Dünner Kirchweg 77
32257 Bünde
Germany
www.trenz-electronic.de

Full-Speed USB 1.1 Function Controller

2000-September-7

Product Specification

Introduction

The lack of flexibility in reconfiguring the PC has been acknowledged as the Achilles' heel to its further deployment. The combination of user-friendly graphical interfaces and the hardware and software mechanisms associated with new-generation bus architectures have made computers less confrontational and easier to reconfigure. However, from the end user's point of view, the PC's I/O interfaces, such as serial/parallel ports, keyboard/mouse/joystick interfaces, etc., do not have the attributes of plug-and-play.

The USB is the answer to connectivity for the PC architecture. It is a fast, bi-directional, isochronous, low-cost, dynamically attachable serial interface that is in line with the requirements of the PC platform of today and tomorrow.

While several microcontrollers provide USB functionality, many applications require additional flexibility, reduced component count or maximum performance. These requirements are fully addressed by the Full-Speed USB Function controller while preserving development cycles as short as possible.

Features

- Fully compliant to USB 1.1 specification
  - Full-Speed (12Mbps) operation
  - Support for 4 Endpoints, including up to 3 user-configurable Endpoints
  - Supports Bulk, Interrupt or Isochronous data transfers
- Hardwired USB protocol layer
  - No firmware intervention required
  - Up to 10Mbps bandwidth
- Very compact design
  - On-chip digital PLL
  - On-chip Endpoint FIFOs
  - Minimum gate count
  - Optimized for FPGA implementation
- Lowest possible design risk
  - Free behavioral model
  - Comprehensive reference application
  - USB Packet-oriented testbench
  - Synthesizable VHDL model
- Very low cost

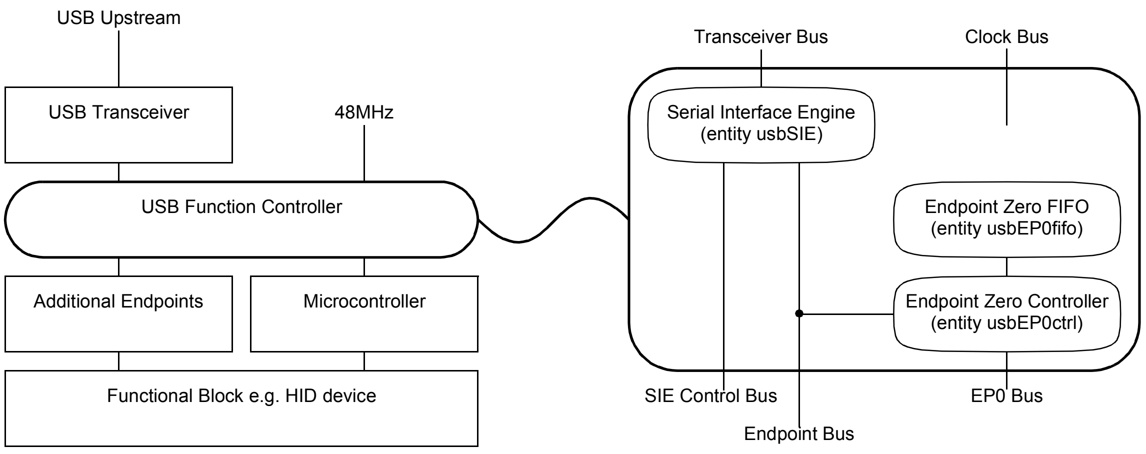
USB Upstream
USB Transceiver
48MHz
USB Function Controller
Additional Endpoints
Microcontroller
Functional Block e.g. HID device
Transceiver Bus
Clock Bus
Serial Interface Engine (entity usbSIE)
Endpoint Zero FIFO (entity usbEP0fifo)
Endpoint Zero Controller (entity usbEP0ctrl)
SIE Control Bus
EP0 Bus
Endpoint Bus

Figure 1: System architecture.

General Overview

Figure 1 shows the system architecture of a device based on the USB Function Controller. It consists of the following components:

- *USB Transceiver.* To meet the electrical requirements of the USB specification, an external Transceiver is required. The reference application utilizes Philips Semiconductor´s PDIUSBP11A, but other devices can also be interfaced easily.
- *Microcontroller.* To implement the USB device framework a microcontroller is used. Glue logic to interface with a standard 8051 derivative is provided with the reference application, other microcontrollers can easily be interfaced.
- *Additional Endpoints.* In case the device requires additional endpoints, the associated endpoint controllers need to be implemented. A simple example is provided with the reference application.

The USB function Controller consists of the following internal building blocks:

- *Serial Interface Engine.* This entity implements the full USB protocol layer. It is completely hardwired for speed and needs no firmware intervention. The functions of this entity include: frame recognition/generation, parallel/serial conversion, bit-stuffing/de-stuffing, CRC checking/generation, PID verification/generation, address recognition and handshake evaluation/generation.
- *Endpoint Zero Controller.* This entity implements the dedicated Endpoint Zero functionality, mainly FIFO control and arbitration.
- *Endpoint Zero FIFO.* This entity implements the 8byte FIFO associated with Endpoint Zero.

Architectural Description

The USB Function Controller is based on several bus systems, which interface the building blocks with each other:

- *Clock Bus.* This bus carries all clock and reset signals.
- *Transceiver Bus.* This is a bidirectional serial bus connecting the Serial Interface Engine with the external USB Transceiver.
- *Endpoint Bus.* This is a bidirectional bus connecting the Serial Interface Engine with up to 4 Endpoint Controllers.
- *SIE Control Bus.* This is a unidirectional bus connecting the Serial Interface Engine with the external microcontroller to set the device address.
- *EP0 Bus.* This is a bidirectional bus connecting the Endpoint Zero Controller with the external microcontroller.

Table 1: Clock Bus: Signals.

signal	direction	purpose
clk48	in	48MHz clock input
clk12	in	12MHz clock input
rst	in	asynchronous reset
clk12o	out	12MHz clock output

Clock Bus

The Clock Bus carries all clock and reset signals. The USB Function Controller contains two clock domains, one clocked with 48MHz and another clocked with 12MHz. Table 1 summarizes the Clock Bus signals.

The *clk48* clock domain contains the digital PLL which outputs the reconstructed USB clock at *clk12o*.

All other building blocks are clocked by *clk12*. This signal is usually directly driven by *clk12o*.

The *rst* signal is used for power-on reset. No USB reset condition is generated on *rst*.

Table 2: Transceiver Bus: Signals.

signal	direction	purpose
urxd	in	receive data
urx0	in	receive SE0
utxd	out	transmit data
utx0	out	transmit SE0
utxoe	out	transmit enable

Transceiver Bus

The Transceiver Bus connects the Serial Interface Engine with the external USB Transceiver. Table 2 lists the signals building the Transceiver Bus.

Table 3 shows the relationship between the USB States and the Transceiver Bus signals.

Table 3: Transceiver Bus: Mapping.

Bus State	urxd	urx0	utxd	utx0	utxoe
receiving 1 (J)	1	0	X	X	0
receiving 0 (K)	0	0	X	X	0
receiving SE0	X	1	X	X	0
driving 1 (J)	X	X	1	0	1
driving 0 (K)	X	X	0	0	1
driving SE0	X	X	X	1	1

Endpoint Bus

The Endpoint Bus connects the Serial Interface Engine with up to 4 Endpoint Controllers. Table 4 lists the signals building up the Endpoint Bus.

Table 4: Endpoint Bus: Signals.

signal	direction	purpose
rxd(7:0)	out	receive data
rxen	out	rxd enable
txd(7:0)	in	transmit data
txen	in	txd enable
in_trac(3:0)	out	IN transaction
out_trac(3:0)	out	OUT transaction
setup_trac(3:0)	out	SETUP transaction
nak(3:0)	in	NAK handshake
stall(3:0)	in	STALL handshake
togglein(3:0)	in	DATA toggle
datain(3:0)	in	IN data pending
sof_trac	out	SOF transaction
rxfrm(10:0)	out	frame number

All signals except for the data buses, data bus enables and SOF signals are available as groups of 4 signals, one associated with each endpoint. Endpoints interfacing to *txd* should use three-state buffers enabled with *in_trac*.

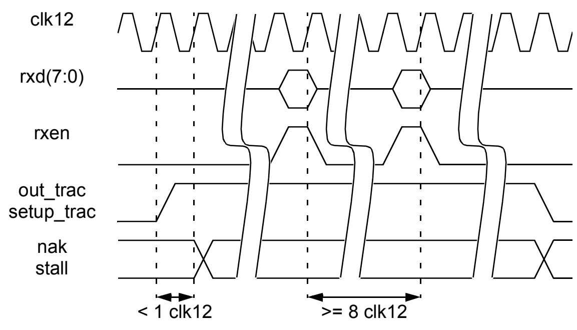

Figure 2: Endpoint Bus: Receive Timing.

Figure 2 illustrates the timing of SETUP and OUT transactions. *nak* and *setup* must become valid within 1 clock cycle after the rising edge of *out_trac* or *setup_trac*. Data bytes are delivered approximately every 8 clock cycles. Data bytes will be delivered, even if the USB packet is ignored due to NAK or STALL handshake.

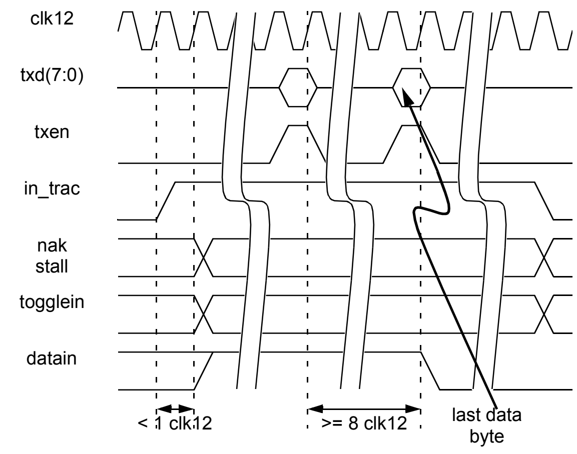

Figure 3: Endpoint Bus: Transmit Timing.

Figure 3 illustrates the timing of IN transactions. *nak*, *setup*, *togglein* and *datain* should become valid within 1 clock cycle after the rising edge of *in_trac*. Data bytes are requested approximately every 8 clock cycles. *datain* should be released with the last data byte's *txen* pulse. To send a zero-length data packet, *datain* should be pulled to zero within 1 clock cycle after the rising edge of *in_trac*.

Figure 4 illustrates SOF token timing. *sof_token* should be used to validate *rxfrm*, if glitch-free frame numbering is required.

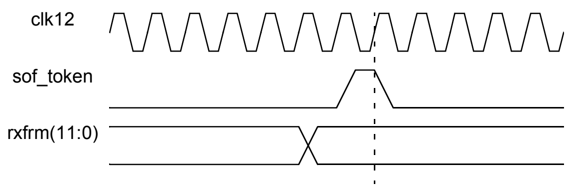

Figure 4: Endpoint Bus: SOF Token.

SIE Control Bus

The SIE Control Bus connects the Serial Interface Engine with the external microcontroller to set the device address. Table 5 summarizes the signals.

Table 5: SIE Control Bus: Signals.

signal	direction	purpose
uc_dadx(6:0)	in	USB device address
uc_wradx	in	write enable

Figure 5 illustrates device address write timing. The actual address change is delayed until 1 clock cycle after the next rising edge on *setup_trac*.

EP0 Bus

The EP0 Bus connects the external microcontroller with the Endpoint Zero Controller. Table 6 summarizes its signals.

Table 6: EP0 Bus: Signals.

signal	direction	purpose
uc_adx(2:0)	in	FIFO address
uc_drd(7:0)	out	FIFO read data
uc_dwr(7:0)	in	FIFO write data
uc_wren	in	FIFO write enable
uc_ctrl(7:0)	in	control
uc_status(7:0)	out	status
uc_wrctrl	in	control write enable

The EP0 Controller provides access to the 8 byte Endpoint FIFO. Figure 6 illustrates FIFO read and write cycles.

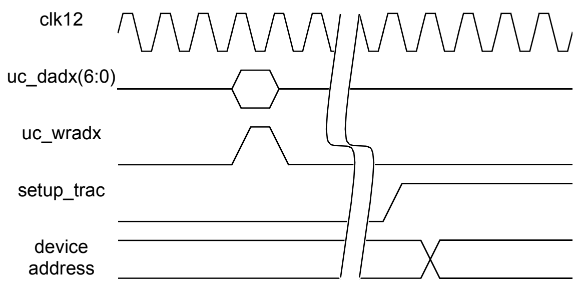

Figure 5: SIE Control Bus: Write timing.

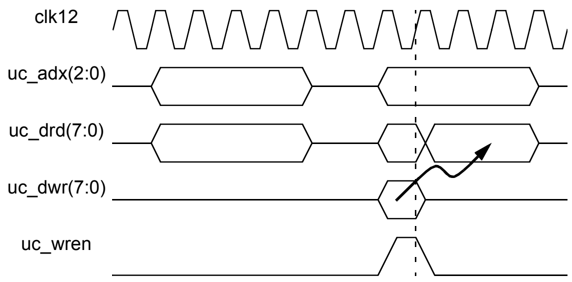

Figure 6: EP0 Controller: FIFO Read/ Write.

To control operation of the EP0 Controller, a command register is used. Table 7 illustrates the command register layout.

Table 7: EP0 Controller: Command Register.

bit	purpose
7	enable Control-Read
6	enable Control-Write/ No-Data Control
5	acknowledge Status stage
4	stall Endpoint
3:0	number of valid bytes in FIFO

The command register is write-only, Figure 7 illustrates the command register write cycle.

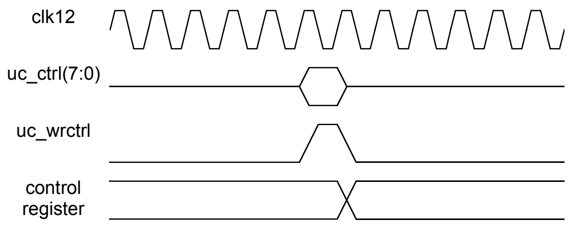

Figure 7: EP0 Bus: Command Register Write.

The EP0 Controller provides status information through a status register. *Table 8* illustrates the status register layout.

Table 8: EP0 Bus: Status Register Layout.

bit	purpose
7	setup stage
6	data stage
5	status stage
4	reserved
3:0	number of valid bytes in FIFO

The Endpoint Zero Controller interacts closely with the firmware to implement USB protocol layer sequences. *Figure 8* illustrates, how to implement Control-Read Sequences. The basic scheme is as follows:

- Setup Stage
  - Wait until status(7:4) is 1000
  - Evaluate Device Request
- Data Stage
  - Copy data into FIFO
  - Validate FIFO, set control(7:4) to1000
  - Wait for non-zero status
  - Go to Status Stage, if status(7:4) is 0100
  - Continue Data Stage
- Status Stage
  - Acknowledge, set control(7:4) to 0010

Figure 9 illustrates, how to implement Control-Write Sequences. The basic scheme is as follows:

- Setup Stage.
  - Wait until status(7:4) is 1000
  - Evaluate Device Request
- Data Stage.
  - Empty FIFO, set control(7:4) to 0100
  - Wait for non-zero status
  - Go to Status Stage, if status(7:4) is 0100
  - Copy data from FIFO
  - Continue Data Stage
- Status Stage
  - Acknowledge, set control(7:4) to 0010

No-Data Control Sequences are implemented as Control-Write Sequences without Data Stage.

Generics

Endpoint setup is controlled by a set of generics, which are listed in *Table 9*.

Table 9: Generics.

generic	purpose
epin_mask(3:0)	IN transaction mask
epout_mask(3:0)	OUT transaction mask
epsetup_mask(3:0)	SETUP transaction mask
episo_mask(3:0)	isochronous endpoint mask

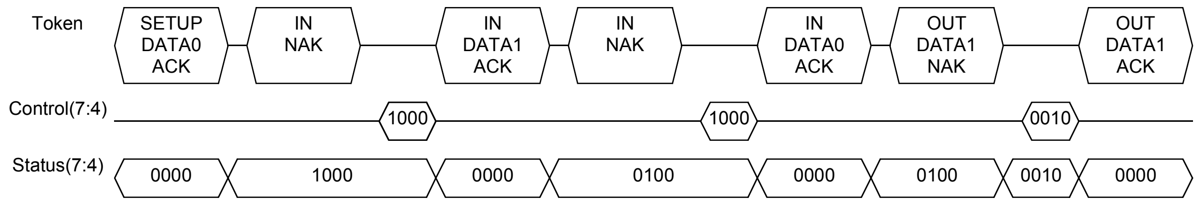

Figure 8: EP0 Controller: Control-Read.

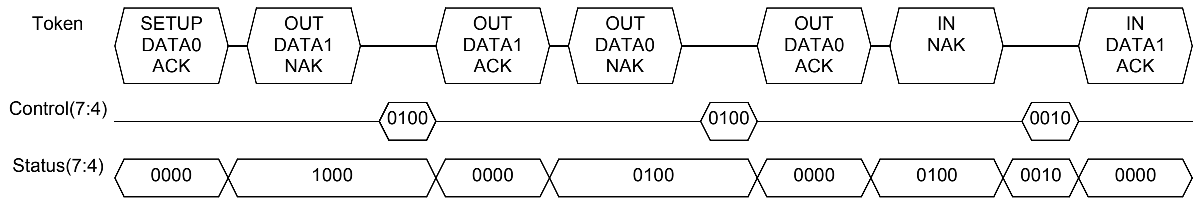

Figure 9: EP0 Controller: Control-Write.

Simulation

Simulation is a vital part of design verification. To ease the task of test vector creation, a powerful VHDL package has been created. The main features of this package are:

- USB Packet-oriented
- Stimuli creation
- Response verification

With USB packets being the atomic items of the package, complex testbenches can easily be created, which are isolated from USB protocol and timing details. Devices may be stimulated from USB and their responses to USB may be verified. This allows for simulation of complete systems including microcontroller and firmware.

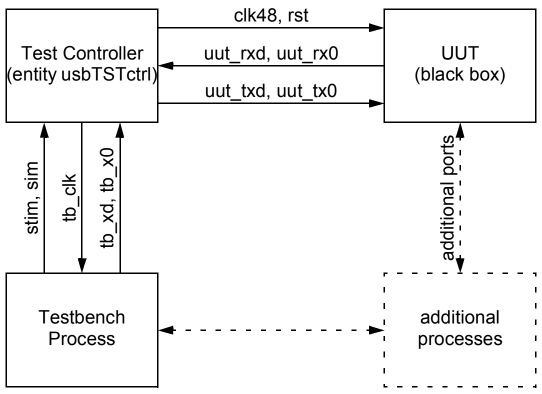

Figure 10: Simulation: Testbench Setup.

Figure 10 illustrates the basic testbench setup. It consists of the following main components:

- *UUT.* This is the Unit Under Test. It is treated as a black box and exposes its behavior via its outside ports only.
- *Test Controller.* The Test Controller stimulates the UUT and verifies the UUT's responses. In case of a mismatch, a report is asserted.
- *Testbench process.* The Testbench Process controls the Test Controller's operation based on USB Packets.

Figure 11 illustrates the Testbench process. The testbench package is implemented in *usbTSTPAK*. The signal *sim* is used to limit the total simulation time by disabling the clock oscillators inside *usbTSTctrl*. The signal *stim* is used to distinguish stimuli from responses. The example transmits the Setup Stage of a GetDescriptor Device Request and verifies that the Device answers with an ACK handshake.

```
use work.usbTSTPAK.all;
...
sim<= '1', '0' after 15us;
...
process
  constant get_dscr: packetBUF:=
    ( x"80", x"06", x"00", x"01",
      x"00", x"00", x"40", x"00");
begin
  stim<= '1';
  packetSETUP(tb_clk, tb_xd, tb_x0,
      "0000000", "0000");
  packetDATA1(tb_clk, tb_xd, tb_x0,
      get_dscr);
  stim<= '0';
  packetACK(tb_clk, tb_xd, tb_x0);

  wait;
end process;
```

Figure 11: Simulation: Testbench Process.

Table 10 summarizes the Test Controller ports. It is important to understand, that the Test Controller interfaces the Testbench Process with the UUT, instead of the Testbench Process interacting with the UUT directly.

Table 10: Test Controller Ports.

signal	direction	purpose
sim	in	'1' during simulation
stim	in	'1' to stimulate UUT
clk48	out	48MHz clock output
rst	out	asynchronous reset
uut_rxd	out	connected to UUT's urxd
uut_rx0	out	connected to UUT's urx0
uut_txd	in	connected to UUT's utxd
uut_tx0	in	connected to UUT's utx0
tb_xd	in	testbench xd signal
tb_x0	in	testbench x0 signal
tb_clk	out	testbench 12MHz clock

The following paragraphs summarize the procedures contained in *usbTSTPAK*.

packetIN, packetOUT, packetSETUP

```
(signal clk: in  STD_LOGIC;
 signal xd:  out STD_LOGIC;
 signal x0:  out STD_LOGIC;
 addr:       in  STD_LOGIC_VECTOR
                 (6 downto 0);
 ep:         in  STD_LOGIC_VECTOR
                 (3 downto 0));
```

packetIN, *packetOUT* and *packetSETUP* initiate an IN, OUT or SETUP transaction to the address/endpoint specified by *addr* and *ep*.

packetSOF

```
(signal clk: in  STD_LOGIC;
 signal xd:  out STD_LOGIC;
 signal x0:  out STD_LOGIC;
 frame:      in  STD_LOGIC_VECTOR
                 (10 downto 0));
```

packetSOF sends a Start-Of-Frame token with the frame number specified by *frame*.

packetDATA0, packetDATA1

```
(signal clk: in  STD_LOGIC;
 signal xd:  out STD_LOGIC;
 signal x0:  out STD_LOGIC;
 data:       in  packetBUF);
```

packetDATA0 and *packetDATA1* send/receive a DATA package with the contents specified by *data*. In case *data* is omitted, a zero-length packet is sent/received.

packetACK, packetNAK, packetSTALL

```
(signal clk: in  STD_LOGIC;
 signal xd:  out STD_LOGIC;
 signal x0:  out STD_LOGIC);
```

packetACK, *packetNAK* and *packetSTALL* sends/receives an ACK, NAK or STALL handshake.

packetIDLE

```
(signal clk: in  STD_LOGIC;
 signal xd:  out STD_LOGIC;
 signal x0:  out STD_LOGIC;
 cycles:     in  INTEGER);
```

packetIDLE keeps the bus in idle (J) state for the number of USB bit cycles specified by *cycles*.

packetRESET

```
(signal clk: in  STD_LOGIC;
 signal xd:  out STD_LOGIC;
 signal x0:  out STD_LOGIC);
```

packetRESET keeps the bus in SE0 state for more than 2.5us to generate a USB reset condition.

Synthesis

Synthesis of the USB Function Controller is straightforward, only very few constraints need to be considered for a successful implementation. *Table 11* summarizes the timing groups and their constraints. The design uses the two clock domains *clk48* and *clk12*. *clk12* is usually driven by the output of the internal DPLL on port *clk12o*.

Table 11: Synthesis: Timing Groups.

From	To	Required Delay
All Input Ports	RC(clk48)	20ns
All Input Ports	FC(clk48)	10ns
All Input Ports	RC(clk12)	20ns
RC(clk48)	All Output Ports	20ns
RC(clk48)	RC(clk48)	20ns
RC(clk48)	RC(clk12)	20ns
FC(clk48)	RC(clk48)	10ns
RC(clk12)	All Output Ports	62ns
RC(clk12)	RC(clk12)	62ns

The USB Function Controller contains a FIFO, which should be implemented with a technology dependent macro to achieve minimum resource usage. While some synthesizers support automatic RAM inference others require a manual black-box implementation. To do so, create the FIFO macro with the vendor-provided tools and remove the entity from the synthesis project. Instead the FIFO is added to the design during place & route. As FIFO depth is only 8 bytes, this optimization is not required by most applications.

Implementation

Assuming a technology-independent FIFO implementation was chosen, implementation issues are reduced to the resource usage of the final design.

Resource usage depends on the number of endpoints used and additional features implemented with the core. *Table 12* summarizes the synthesis results achieved with the reference application described in a separate application note.

Table 12: Synthesis: Xilinx XCS10 Results.

Resource	Usage
CLBs	190
CLB Flip Flops	203
4 input LUTs	337
3 input LUTs	52
16x1 RAMs	8
bonded IOBs	42
IOP Flops	12
IOB Latches	0
clock IOB pads	3
primary CLKs	3
secondary CLKs	1
TBUFs	24
total equivalent gate count	4106

The reference application uses a single endpoint and adds 8051 interface glue logic to the core. The following resources are clearly identified to belong to the application:

- 1 primary CLK is used to buffer the 8051's ALE signal.
- 1 secondary CLK is used to buffer the 8051's WR signal.
- 8 TBUFs are used to isolate the 8051's P0 port.
- 16 TBUFs are used to isolate the glue logic's internal data bus.
- 4 CLBs are required to de-multiplex the 8051's P0 bus.
- 4 CLBs are required to implement the 7-segment LED register

Evaluation Package

To achieve a risk-free evaluation of the USB Function Controller, an Evaluation Package is available. The Evaluation Package is created from a post-synthesis VHDL simulation model with the generics being set up according to *Table 13*. In conjunction with the testbench package, complete system simulation is possible prior to product purchase.

Table 13: Evaluation Package: Generics.

generic	value
epin_mask(3:0)	0001
epout_mask(3:0)	0001
epsetup_mask(3:0)	0001
episo_mask(3:0)	0000

References

- Universal Serial Bus Specification
  USB Implementers Forum
  http://www.usb.org
- Xilinx Inc.
  2100 Logic Drive
  San Jose, CA 95124
  Phone: +1 408-559-7778
  Fax: +1 408-559-7114
  http://www.xilinx.com

Revisions History

Table 14: Revisions History.

Version	Date	Who	Description
1.0	00aug26	FB	Initial version