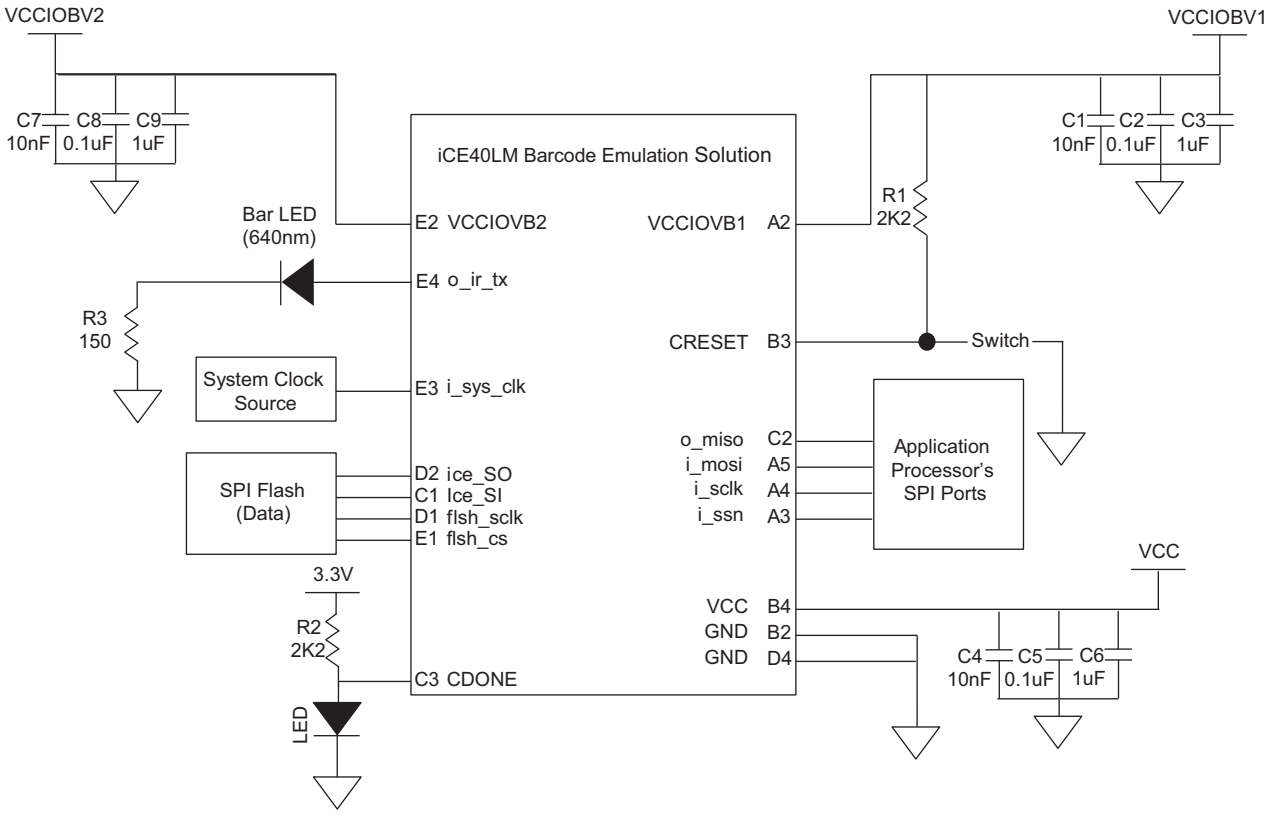


Figure 8. Barcode Emulation Solution with Direct Programming through FTDI

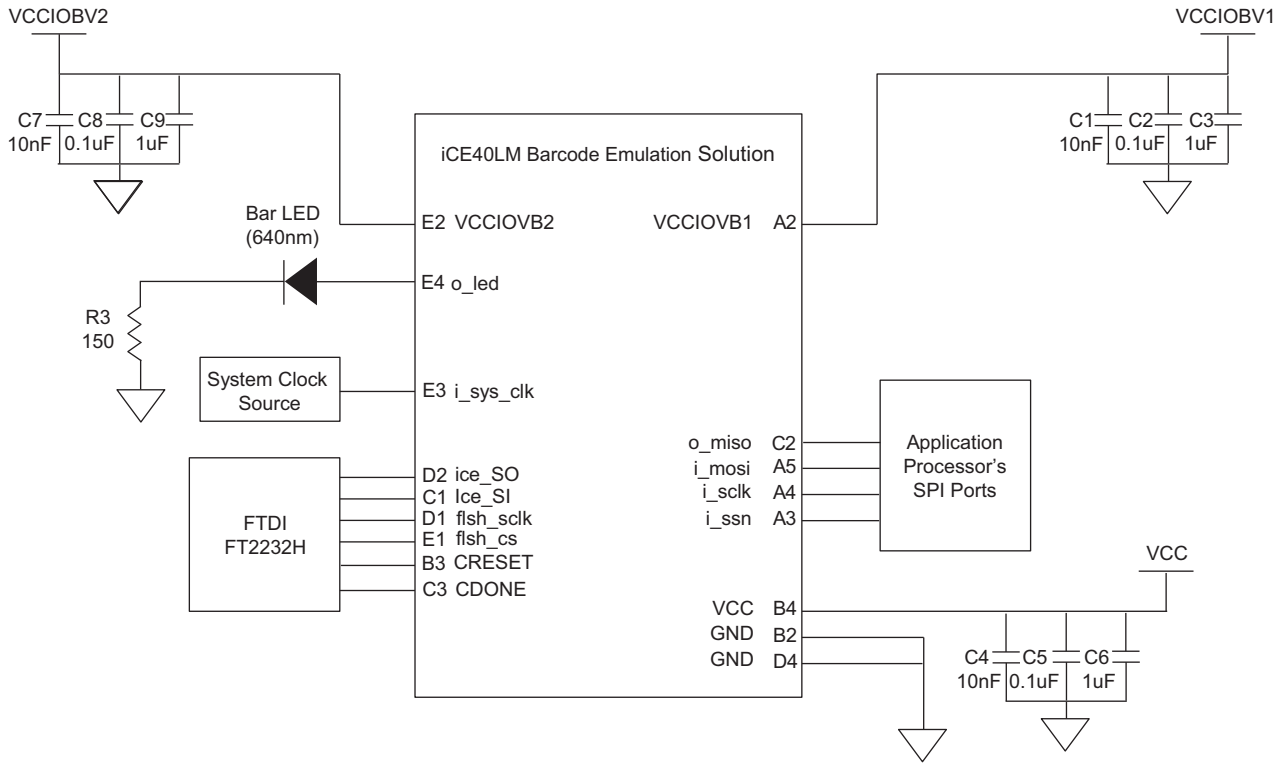
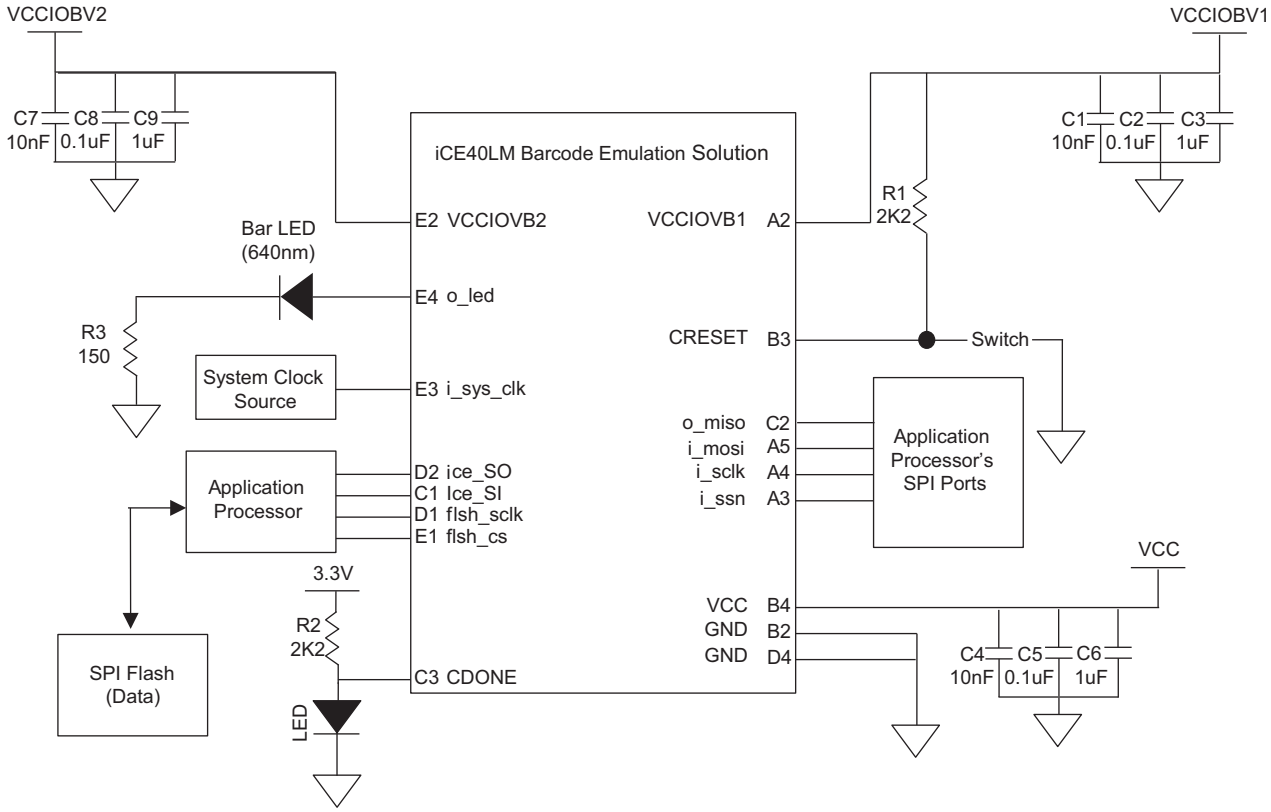
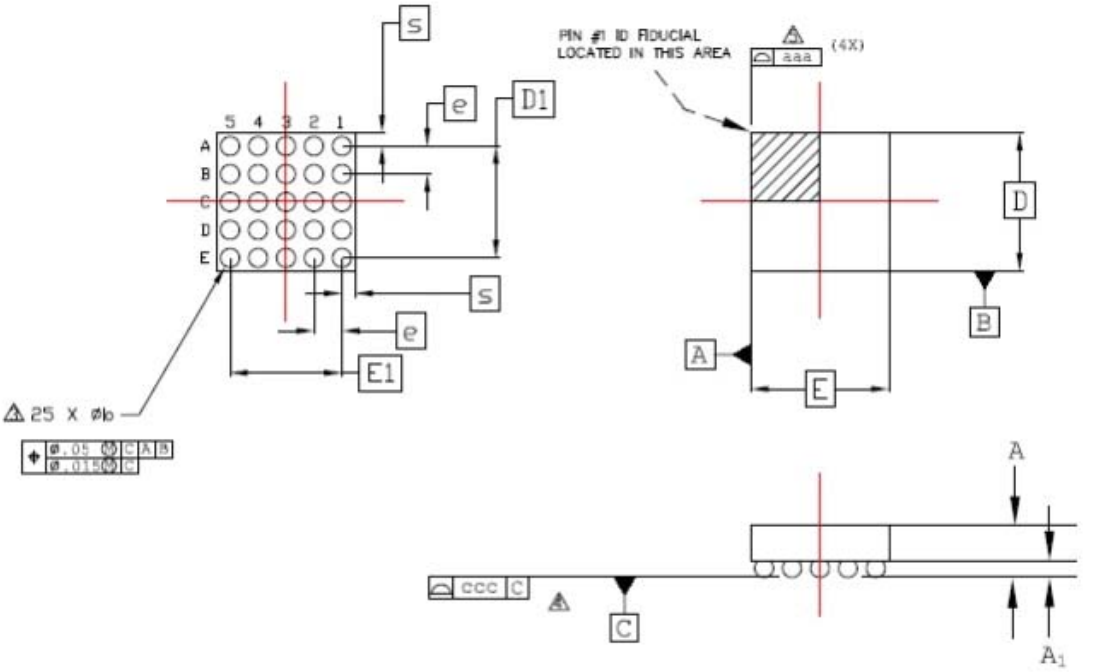


Figure 9. Barcode Emulation Solution with Programming through Application Processor



Package Diagram



Notes:
1 ALL DIMENSIONS AND TOLERANCE PER ASME Y 14.5M – 1994.
2 ALL DIMENSIONS ARE IN MILLIMETERS.
3 DIMENSION "b" IS MEASURED AT THE MAXIMUM BUMP DIAMETER PARALLEL TO PRIMARY DATUM C.
4 PRIMARY DATUM C AND SEATING PLANE ARE DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BUMPS.
5 BILATERAL TOLERANCE ZONE IS APPLIED TO EACH SIDE OF THE PACKAGE BODY.

REF.	Min.	Nom.	Max.
A	0.413	0.452	0.491
A1	0.122	0.152	0.182
b	0.188	0.218	0.248
D		1.71 BSC	
E		1.71 BSC	
D1		1.40 BSC	
E1		1.40 BSC	
e		0.35 BSC	
aaa		0.03	
ccc		0.03	
s	–	0.155	–

Disclosures

The Barcode Emulation solution is an FPGA based solution which requires IP to be downloaded to the device for this solution. This solution includes the Diamond Programmer for IP download and iCEcube2 design software for customization. The design files and ready-for-download .hex file are also included. Finally, SPI Flash might be needed depending on whether one time or multi programmable scheme is used.

Ordering Information

Solution Name	Description	Package	BOM
Barcode Emulation Reference Design (Commercial Grade)	Commercial Grade Solution	25-pin WLCS at 1.71mm x 1.71mm	iCE40LM4K-SWG25TR Device, iCEcube2 Design Software, Diamond Programmer, Barcode Emulation Design Files, barcode_fsm_top_bitmap.hex
Barcode Emulation Reference Design (Industrial Grade)	Industrial Grade Solution	25-pin WLCS at 1.71mm x 1.71mm	iCE40LM4K-SWG25TR Device, iCEcube2 Design Software, Diamond Programmer, Barcode Emulation Design Files, barcode_fsm_top_bitmap.hex

Technical Support Assistance

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com

Revision History

Date	Version	Change Summary
October 2013	01.0	Initial release.
	01.1	Updated the Single Byte Read Operation (Read FIRMWARE_VERSION) figure.
		Updated the Single Byte Write Operation figure.