ALTERA® CYCLONE™ FPGA EVALUATION BOARD

SOCkit User Guide

Dallas Logic
corporation

2300 McDermott #200-305
Plano, TX 75025
www.dallaslogic.com

Version 1.2-February 2004

Table of Contents

Introduction

Thank you for purchasing Dallas Logic's SOCkit (System On a Chip) FPGA evaluation board. This FPGA kit has everything you need to start designing with and evaluating the powerful features of Altera® Cyclone™ FPGA devices. Whether you just want to learn about FPGA design, or have a specific design implementation to complete, this evaluation board will jump-start the your own Cyclone™ efforts. Altera's Quartus II Web-pack software which contains Nios SOPC builder (trial version) is a state of the art software tool which allows coding, compilation, and simulation of fairly complete SOC designs in a programmable logic environment. Your SOCkit includes the following items:

- SOCkit evaluation board
- External 16 character X 2 line LCD module
- 14 pin LCD ribbon cable
- LVDS patch cable
- Wall mount type 5VDC/2A switching power supply (US or Asian/European model, depending on which was ordered)
- DB-9 RS232 cable (male to female, wired straight thru)
- ByteBlaster II FPGA programming cable
- Altera® "web-pack" Quartus II CD (SOPC builder trial version and GNUPro C toolkit).
- SOCkit reference design and data files CD.

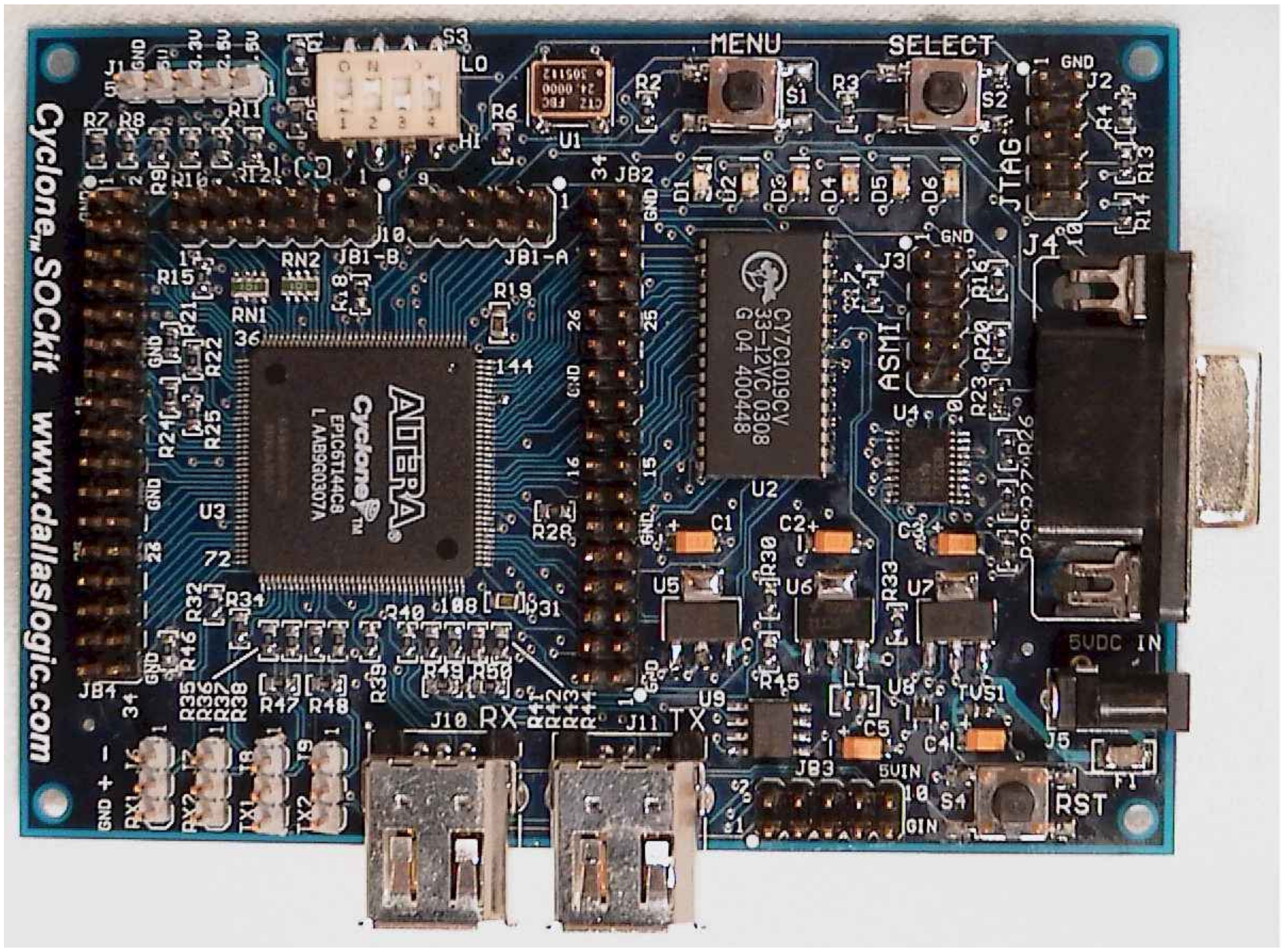
ALTERA
Quartus II Software Starter Suite
July 2003
Everything You Need to Get Started Designing for Altera® Stratix™, Cyclone™ & MAX® Devices
ALTERA
ByteBlaster II
Cyclone SOCkit www.dallaslogic.com

Although the SOCkit board will run by itself "out of the box", you will need access to a PC to view and explore the reference design files. Altera's Quartus II software supports both Windows® and Linux operating systems. Consult Altera's website at www.altera.com for specifics on operating system requirements. As of this writing, Quartus II version 3.0 for Windows® requires Microsoft Windows® NT4.0 (SP3), 2000, or XP. Users who need support for Windows® 98 can download Quartus II Web Edition version 2.2 software including service pack 2.

A male to female DB25 "LPT printer port" cable will be required to connect Altera's ByteBlaster II programmer cable to your PC (The ByteBlaster can be plugged directly into a notebook PCs LPT port, but the connector cable is only 11.5 inches long). See the section on Software and Hardware setup for more details on setting up your SOCkit.

Note that Altera® and Cyclone™ are registered trademarks of Altera Corporation. Windows® is a registered trademark of Microsoft Corporation. No further reference to this will be made.

A close-up picture of the SOCkit card is shown below:



SOCkit Board Quick Start

To quickly see your SOCkit board function, complete the following steps:

1. Locate your SOCkit board and LCD module so as to not short any of the pins on the bottom of the PCB. Make sure to clean away any loose wire or solder from the SOCkit/LCD bench area.

2. Connect the external LCD module to header JB1-B using the supplied 14 pin ribbon cable. Be sure to properly orient pin 1 on the ribbon cable (red wire) to pin 1 on the SOCkit board header connector and LCD header connector. Pin one is labeled on each header connector (labeled bottom of LCD PCB).

3. Verify that Dip-Switch S3, switch 1 is in the "on" position (selects boot Nios processor from flash, instead of load via RS-232 using the GERMS monitor).

4. Plug the external power-supply into a wall outlet. If outside the U.S., make sure you received the Asian/European model and have an appropriate plug adapter (supply input voltage is 230V AC, 50/60Hz in some areas overseas).

5. Connect the external power supply to J5 (1.3mm DC jack).

6. Your SOCkit evaluation board will power-up and boot the Nios processor. You can experiment with the LCD menus by pressing the "menu" and "select" buttons which are labeled on the SOCkit PCB.

WARNING!! Do not directly connect 5V devices to the SOCkit IO pins. See the section on 5V interfacing for proper connection methods.

Component Descriptions

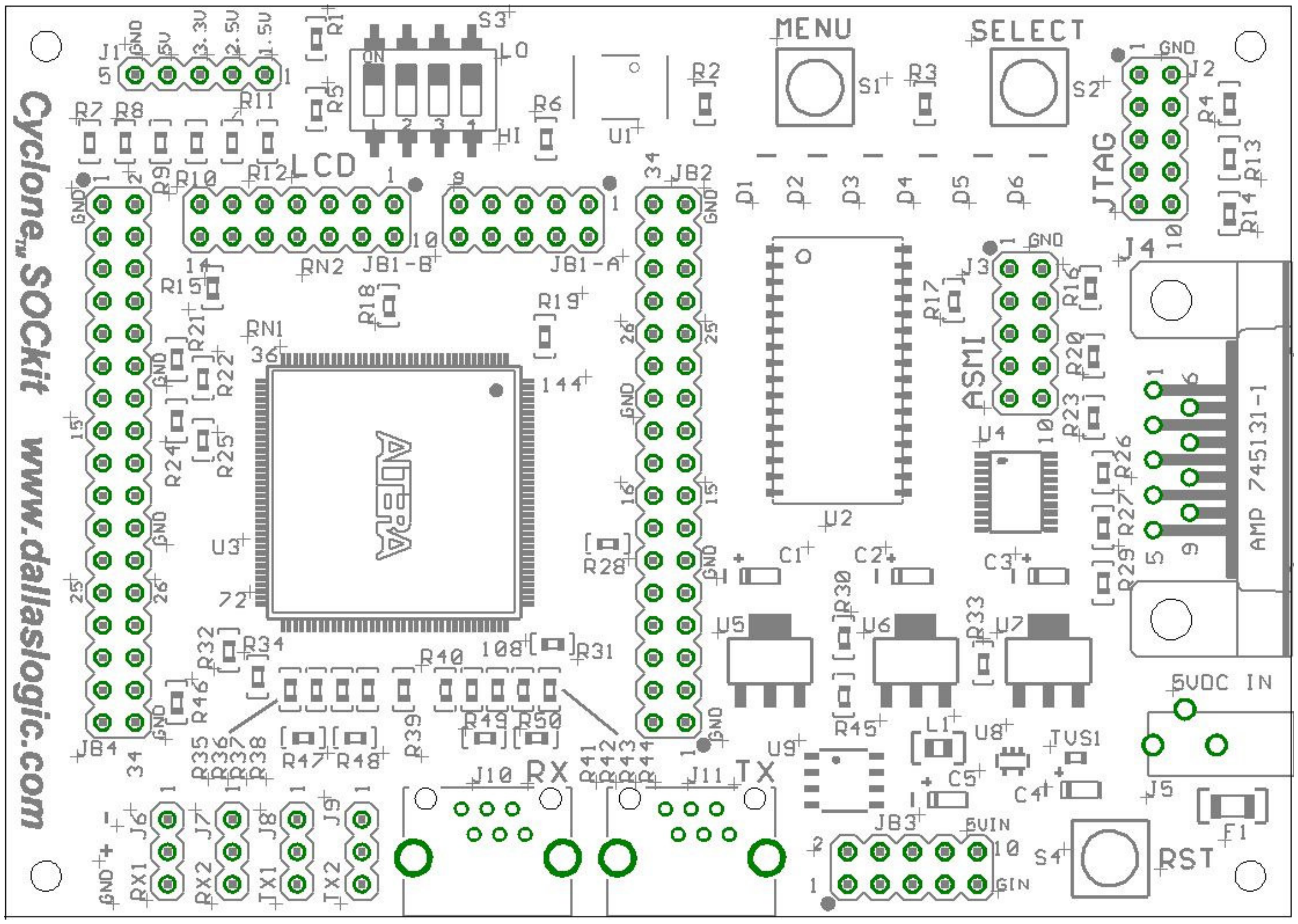
The SOCkit evaluation board provides all the "essentials" for properly implementing a compact and cost effective Cyclone/embedded Nios design. Specific components of the SOCkit board design are:

- Altera EP1C6 Cyclone FPGA, available in –6 or –8 speed grades.
- 128K x 8 bit asynchronous SRAM with 12ns access time.
- EPCS4 serial flash for FPGA configuration, Nios software image storage, and non-volatile data storage
- Separate ByteBlaster II programming ports (ASMI flash interface and JTAG interface).
- LCD interface and 2 line X 16 character LCD display with "menu" and "select" buttons.
- 8 channel LVDS support with data rates up to 312 M bit/s and provision for PLL clock transmit and receive.

- Reset button and monitor IC which provides 400mS reset pulse.
- 9 pin D-SUB connector and RS-232 interface
- Up to 94 FPGA IO available on header pins.
- Four Banks of user IO, two at 3.3V VIO, one at 2.5V (LVDS) VIO, and one voltage configurable VIO bank which is pre-set for 3.3V VIO.
- 6 discrete indicator LED (one red, one yellow, four green).
- 4 position general purpose dip-switch.
- Clock oscillator (24 Mhz).

SOCkit Board Device Placement Diagram

The SOCkit board placement diagram is shown below. Refer to this diagram to find the reference designators of components discussed in this user manual. All pin headers are placed on 0.1 inch spacing to allow the use of standard prototyping "perf-board" with the SOCkit.



6

Altera Cyclone Device

The SOCkit evaluation board comes populated with one Altera EP1C6 Cyclone FPGA in the 144 pin TQFP package (U3). Boards come with either the –6 or –8 device, depending on which configuration was ordered. The 144 pin EP1C6 device provides the following programmable logic resources:

- 98 user IO pins
- 5980 Logic Elements (LE)
- 92,160 RAM bits or approximately 11.5K bytes (20 M4K RAM blocks which are 128x36 each)
- 2 PLL modules
- Four banks of user IO, each individually powered by separate VIO input pins.

Note that with the 144 pin package, one PLL device can drive and receive on dedicated PLL pins, and the other PLL can only receive on dedicated PLL pins (only drive to a user IO pin as the dedicated transmit PLL output pins are not present). Altera specifications provide for the following PLL operating frequencies based on speed grade:

Device	PLL Input Frequency	Maximum PLL output
EP1C6 -6	15.00-200 Mhz	312 Mhz
EP1C6 -7	15.00-181 Mhz	283 Mhz
EP1C6 -8	15.00-166 Mhz	260 Mhz

The maximum operating frequency of a logic design is dependent on the number/speed of logic delays and the logic fit/placement to the device. Consult Altera's Cyclone Device Handbook for more information on internal device timing and design speed estimation.

128K x 8 bit SRAM

The 128K x 8 bit asynchronous SRAM (U2) on the SOCkit board provides more RAM storage space off-chip of the FPGA. This can be used as general purpose buffer space or as execution and data space for larger Nios software programs. Small Nios routines can execute out of the internal M4K block RAMS, but this will not be enough space for larger software routines. Also using the bulk of the M4K RAM blocks for software execution means they are not available for FIFO or DPRAM type storage circuits.

The Nios software load can be copied from the EPCS4 flash device after FPGA configuration by a small resident boot program (in the Cyclone M4K blocks), or downloaded via RS232 using Altera's GERMS monitor. Both the boot program source file (.srec) and GERMS monitor option is selected when instantiating a Nios ROM (using SOPC builder) and is automatically programmed into the M4K RAM blocks at FPGA configuration.

The SRAM is a 12ns access time device. Note that for Nios interfacing, dynamic bus sizing is implemented. The Nios uses a 16 bit instruction which will require two clock cycles to fetch one instruction from memory.

EPCS4 serial flash

The EPCS4 serial flash device (U9) is used to load the FPGA hardware configuration data, and then also contains the NIOS software image which is loaded into SRAM by Nios after boot-up. The EPCS4 device is a 4Mbit device and contains 4,194,304 bits of program space. The EP1C6 FPGA requires 1,167,216 bits (non-compressed) for its configuration load, which leaves approximately 3,027,088 bits (378K bytes) for Nios software images and user data. Note that an EPCS1 device provides 1,048,576 bits of storage. This will provide FPGA configuration space for the EP1C6 device (compressed mode), but will not provide adequate data space for user applications.

The EPCS4 device supports byte or buffer moves of data, and also has software driver and interface support via Altera's SOPC builder.

ByteBlaster II Programming Headers

Two separate 10 pin programming headers are provided on the SOCkit board. One header is for the ASMI flash interface (J3) on the EPCS4/Cyclone devices, and the other is for the Cyclone JTAG port (J2). Two separate programming ports allows the SOCkit design to support Flash reads and writes at the same time the JTAG port is being used for operations like Altera SignalTap logic analyzer (Quartus II feature), or in-circuit debug. Note that the ASMI port is used to program the FPGA load into the EPCS4 flash device.

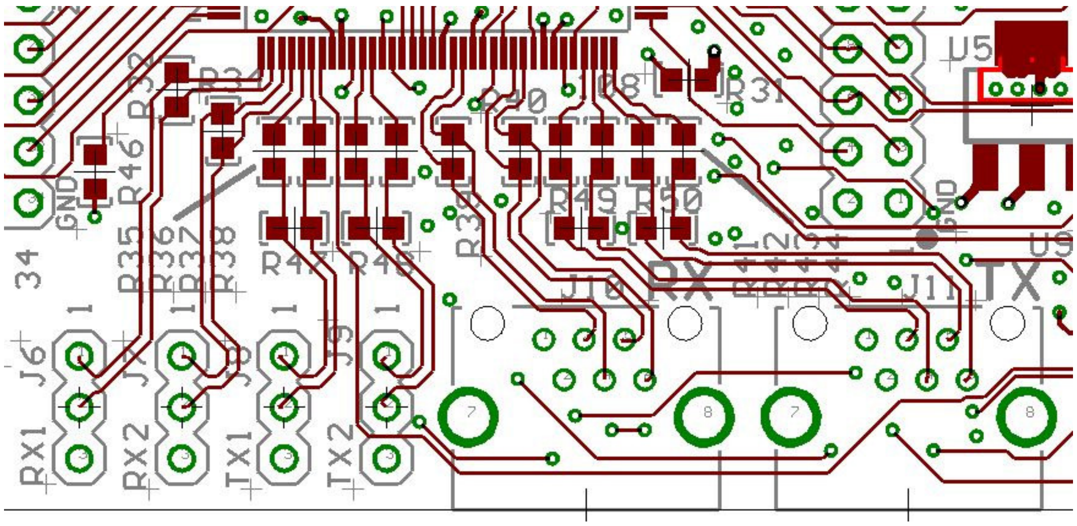
16x2 LCD Display and Menu Buttons

The LCD panel is a 16 character by two line display unit. This can be used for general character display of data and menu information. Two menu buttons can be used to select items on the LCD display. One button is labeled MENU (can be used to step through menu selections), and the other is labeled SELECT (use to select menu items). Note that the LCD is the only 5V device on the SOCkit board. See section on 5V interfacing for detailed information on 5V interfacing considerations.

8 Channel LVDS

On the SOCkit design, Cyclone FPGA I/O Bank 3 is powered by 2.5V and is dedicated for LVDS operation. There are twenty two total IO available on Bank 3, sixteen of which are used to provide eight differential LVDS channels. The unused IO pins are routed to header JB3. On the Cyclone FPGA there is no dedicated serialization/de-serialization hardware. Therefore, you have to use your own logic design for this, in addition to the on-chip PLL devices and IO pins.

All LVDS nets on the SOCkit PCB are routed as differential pairs to maintain proper differential signal impedance (nominal 100 ohm differential impedance and 52 ohm impedance to ground). Four of these LVDS channels are for transmitting data, and four are for receiving data. These differential pairs are 7 mil width at 20 mil center to center spacing. They are also 5 mil above the reference ground plane located directly underneath the top layer. A PCB image of the differential routing is shown in the following picture:



Four of the LVDS channels are connected to IEEE1394 fire-wire connectors due to the connectors controlled impedance and balanced signal properties (~100 ohm differential impedance). These channels are on header J10-clk/data receive, and J11-clk/data transmit. The other four LVDS channels are each connected to simple three pin headers (J6-RX1, J7-RX2, J8-TX1, J9-TX2). This allows for user characterization of the LVDS signals at various data rates on both controlled impedance and low cost pin headers with varying lengths of cabling. For the pin headers, shielded twisted pair or just twisted pair cable can be tested. For the IEEE1394 connectors, controlled impedance/balanced fire-wire patch cords in varying lengths can be tested. The reference design provides byte error testing of one set of LVDS channels (using the LCD display) as well as multiple test pattern transmission. Modifying the FPGA pin definitions and then initiating a Quartus II compilation changes which pins, and thus the associated header the LVDS patterns will be output to and received on. The SOCkit-6 and SOCkit-8 LVDS boards have both been tested up to 320Mbs using the supplied fire-wire patch cord. Testing was done in loop-back mode and board to board at room temperature and nominal operating voltage. This data rate slightly exceeds Altera's maximum specification. Not all boards are guaranteed to operate while exceeding specification. Note that the SOCkit uses IEEE 1394 fire-wire cables and connectors due to their electrical properties and low-cost. **J10/J11 DO NOT provide IEEE1394 physical layer signals** and should not be connected to IEEE 1394 fire-wire devices. Only attach these connectors to other SOCkit boards or LVDS IO based components.

For LVDS designs which transmit a separate clock and data signal, the clock signal can only be received to the internal Cyclone PLL on the dedicated RX PLL CLOCK pins. On the SOCkit design, these pins are on RX connector J10 which is one of the controlled impedance connectors. Therefore, to test the pin header data channels, you will still have to receive the clock signal on J10. The provided fire-wire patch cable can be used for this (loop-back), or you can "touch solder" your own cable to the bottom of the SOCkit PCB J11/J10 pins. The pin numbers for J11/J10 are present on the bottom of the PCB. Note that the length of cabling for your **data channels must match the length of cabling for the clock signals**, or else the clock and data signals will be skewed with respect to each other. At 312Mhz, you only have 3.20ns in your clock period (rule of thumb, 6 inches of conductor incurs 1ns of delay). **IMPORTANT:** When manually loop-back wiring the clock (J11 to J10) on the bottom of the PCB, make sure you "cross" the differential signals (fire-wire cable standard). This is the way it is implemented in the schematic, but it is easy to overlook and make a mistake. Wire J11 pin 5 to J10 pin 3. Wire J11 pin 6 to J10 pin 4.

Finally, for implementation of LVDS on Cyclone FPGA devices, special attention must be given to placement of differential and single-ended IO within the same bank. Also the pin assignment and routing of a data/clock pair must be done while paying attention to incurred timing skew. See Altera's Cyclone Device Handbook, section 9 titled "Implementing LVDS in Cyclone Devices" for details on Cyclone LVDS design.

Reset Circuit and Button

The SOCkit board provides a TPS3802 voltage monitor (U8) and reset button (S4). This reset signal is active low and drives the dedicated reset pin on the Cyclone FPGA device. A Quartus II setting defines this Cyclone FPGA pin to be either a dedicated chip wide reset or a user IO. Even if the pin is defined as user IO, the signal can still be used to reset individual circuits within a given design.

The TPS3802 will provide a 400mS reset pulse for the following events:

- Board power up
- Press of reset button
- 3.3V power out of range
- Following FPGA configuration and loading

During FPGA initialization, the conf_done (open drain) pin from the FPGA asserts the MR- (manual reset) input pin of the TPS3802 and causes a logic reset to the FPGA and any external devices connected to the reset signal on the pin header. Once the FPGA has finished loading, conf_done will float high, and reset will de-assert approximately 400ms after FPGA configuration has finished.

RS232 Interface

The RS-232 interface provides RX, TX, CTS, and RTS signals. Connection is made via J4 which is a female DB-9 connector. A DB-9 RS-232 male to female cable is provided in your SOCkit , and is used to connect the SOCkit RS-232 port to a PC. This cable is wired "straight-thru" and does not have any crossed wires as does a null modem cable. The RS-232 interface supports Altera's GERMS monitor, and general Nios program loading and debugging. It can also be used for user applications.

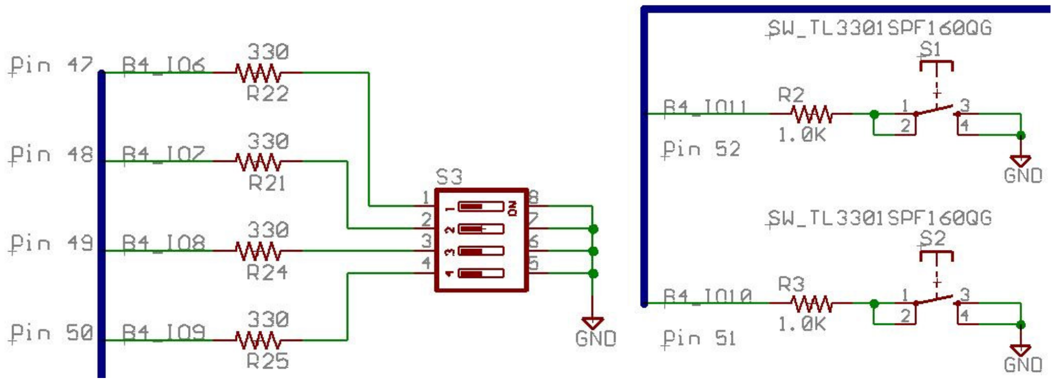
FPGA User IO Banks

There are four separate IO banks on the Cyclone device. Each has its own power pins so that multiple IO types can be supported from a single Cyclone device (IO Bank voltages in addition to the 1.5V core voltage). On the SOCkit board, banks 1,2 and 4 are powered with 3.3V. Bank 3 is powered with 2.5V in order to support LVDS operation. Note that bank 4 is powered through 0 ohm resistor R15. This allows lifting of the resistor and powering with a different IO voltage if desired. Pin headers are labeled on the PCB according to their associated IO bank number (JB1, JB2, JB3, JB4).

Indicator LED, Dipswitch

The SOCkit board provides six general purpose LED outputs (four green, one yellow, one red), and four general purpose dip-switch inputs. Refer to schematic page 6 for details on which FPGA I/O pins are used for the LED illumination. The switch schematics are shown below. S3 provides a general purpose dip-switch and should be used in conjunction with the programmable pin pull-up resistors on the Cyclone FPGA. S1/S2 provide general purpose push-button switches and are labeled as "MENU" and

"SELECT" for use with the LCD display and the SOCkit reference design. Note that a timer based de-bounce circuit is used along with input switches S1 and S2.



Clock Oscillator

There is a 24.000 Mhz oscillator (U1) installed on your SOCkit board. This value is evenly divisible by a baud rate of 19.2K (1250 x 19200 = 24000000). It also works well to generate the LVDS data rates required, and is above the 15.0 Mhz required minimum PLL input frequency for Cyclone FPGAs. SOCkit boards are shipped with either the EP1C6-8 or EP1C6-6 devices which have a maximum PLL operating frequency of 260 and 312 Mhz respectively. For the -8 speed grade EP1C6, this requires use of a 4/3 PLL ratio (at 24 Mhz) which results in an LVDS working frequency of 32 Mhz (4/3 * 24 = 32). The resulting LVDS transfer frequency is 256Mhz (32Mhz X 8). For the -6 speed grade part, this requires use of a 13/8 PLL ratio (at 24 Mhz) which results in an LVDS working frequency of 39 Mhz (13/8 * 24 = 39). The resulting LVDS transfer frequency is 312 Mhz (39Mhz X 8). The table below shows device speed grades along with their associated PLL ratios and LVDS data rates.

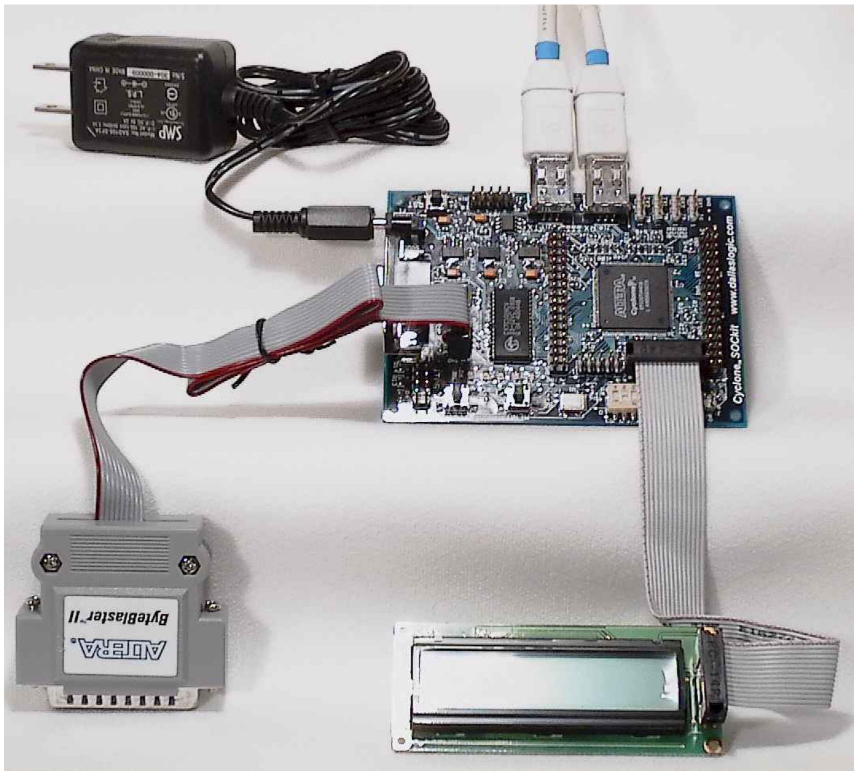
SOCkit Part Number	Device	PLL Input Frequency	Max. Specified PLL output	PLL Ratios at 24Mhz
SOCkit-6	EP1C6 -6	15.00-200 Mhz	312 Mhz	(24 * 13/8) * 8 = **312**
SOCkit-8	EP1C6 -8	15.00-166 Mhz	260 Mhz	(24 * 4/3) * 8 = **256**

Altera EP1C6 devcies support two on-chip PLL modules. Other PLL ratios can be implemented depending on design clock requirements. Both the SOCkit -6 and SOCkit-8 LVDS designs were tested at transfer rates of 320Mhz at normal operating conditions (room temperature and nominal voltage). This slightly exceeds Altera's maximum specified frequency.

Software and Hardware Setup

SOCkit Board Setup

The SOCkit board and accessory cables should be connected as shown in the picture below:



Detailed steps to configure and run your SOCkit board are:

1. Locate the PCB and LCD module so as to not short any of the pins on the bottom of the PCB. Make sure to clean away any loose wire or solder from the SOCkit/LCD bench area.

2. Verify that Dip-Switch S3, switch 1 is in the "on" position (selects boot Nios processor from flash, instead of load via RS-232 using the GERMS monitor).

3. Connect the external LCD module to header JB1-B using the supplied 14 pin ribbon cable. Be sure to properly orient pin 1 on the ribbon cable (red wire) to pin 1 on the SOCkit board header connector and LCD header connector. Pin one is labeled on each header connector (labeled bottom of LCD PCB).

4. Loop-back LVDS connector J13 to J12 with the supplied LVDS patch cord (if you want to run LVDS tests).

5. Connect the ByteBlaster II cable to J2 (JTAG) programming header.

6. Connect the DB-9 RS-232 cable to J4 on the SOCkit board, and an available 9 pin RS-232 COM port on your PC. You will need to know which specific COM port the cable is plugged into (COM1, COM2, etc).

7. Plug the external power-supply into a wall outlet. If outside the U.S., make sure you received the Asian/European model and have an appropriate plug adapter (supply input voltage is 230V AC, 50/60Hz in some areas overseas).

8. Connect the external power supply to J5 (1.3mm DC jack).

Note that the SOCkit board can accept 5V input voltage via the 1.3mm DC jack (J5-center pin positive). It can also be powered by using pin 9 and pin 10 of JB3 (for use with a bench-top power supply). Pin 9 is Input Ground and Pin 10 is for 5V. J1 provides power output connections, and is labeled with 5V, 3.3V, 2.5V, 1.5V, GND. These outputs can be used when connecting external components to the SOCkit assembly. The input fusing on the SOCkit board is limited to 1A. For bench-top power supply operation the SOCkit is designed to operate at **5V +/- 5%** and requires approximately 200 mA of current. Do not exceed this voltage specification when applying external power.

Altera Software and ByteBlaster II

Your SOCkit comes with Altera's Web-pack development CD and a ByteBlaster II programming cable. The Web-pack CD should be installed on your PC to provide Quartus II and Nios/SOPC support (Nios/SOPC install is optional). Quartus II is used to compile your design files and Program the Cyclone FPGA on the SOCkit board (via the ByteBlaster II programmer). Nios/SOPC installation is required only to support implementation of Nios processor designs. Quartus II version 3.0 for Windows requires Microsoft Windows NT4.0 (SP3), 2000, or XP. Users who need support for Windows 98 can download Quartus II Web Edition version 2.2 software including service pack 2. You may be required to request a web-pack license from Altera, even though the software is essentially free.

Web-pack Quartus II provides incredible design capability and fully supports the Cyclone line of FPGA devices (web pack has limited compile functionality for Stratix and higher end FPGA devices). The web-pack SOPC builder version is a trial version and implements a "clock-counter" limitation for Nios processors. This means that a Nios processor instantiated with web-pack will quit functioning after a few days (dependent on clock frequency). For learning or testing on the SOCkit board this is not a serious limitation, but to implement working Nios designs on your own products, you will have to acquire a full Nios license from Altera. The reference design Nios implementation that came with your SOCkit board does not have this limitation. This will only apply to Nios designs that are generated with the web-pack version of SOPC builder.

The ByteBlaster II cable is plugged into the LPT (printer) port of your Windows/Linux PC. For detailed information on Quartus II/ByteBlaster II installation, setup, and requirements access www.altera.com and reference their online documentation.

GNUpro C Development Environment

The Quartus II web-pack CD contains the C development environment for Altera Nios. This is the standard GNUpro C development environment which has some extras for working with Altera Nios. This option should be installed to your PC if you want to write Nios routines in C or change the programs that came with the SOCkit reference design.

5V and external Interfacing (LCD Module)

WARNING!! Do not directly connect 5V devices to the SOCkit IO pins. Read further for proper 5V interfacing methods.

Cyclone FPGA devices provide interfacing capability for multiple IO standards, and flexibility in interfacing. By choosing the right VIO voltage, various IO standards can be supported. But, Cyclone devices do not support 5V VIO and therefore 5V interfacing techniques must be given special attention. From Altera's Cyclone Device Handbook, section 11, the main interfacing features that Cyclone FPGA devices provide are:

- Hot-Socketing- add and remove Cyclone devices to and from a powered-up system without affecting the device or system operation
- Power-Up Sequence flexibility—Cyclone devices can accommodate any possible power-up sequence
- Power-On Reset- Cyclone devices maintain a reset state until voltage is within operating range

Please refer to chapter 11 of Altera's Cyclone Device Handbook for details on Cyclone IO interfacing. Cyclone FPGA devices provide a separate IO and core voltage power structure.

The first main consideration for 5V operation is PCI clamping diodes on the Cyclone FPGA IO pins (which are enabled by the user, during re-configuration, or if the FPGA device is not configured). The PCI clamping diodes will cause 5V signals being driven to the FPGA to essentially be shorted to ground via a 4.1V clamping diode. In this situation series resistors are required to limit the current flow out of the 5V driving pin. The second main consideration is the V-high voltage of the Cyclone IO pin that is driving to a 5V device. For TTL type 5V devices, the drive voltage is sufficient (voltage above 2.4V). For 5V CMOS type devices, the drive voltage may not be sufficient to signal a high voltage level to the 5V device. Note that adding a pull-up resistor will not provide a higher voltage than the output voltage that is being driven by the Cyclone IO pin.

On the SOCkit board, The LCD module is the only 5V device. The interfacing precautions taken are the use of 100 ohm series resistors (RN1, RN2) on the bi-directional LCD data bus. The LCD data bus pins are the only 5V pins that can drive back to the Cyclone FPGA (during FPGA re-configuration, FPGA not configured, or FPGA tri-stated). Note that the LCD module is always written to in normal operation (no reads) and will not drive back to the FPGA unless the R/W- and E pins are asserted accordingly by the FPGA.

Header J1 provides 1.5, 2.5, 3.3, and 5V pins for powering external devices. Make sure you **properly configure any 5V circuits** which you attach to the SOCkit board (add series resistors and check voltage high requirements). Improper connection of 5V circuits can **damage the SOCkit Cyclone FPGA** or the external components which you are interfacing to. Finally, also make sure you do not exceed the power and current capability of the SOCkit board when connecting and powering external devices (SOCkit power input has a 1 amp fuse at 5V DC).

Altera Reference Documents

This User Manual is a guide to the SOCkit reference design as well as a "quick start" guide for basic Nios and Quartus II operations. It is not intended to replace the standard set of Altera documentation which is necessary for detailed design of Cyclone/Nios systems. All of Alteras documentation is available online at www.altera.com. In particular, the following documents should be referenced:

- Cyclone Device Handbook , Volume I and II
  http://www.altera.com/literature/hb/cyc/cyclone_device_handbook.pdf

- Serial Configuration Devices Datasheet,
  http://www.altera.com/literature/hb/cyc/cyc_c51014.pdf

- Nios 32bit Programmers Reference Manual,
  http://www.altera.com/literature/manual/mnl_nios_programmers32.pdf

- Nios Avalon Bus Specification and Reference Manual,
  http://www.altera.com/literature/manual/mnl_avalon_bus.pdf

- Nios UART Data Sheet,
  http://www.altera.com/literature/ds/ds_nios_uart.pdf

- Nios PIO Data Sheet,
  http://www.altera.com/literature/ds/ds_nios_pio.pdf

- Nios Timer Data Sheet,
  http://www.altera.com/literature/ds/ds_nios_timer.pdf

- GNUpro Users Guide,
  http://www.altera.com/literature/third-party/nios_gnupro.pdf

- Nios Embedded Processor Software Development Reference Manual
  http://www.altera.com/literature/manual/mnl_niossft.pdf

Exploring the SOCkit Reference Design

Your SOCkit board comes with the SOCkit reference design and Nios software stored in the EPCS4 flash device. After applying power to the board, the "MENU" and "SELECT" buttons can be used to select various board functions on the LCD display. To explore or modify the design further, you will need to open the reference design project files and software source code. These files are located on the SOCkit data files CD. This CD contains the latest archived Altera project file (.qar), and the C source code. These files should be copied to a project file in your PCs Altera Quartus II installation folder.

The SOCkit reference design blocks are coded in VHDL. VHDL is actually an acronym for an acronym! It stands for VHSIC Hardware Description Language where VHISC represents Very High Speed

Integrated Circuit. For the reference design, VHDL was used to code logic circuit "blocks", and then graphical representations of those blocks (.bdf files) were "stitched" together using Altera's graphical editor. This creates the top-level graphical ".bdf" design file (hierarchical design). Quartus II supports VHDL, Verilog, and AHDL (Altera HDL). Verilog and VHDL are more "mainstream" and both are standards in the design industry. AHDL is Altera's own HDL syntax, and is very similar to ABEL. AHDL is very simple to use and is still very powerful. AHDL is a good language to learn for those who are just getting started in FPGA design and HDL. The details of this reference design are in VHDL (and Verilog, as Verilog is the syntax that SOPC builder uses when generating design modules). Just keep in mind that our reference design will show you from a top level how to put things together. The details of implementation could be coded in any of the three HDLs (VHDL, Verilog, AHDL). The reference design will give examples of how to utilize all the major features of Altera's Cyclone FPGA, and will help you avoid many of the pitfalls encountered when "starting from scratch". The reference design utilizes all the hardware peripherals which are provided on the SOCkit PCB and provides:

- High speed LVDS pattern generation and detection with byte error rate.
- 32 bit Nios processor supporting interrupts, timer, UART, and PIO.
- Nios software image store to EPCS4 flash device (select by LCD menu).
- Configurable Nios software boot using GERMS RS-232 monitor or EPCS4 image copy to SRAM (configurable via dip-switch).
- Nios program execution from external 128K X 8 bit SRAM.
- EPCS4 nonvolatile flash data storage
- LCD menu operation
- LED test

Quartus II Tutorial Instructions

After installation of your Quartus II web edition CD, it is highly recommended that you go through the exercises contained in the Quartus II tutorial. The tutorial will help you to become comfortable with the QII 3.0 development environment.

Launching QII:

- Double click the Quartus II shortcut (icon) found on the desktop of your computer.
- Allow the computer to search the Altera site for updates. This will ensure that you have the latest copy of web edition.
- Under the Help menu on the top toolbar, select tutorial.
- At the top of the tutorial window push the Next button this will take you through tutorial basics and into the design sections.

Once you have completed the tutorial sections you will be ready to explore the SOCkit design.

Nios Shell Environment Basics

In order to build software loads for the Nios controller you will need to install the GNUPro C programming environment. This is an install option on the Quartus II web-pack CD that comes with the SOCkit (full kit).

A good place to start with the software is to first compile the existing C files that were provided with your SOCkit reference design. Before we get started, I would recommend that you download the following document, http://www.altera.com/literature/manual/mnl_niossft.pdf. Take a few minutes to scan over it. This document contains all of the commands necessary to build and run your software.

To begin, open a Nios SDK Shell. This shell can generally be found under /altera/excalibur/sopc_builder/"Nios SDK Shell" program directory. The shell will open a command line window. Using the command line, navigate to the SOCkit reference design directory and continue down into the SRC directory. This is located at "Your path here/cpu_sdk/src/sockit.c". This is the location for the Nios SRC files, so it is a convenient place to compile the SOCkit design file. Below is a few basic build commands and some options you might find helpful.

nb sockit.c

This command will compile the sockit.c design file. The output from the nb command is a .srec (S record) file. This file is then used to download into memory on the PCB when running GERMS

nr sockit.srec

This command is used to download your compiled .srec file using the Nios GERMS monitor. Following a successful download, GERMS will jump to the start of your software routines..

nr options

if you have trouble opening your com port or establishing a connection, here are a few options to try:

-p com2: This option instructs the nr command to use com port 2.

-b 19200: This option instructs the nr command to establish connection at 19200 baud. Any baud rate can be used i.e. 4800, 9600 56K etc…

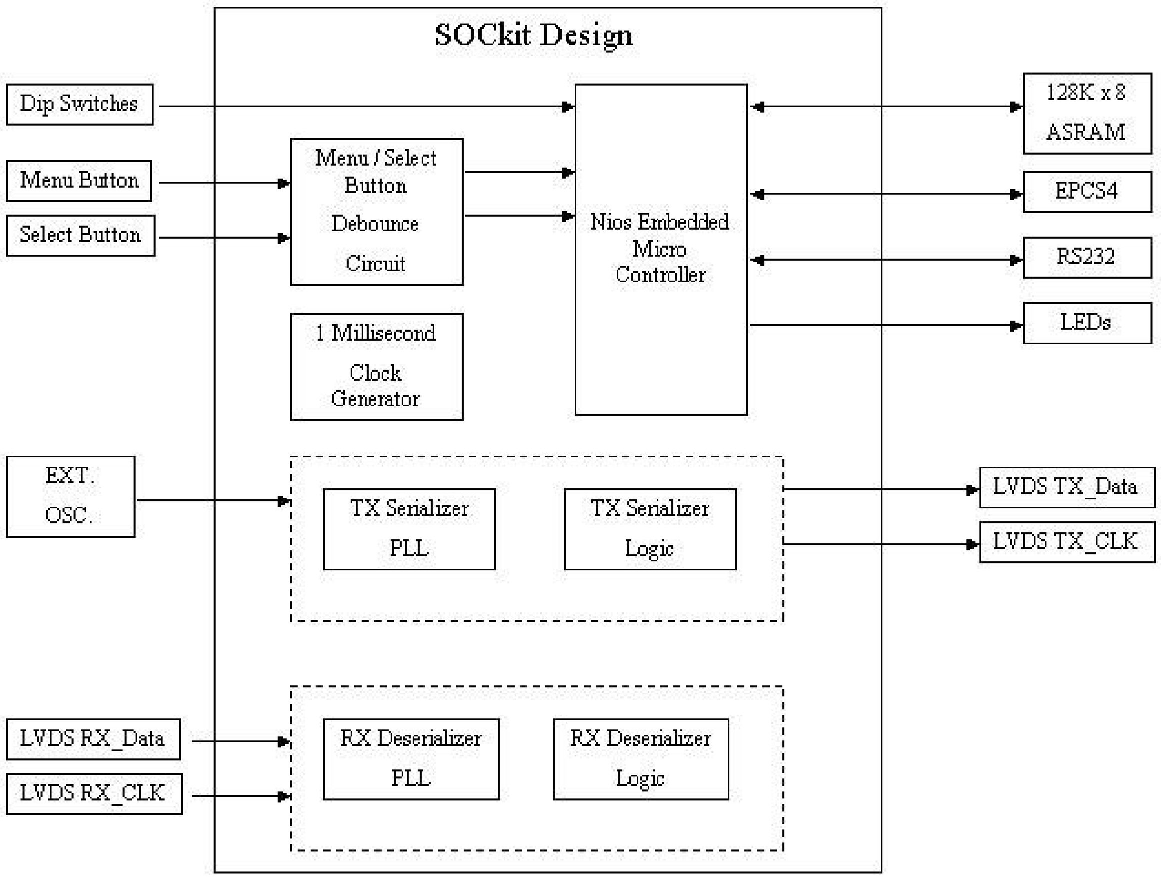
A typical nr command might be: **nr sockit.srec** (uses default com1 and 9600 baud), or the nr command could look like this: **nr –p com2 –b 19200 sockit.srec** (uses com2 and 19200 baud for download and communication).

Another important note regarding building loads. If you are embedding a software load in an internally defined ROM, you can find an example of this is boot.c which was embedded in the SOCkit reference design. There are two methods that can be used to do this. The first is by having SOPC builder actually compile your software. This is done by assigning the contents of the ROM to your.c file. When using this method SOPC will automatically build your.c file for the ROM location. The above method is the easiest and is less likely to create trouble.

The second method is to compile your.c file externally in a sdk shell using the nb command. Then when using SOPC Builder, assign the contents of the ROM to the .srec file that you just generated. This method works as well as the above approach, but care must be taken when creating the .srec file. When you build a .c file for a ROM you must tell the compiler where to place the code. Please refer back to the SOCkit design. The Boot ROM was located at address 0x1000. You can verify this by opening SOPC builder and looking at the "SYSTEM CONTENTS" window. When we created the ROM we assigned the contents to the boot.srec file. Now when SOPC generates the NIOS core, it assumes that the boot.srec file was built to the appropriate location. If this was not done and you are using the Boot ROM as your reset location, then your software will not execute correctly. The basic reason is that boot.srec is likely at 0x0000 and not 0x1000. To correctly build an internal ROMs contents using the SDK shell, you must explicitly tell the build command where to locate the code. In the case of boot.c, the build command looked like this: **nb –b 0x1000 boot.c**.

SOCkit Board Block Diagram

An overall block diagram of the SOCkit design is shown below.



The SOCkit design can be found in the top.bdf file of the SOCkit project directory. The SOCkit designs top level module was created using Altera's schematic editor. The schematic allows the user to create graphical block representations for sub-modules. These sub-modules can then be stitched together using "wires" for connectivity. The SOCkit Design is broken into three primary design sections, which are: Nios Softcore, LVDS Serializer, and LVDS Deserializer.

Nios Softcore and SOPC Builder

This section will describe the Nios softcore's connectivity to the rest of the design and the actual softcore's setup and configuration using SOPC Builder. The Nios block symbol can be found at the top middle of the SOCkit's top level schematic file (top.bdf). The Block symbol is labeled Nios, and like most schematic symbols inputs are shown on the left and outputs are shown on the right. Below is a description for each of the port pins found on the SOCkit's Nios module. All Nios ports are defined from the peripherals point of view (input/output). Keeping this in mind will help when adding your own module interfaces to a Nios softcore.

INPUTS:

Clk: This input is the primary clock for the Nios controller. The clock will establish the operating frequency for this design.

Reset_n: This is the active low reset signal that is used to reset the Nios softcore.

Data_from_the_byte_test_reg[31..0] : 32bit data-bus that is sourced from the byte_error rate test module. This bus allows the Nios softcore to read data information from the sub-module.

In_port_to_the_dipswitch_bank[3..0]: Inputs are directly connected to the bank of four dipswitches using PIO (programmable IO). The Nios softcore is able to read these switches and branch accordingly.

In_port_to_the_forward_button_pio: This input comes from one of the switch de-bounce modules. This input is directly connected to an interrupt and is used to branch the software when this button is pushed. The forward button is actually labeled MENU on your SOCkit board.

In_port_to_the_select_button_pio: This input comes from the other switch debounce module. This input is also directly connected to an interrupt and is used to branch the software when this button is pushed. The SELECT button is labeled on your SOCkit board.

Rxd_to_the_uart_1: Receive input from the external RS232 transceiver. This is connected to the Nios' internal UART.

OUTPUTS:

A_to_the_cy7c1019cv33_asram[16..0]: This output bus connects the Nios softcore's Avalon bus to the external Asynchronous SRAM address bus. This gives Nios full access to the SRAM 128K x 8 memory space.

D_to_and_from_the_cy7c1019cv33_asram[7..0]: This is the bi-directional data-bus connecting Nios to the external asynchronous SRAM.

Cs_n_to_the_ cy7c1019cv33_asram: External chip-select that allows Nios to select the Asynchronous SRAM .

Oe_n_to_the_ cy7c1019cv33_asram: External output enable that allows Nios to select a read operation to the Asynchronous SRAM

We_n_to_the_ cy7c1019cv33_asram: External write enable that allows Nios to select a write operation to the Asynchronous SRAM

Out_port_from_the_byte_test_en: This output port is used to enable and disable the byte error rate test.

Addr_to_the_byte_test_reg[3..0]: Address bus that is connected to the byte_error module. This bus allows Nios to address several internal registers for reading.

Nios_cs_to_the_byte_test_reg: Chip select is connected to the byte_error module. This allows the Nios to select the internal registers.

Nios_rd_to_the_byte_test_reg: Read enable is connected to the byte_error module. This allows the Nios to setup a read operation for the selected internal registers.

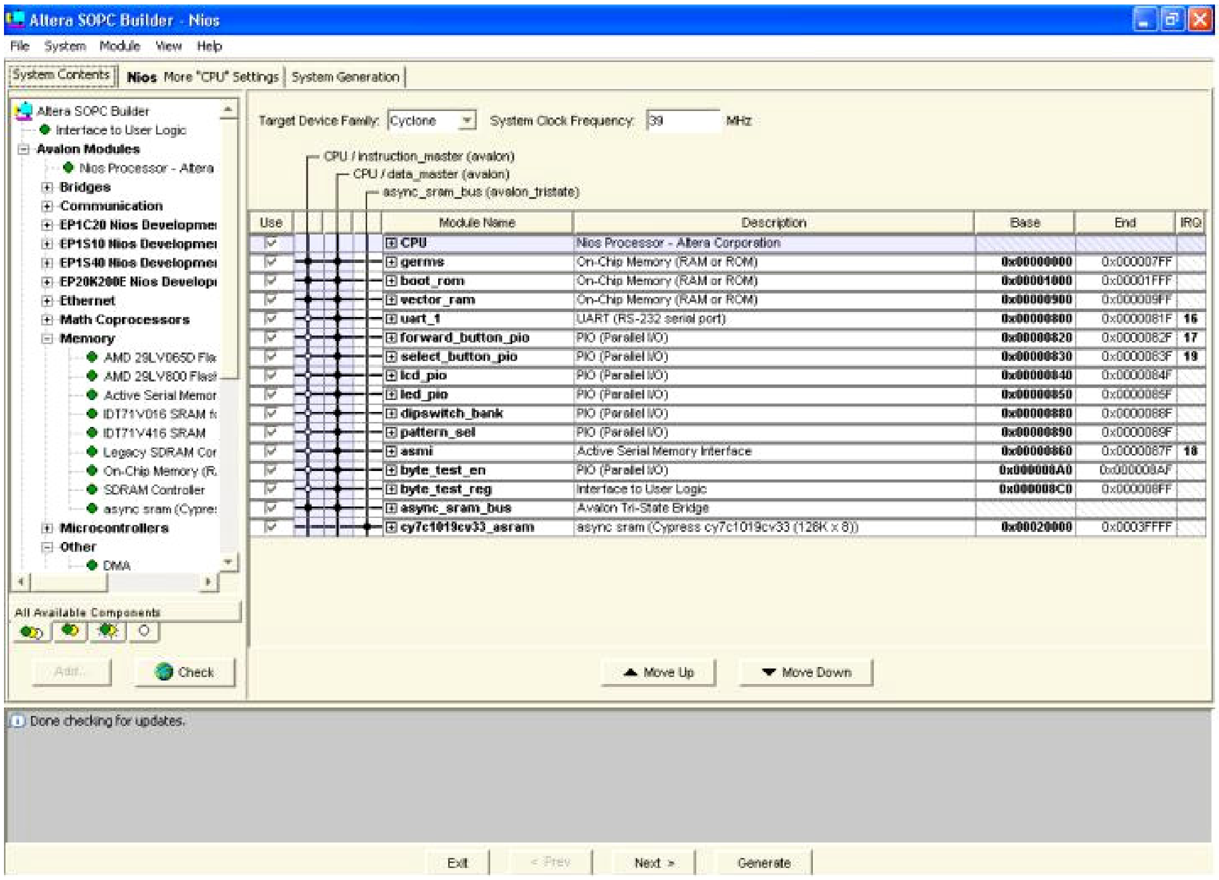
Bidir_port_to_and_from_the_lcd_pio[10..0]: External Bi-directional port that is used to communicate with the LCD display. The 10 bit bus is composed of 8 data bits, register selection control bit, Read/Write control bit, and an enable signal.

Out_port_from_the_led_pio[5..0]: These outputs control the six external LEDs. These outputs are used in a sinking manner. If a low is written to the port, the LED will illuminate. If a high is written to the port the LED will not illuminate.

Out_port_from_the_pattern_sel[1..0]: This output port is connected to the lvds_patterns module. This module allows the user to select between four unique LVDS output patterns. The selected pattern is then serialized and transmitted using the LVDS data outputs.

Txd_from_the_uart_1: Output is connected to the external RS232 transceiver's transmit pin.

In order to see the actual internal configuration for a Nios softcore, it is necessary to open it using SOPC Builder. This is done by double clicking the Nios block symbol located in the top.bdf design file. After double clicking the Nios block symbol, Quartus II will launch SOPC Builder. This will automatically bring the user to the first page of SOPC builder which is labeled "SYSTEM CONTENTS". Below is a screen shot of SOPC's "SYSTEM CONTENTS" page:



Target Device Family: Cyclone System Clock Frequency: 39 MHz

CPU / instruction_master (avalon)
CPU / data_master (avalon)
async_sram_bus (avalon_tristate)

Use	Module Name	Description	Base	End	IRQ
✓	CPU	Nios Processor - Altera Corporation			
✓	germs	On-Chip Memory (RAM or ROM)	0x00000000	0x000007FF	
✓	boot_rom	On-Chip Memory (RAM or ROM)	0x00001000	0x00001FFF	
✓	vector_ram	On-Chip Memory (RAM or ROM)	0x00000900	0x000009FF	
✓	uart_1	UART (RS-232 serial port)	0x00000800	0x0000081F	16
✓	forward_button_pio	PIO (Parallel I/O)	0x00000820	0x0000082F	17
✓	select_button_pio	PIO (Parallel I/O)	0x00000830	0x0000083F	19
✓	lcd_pio	PIO (Parallel I/O)	0x00000840	0x0000084F	
✓	led_pio	PIO (Parallel I/O)	0x00000850	0x0000085F	
✓	dipswitch_bank	PIO (Parallel I/O)	0x00000880	0x0000088F	
✓	pattern_sel	PIO (Parallel I/O)	0x00000890	0x0000089F	
✓	asmi	Active Serial Memory Interface	0x00000860	0x0000087F	18
✓	byte_test_en	PIO (Parallel I/O)	0x000008A0	0x000008AF	
✓	byte_test_reg	Interface to User Logic	0x000008C0	0x000008FF	
✓	async_sram_bus	Avalon Tri-State Bridge			
✓	cy7c1019cv33_asram	async sram (Cypress cy7c1019cv33 (128K x 8))	0x00020000	0x0003FFFF	

Done checking for updates.

This window is used to add all Nios interfaces, and allows the user to select the targeted device family, i.e. Cyclone, Stratix etc. Also found on this window is the system clock frequency. This needs to be set to the same frequency as the clock that will be input to the Nios modules clock input. This is particularly important when using the Nios internal UART, as this clock will be used to establish the count values necessary to configure the RS232 baud-rate. The "SYSTEM CONTENTS" page lists an entry for each component found in the Nios softcore. Next to each entry is the assigned memory space and IRQ (if applicable). The left side of the screen contains a list of peripherals that can be added to your design. These "ready to use" modules include UARTs, PIOs, memory interfaces etc. A basic set of the modules have been included with SOPC, however, other modules may have to be purchased as IP (Intellectual Prorperty).

Most of these modules reside in the SOPC installation directory, and are Altera ".ptf" files. Most basic interfaces can be generated using the "INTERFACE TO USER LOGIC" SOPC wizard located at the left of the screen. Sometimes, for advanced functions, .ptf files have to be generated manually. For more information on .ptf files please refer to the following document:

http://www.altera.com/literature/manual/mnl_sopcptf.pdf

One last item to note is the "matrix" type format of the main SOPC builder screen. Nios basic components (instruction master, bus master, Avalon bus) are shown across the top of the screen. These basic Nios architectural components are connected to internal (to Nios block) and external peripherals by "closing the dot" on the column/row matrix that comprises this main SOPC builder screen.

The second window in SOPC builder is labeled "NIOS MORE CPU SETTINGS". This window is used to define where the Nios softcore will begin execution and what memories will be used for different tasks. These tasks include interrupt vector storage, data memory and program memory. This window is also used to define UART assignments and Reset location. A system Boot ID can also be set here. This Boot id will be displayed at power-up via GERMS and an RS232 monitor if one is present.

The third and final window found in SOPC Builder is the "SYSTEM GENERATION" window. This window allows the user to select file generation options for the Nios build. Here the user can select to have header files and libraries generated during the build. Other selections include a HDL output option and a simulation option.

Once you have completed your selections using the three SOPC windows, select the generate button and a new Nios Softcore will be created. For more information regarding the use of SOPC builder please refer to the following hyperlink:

http://www.altera.com/products/software/system/products/sopc/sop-index.html

Nios Memory Organization

The memory structure found in the SOCkit design was chosen to allow system branching based on user inputs. This section will define the memories used in the SOCkit design. Refer to the actual design files for details.

SOCkit Memory Map:

Module Name	**Address Range**	**Assigned IRQ**
GERMS (monitor) ROM	0x00000000 – 0x000007FF	
Boot ROM	0x00001000 – 0x00001FFF	
Vector RAM	0x00000900– 0x000009FF	
UART	0x00000800 – 0x0000081F	16
Menu PushButton	0x00000820 – 0x0000082F	17
Select Push Button	0x00000830 – 0x0000083F	19
LCD Interface	0x00000840 – 0x0000084F	
LED interface	0x00000850– 0x0000085F	

Dip-switches	0x00000880 – 0x0000088F	
Pattern Select outputs	0x00000890 – 0x0000089F	
ASMI (Flash Interface)	0x00000860 – 0x0000087F	18
Byte test enable control	0x000008A0 – 0x000008AF	
Byte Error Interface	0x000008C0 – 0x000008FF	
Async. SRAM Interface	0x00020000 – 0x0003FFFF	

The SOCkit design uses three internal memories, and two external memories. The three internal memories are:

GERMS (monitor) ROM: The monitor ROM is an internal memory that stores Altera's GERMS ROM monitor. This monitor program provides basic memory peeks/pokes, and to operates the RS232 terminal. Any internally defined RAM/ROM can be initialized as a GERMS ROM using SOPC builder. To add GERMS, select the GERMS button under the "CONTENTS" window of the on-chip memory module (during the adding of the memory to the SOPC design, or afterwards by double-clicking on the RAM/ROM entry).

Boot ROM: This internal ROM is located at the RESET location of the SOCkit design. The source file for this ROM can be found in the SOCkit project directory /cpu_sdk/src/boot.c. This file will either load the user program from the EPCS4 flash device or will branch to GERMS. This selection is determined by the setting of S3 dipswitch 1. If dipswitch 1 is set to "0" (on position), the design will branch to GERMS, thus allowing user debug or program download. If dipswitch 1 is set to "1" (off position), the SOCkit design will load its program memory using the design stored in the flash device. Loading the Flash memory with user programs will be discussed later in the booter section of this manual.

Vector RAM: This internal ram is used for the interrupt vector table of the Nios Softcore.

ASMI (Flash Interface): This external flash is unique to Altera's Cyclone family. The EPC device found on the SOCkit design is a 4Mbit device (4,194,304 bits). This device uses a dedicated FPGA serial interface for communication with the Nios softcore. The EPCS4 is not only used for Cyclone device configuration, but can also be used for program store (user data).

Asynchronous SRAM Interface: This external memory is used for both program and data store. This memory can be loaded at boot-up by the boot ROM if dipswitch 1 is set to the off position. The user program is transferred from the EPCS4 to the SRAM and then executed. It is possible to use the Cyclone FPGA's own internal memory for program and data store, however, the user will have to be very diligent not to exceed the available memory size, especially when coding in C.

Actual program data size is very important, especially when working with smaller internal memories. The boot ROM executes a small piece of code at power-up. You, the designer must define this memory as a ROM. ROMs are always defined as initializable memories, meaning the actual FPGA load provides the memories initial contents. It has been observed that the smallest S record size will be produced by the Nios assembler. The Gnu C compiler tends to produce much larger record sizes, and therefore requires larger memory devices.

LVDS Serializer

The LVDS Serializer found on this reference design is a rather simple approach to data transmission. The basis of the LVDS design is a separate data and clock channel utilizing the two Cyclone EP1C6 PLLs and a reference input frequency of 24 MHz.

There are two version of the SOCkit reference design. The difference between these two versions is the speed grade of the Cyclone EP1C6. The speed grades available for the SOCkit designs are the –6 and –8 FPGA devices. The –6 is a faster device. For details regarding the actual timing differences please refer to Altera's **Cyclone Handbook** Chapter 4.

Below is a table showing the differences between the two design PLL configurations per the devices speed grade. The input clock frequency for both design versions is 24 MHz.

EP1C6 Speed Grade:	**-6**	**-8**
System PLL Multiply/Divide Ratio and resulting frequency	13/8 = 39MHz.	4/3 = 32MHz.
Data PLL Multiply/Divide Ratio and resulting frequency	13/1 = 312MHz.	32/3 = 256MHz.

For the –6 version of the SOCkit development board, the data transfer PLL is configured for a 13/1 multiplier, which results in a 312 MHz serializer clock. Our design works on byte boundaries and as a result the data transfer serializer has to run at eight times our system clock. The system clock is generated from a 13/8 multiplier which results in a system frequency of 39 MHz. Thus the serialized data rate for our –6 reference design is 312 Mbs.

For the –8 version of the SOCkit development board the data transfer PLL is configured for a 32/3 multiplier, which results in a 256 MHz serializer clock. The system clock is generated from a 4/3 multiplier which results in a system frequency of 32 MHz. Thus the serialized data transfer rate for our –8 reference design is 256 Mbs.

In the reference design the serializer circuit is located at the bottom of the top.bdf schematic file. The entire circuit is comprised of the TX PLL (discussed above), lvds_patterns (VHDL module) and a five stage pipeline. The lvds_patterns module contains an eight bit shift register, a pattern generation state machine, and a synchronization circuit. The pattern generation state machine operates at the system clock frequency. This state machine is controlled by the Nios softcore. The state machine input is labeled pattern_sel[1..0] in the top.bdf file. These inputs select the output data pattern the state machine will generate. The table below shows the output state (sequences repeatedly from 0 to 7), pattern_sel inputs, and the corresponding output pattern.

Lvds_patterns Output State	**Pattern_Sel Setting**			
	00	**01**	**10**	**11**
0	00	00	00	55
1	00	01	55	55
2	00	02	AA	55
3	00	03	FF	55
4	00	04	00	55
5	00	05	55	55
6	00	06	AA	55
7	00	07	FF	55

You can see that when the control inputs are set to 00, the LVDS pattern output will be continuous zero. When the control inputs are set to 01, the LVDS pattern output is a repeating count of zero through seven. When the control inputs are set to 10, the LVDS output pattern is a repeating pattern of 00, 55, AA, FF. This pattern is also used to run the byte error rate test (covered in a later section). When pattern_sel is set to 11, the "55" pattern gives us a constant alternating 0 to 1 transition on our high speed serial data output (01010101).

The synchronization circuit was added to insure that the x8 load counter was synchronized to the slower system clock. This circuit operates as a one shot which becomes true only after the first rising edge transition of the system clock. After the one shot becomes true an internal counter will begin running. This counter operates at the faster x8 frequency. Its function is to count eight cycles of the x8 clock, reset to zero and issue the load signal. The load signal drives the load enable on the shift register. This mechanism places the selected pattern data in the shift register where it is then serialized at the high frequency data transfer clock rate.

Now that the selected pattern data has been serialized, it is necessary to send this data through an output pipeline. The pipeline was used to make sure the transmit circuit operates on an even byte boundary. The pipeline ques the x8 serialized output so that data transmission starts on an even byte boundary right at the transition of the slower system clock. This is necessary, to insure that the high speed data receiver samples data on a system clock transition boundary (provides byte aligned data at the slower system clock speed).

LVDS De-serializer and Byte Error Test

Like the serializer, the de-serializer circuit is labeled and easily identified in the reference design. The circuit contains a PLL, shift register, 2 to 1 multiplexer, and the byte_error vhdl module. The receive PLL is configured to generate a x8 multiple of it's input clock. The de-serializer PLL input clock is connected via the fire-wire patch cable to the serializer transmit clock. This clock has a frequency of 39 or 32 Mhz, depending on the Cyclone FPGA speed grade. Using the slower frequency clock as the transmit clock allows transmission of a slower clock, which is then multiplied up again on the receive end. Thus based on our input frequency, the x8 frequency on the receive end will be either 312 MHz or 256MHz.

At the input of our de-serializer design we have placed a single flip flop to register the high-speed serial data. We have also turned on the Quartus II "fast input register" constraint for this input. This will place the initial input register in the IOB of the FPGA. Doing this allows us to register the input right at the edge of the FPGA and thus increases our setup time margin. Once data has been clocked into the de-serializer block, the shift_register megafunction begins shifting serial data in and then presenting byte aligned data to the parallel_reg megafunction. The byte wide interface runs at our slower system clock frequency. From this point on, in our design, byte wide data (slower data rate) is presented to the lvds_mux multiplexer megafunction. The multiplexer has two byte wide inputs, one input is our de-serialized data. The other has been fixed to VCC. The purpose for the multiplexer is to allow generation of test errors for the byte_error module. This will be discussed in more detail later. From the multiplexer data is presented to the byte_error VHDL module.

The **byte_error vhdl** module was designed to detect errors found on the serialized LVDS data path. For the purposes of this reference design the only pattern supported for the byte error rate test is the pattern "00", "55", "AA", "FF" which repeats. By using a compare state machine, the "00" pattern is the test starting point and then subsequent patterns are read in and compared. If the pattern deviates from the expected pattern, the byte_error counter is updated. Otherwise, the byte counter is updated (no error). In this manner we are able to report total bytes received, as well as total errors detected. Reporting is implemented through the Nios interface connected to the error detection module. Several inputs control the operation of the byte_error module. One of these is the byte_test_en input which controls how the state machine will operate. If this input is zero the state machine will be held in the idle state. Otherwise, the machine will be processing byte errors. The other input to this module controls the multiplexer inputting data to the module. This input comes from dip-switch S3 switch[4]. If this switch is set to zero (on), receive data is presented to the byte_error module. If this switch is set to one (off), data presented to the byte_error module will be stuck high (VCC). This method is used to introduce byte errors into our test to verify proper operation. To use this function, you must first start the test with S3 switch[4] in the on position to allow synchronization, and then to inject errors, turn switch[4] off.

SOCkit Software Description

The best way to investigate the SOCkit software structure is to view the source code. For the benefit of new programmers that are learning, the following program flow description paragraphs will clarify things a bit. Refer to the actual sockit.c source code while reading these sections.

The Majority of the LCD functions used in the SOCkit reference design are provided in Altera's Nios Development Kit (Cyclone edition). These functions have been included in the SOCkit design files. The Nios core used in the reference design is not time limited, however, subsequent recompiles of this design using the web edition tools will result in a time limited version of the Nios core (web-pack has trial version of SOPC builder).

The Software reference design that was included with your SOCkit development board can be found in "CPU_sdk\src" directory. The source file is the sockit.c file. This is the main file for the reference design.

The following section will step through the main structure of the sockit.c design file (for the benefit of student programmers).

The top of the file contains include statements, function declarations, and variable declaration. Scroll down to the comment "**Message to germs**". The two printf statements following this comment are the first Output of the SOCkit software design. To see these comments displayed, open a Nios SDK shell and type nr –t. If you have your PC RS232 cable connected to the 9pin D-Sub on the PCB, you should see the comments directly after power-up or following assertion of the RESET button. The next two sets of instructions setup the MENU and SELECT pushbuttons for interrupts. Continue down to the comment "**Design Starts Here**". The first function call is to initialize the LCD. Next we begin displaying our initial message. This portion of the code displays a static "Dallas Logic" on the top line of the 16x2 LCD, while the bottom line operates like a marquee, scrolling a message across. This portion of code will run continuously until the MENU pushbutton is pressed. Once the MENU button is pressed the LCD will display the first line of our main menu. Scroll down to the comment " **Main Menu Starts Here**". The menu is made up of four entries which include: LVDS Pattern Selection, LVDS Byte Error Test, LED Test, and Flash RAM to EPC. These entries are grouped in a single while loop that runs continuously. The menu can be advanced using the MENU pushbutton. As the menu is advanced the LCD will update with the new menu selection. To run a given menu selection, press the SELECT pushbutton. This will call the function contained in the menu entry. Once completed the function will return to the entry called.

The menu was created using a case statement with four entries. The MENU button using an ISR advances through the menu entries. The forward_isr is located immediately following our menu while loop. The forward_isr is the MENU button's interrupt service routine. This ISR works by updating the screen variable that controls the menu's case statement. The ISR updates this variable with every press of the MENU button. It has also been designed to jump back to the first entry if the MENU button is pressed on the fourth entry. If the first entry in the menu is selected, the forward_isr branches to a secondary variable update. This second variable update routine allows the user to advance through the LVDS Pattern Selection menu. The sub-menu entries include: "All Zeros", "0 1 2 3 4 5 6 7", "00 55 aa ff", and "55". Like the previous menu pressing the SELECT button will select a data pattern and return back to the previous menu entry. Note: the program will remain in the forward_isr until a selection is made.

Immediately following the forward_isr is the select_isr. The select_isr is mapped to the SELECT pushbutton found on the SOCkit design. Every time the SELECT button is pushed the Nios controller will automatically call this interrupt service routine. The select_isr works by setting a global flag that indicates the button was pushed. It is then up to each function to branch accordingly and clear the global flag when done.

Below the select_isr you will find three additional LCD control functions. These were added to expand on Altera's LCD functions included in the Nios Development Kit Cyclone edition. The three functions are listed below with a brief explanation of each:

nr_pio_lcd_marquee – This function sets up a scrolling message across the second line of the 16 x 2 LCD. This function is used at power-up and the start of the SOCkit software run. Immediately following reset or power-up you can see the function scrolling on the LCD.

nr_pio_lcdwritescreen_top – This function only writes to the first line of the 16 x 2 LCD. Any characters that exceed the display length are cut off. The SOCkit's LCD display length is sixteen characters.

nr_pio_lcdwritescreen_bot – This function only writes to the second line of the 16 x 2 LCD. Any characters that exceed the display length are cut off.

Following the additional LCD functions is the led_test function. This function is called when the fourth entry in the Main LCD menu is selected. This function repeatedly illuminates each LED until the MENU button is pressed again.

The lvds_test function is a branch function from the main menu's second selection "LVDS Pattern Selection". Once executing in the lvds_test function, the user must select one of the patterns. Pattern selection is done by using the MENU button to advance the selections. Once the pattern to be used is displayed, use the SELECT button to begin transmitting that pattern. Like the Main Menu section described above, the lvds_test menu interrupt operation operates the same way. That is the forward_isr updates the lvds_screen variable. This controls which of the lvds_test selections is displayed on the LCD. The test patterns were defined to allow the user to view a variety of data from the de-serializers data output bus. This bus can be viewed using a Logic Analyzer. Once a pattern is selected the function returns to the main menu.

The byte_test function is used to report the byte error test results that are being generated in the byte_error VHDL module. The byte_error VHDL module has a Nios interface which allows byte_test to read the update registers and report them using the LCD. The function will continue to run until the MENU button is pressed.

The last function used in the SOCkit design is flash_epc. This function was designed to write the contents of the onboard 128Kx8 Asynchronous SRAM to the EPCS4 serial flash device. This function is designed to use the Altera's ASMI **`nr_asmi_write_buffer`** routine to write 256 byte blocks, 512 times to the EPCS4. The target location in the EPCS4 is the two highest 64K flash blocks. This function is called from the main menu under the selection "Flash RAM to EPC". Once selected the last two flash blocks are erased, then the entire 128K x 8 external SRAM contents will be copied to these two flash sectors. On the next PCB power-up, the new user program will automatically be copied to 128K x * SRAM from flash and executed. This copy routine will be discussed in the Boot program section below.

Boot Flash to SRAM Copy Program

The Cyclone EP1C6 has 11.52 Kbytes of total ram. For small embedded applications this RAM size maybe sufficient. However, for applications requiring larger program storage, it becomes necessary to add external memory storage. What the SOCkit design example emphasizes is a lower cost FPGA coupled with an external SRAM for larger program store and a simple way to load it. The SOCkit design uses Altera's EPCS4 flash configuration device. The EP1C6 (Cyclone FPGA) requires a configuration size of 1.16 Mbits. By subtracting the configuration size from the flash device size, we end up with 3.02 Mbits. After dividing this by eight (for bytes), the flash device has approximately 378K bytes remaining for our data use. Altera has designed several functions that allow users to access this spare flash space. It is in this flash space that the SOCkit design stores its program and/or data. In the previous section, we discussed the flash_epc function. This function is used to write the contents of RAM to Flash. The Boot program we now discuss performs the opposite function. Its purpose is to copy from Flash to external 128K x 8 SRAM (usually at initialization).

The boot.c reference program that was included with your SOCkit development board can be found in the CPU_sdk\src directory. The source file we will review is the boot.c file. Boot.c is the first program that Nios executes after power-up or reset. If you open SOPC builder and review the "MORE NIOS SETTINGS" window, you will see that our RESET location is defined to be memory location 0x1000. Now open the "SYSTEM CONTENTS" window, and you can see that the Boot ROM is located at address 0x1000. One more item to note in the "SYSTEM CONTENTS" window is that the GERMS ROM is located at address 0x0000. By placing the GERMS routine into one of the onboard memories, Boot.c will be able to jump to GERMS if a user download is required (selected using dip-switch).

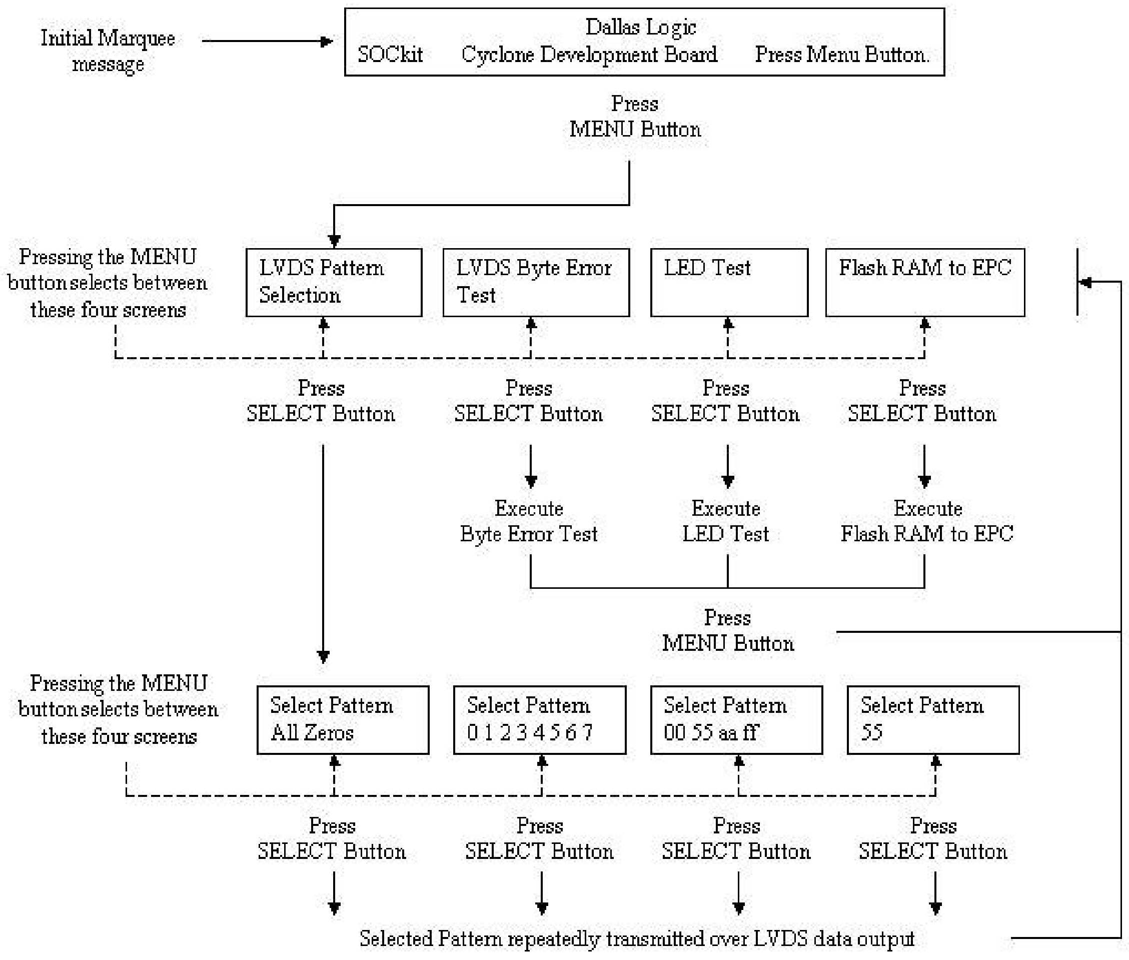
The S3 dip-switch setting will determine if the boot.c flash copy routine will execute or if we jump to GERMS. If the dipswitch is placed in the "on" position, the boot sequence will jump to load GERMS. If the dipswitch is placed in the "off" position the flash copy routine will be executed. Once the copy is completed, program execution will begin at the first location of the external SRAM..

Boot.c can provide an initial boot to GERMS to perform a program download. Once execution begins the sockit.c program provides a "RAM to EPC" function that will copy the downloaded program stored in external SRAM (itself) to the EPCS4 flash. This provides for permanent storage of a user software program in flash. Future versions of Quartus II will hopefully support integrating of FPGA initialization images with software executables directly into the EPCS4 flash device. This is normally done with the Quartus II "CONVERT PROGRAMMING FILES" menu selection. The EPCS4 is not yet supported by this function.

The flash copy routine uses Altera's ASMI **nr_asmi_read_buffer** function. This function is designed to read from the EPCS4 in a user defined block size. In the Case of the boot.c file, a 128 byte buffer size is used. The copy function stays in the while loop until the prom_end address is reached. Once the prom_end address is reached, the program drops to a small embedded assembler routine that jumps to and starts executing from SRAM.

LCD Menu Operation Flowchart

Following power-up or Reset, the SOCkit design will begin running the initial marquee message. Pressing the MENU button will advance to the main menu. Below is a flow diagram for the LCD menu system.



The main menu consists of four menu entries, LVDS Pattern Selection, LVDS Byte Error Test, LED Test and Flash RAM to EPC. Pressing the MENU button will advance through these menu entries. To select one, press the SELECT button while the entry is displayed. In the case of the LVDS Byte Error Test, LED Test, and Flash RAM to EPC, pressing the SELECT button will execute the displayed function. The user will then have to press the MENU button in order to return to the main menu entries. In the case of LVDS Pattern Selection, pressing the SELECT button will advance the user to the Pattern Selection sub-menu. There are four sub-menu entries. They are: Select Pattern All Zeros, Select Pattern 0 1 2 3 4 5 6 7, Select Pattern 00 55 AA FF, and Select Pattern 55. Pressing the MENU button will advance through these menu entries. To select one, press the SELECT button while the entry is displayed. This will begin transmitting the selected pattern continuously to the LVDS data output. Once a pattern has been selected the user will be returned to the main menu. The user is always returned to the entry in the main menu that was originally selected. Modifying the FPGA pin definitions and then initiating a Quartus II compilation changes which pins, and thus the associated header the LVDS patterns will be output to and received on.

Reference Design Schematic

The SOCkit schematic should be referenced for details on circuit design or when connecting devices to the supplied pin-headers. The schematics .pdf pages are located on your SOCkit reference design and data files CD.

SOCkit Board Technical Help

Being an "internet-centric" company, Dallas Logic provides manned 24hr email and web forum support. The best way to obtain help with problems is to post questions to our online support forum. We are prompt about replying, and there is also a good chance you will find your question has already been answered online in the forum. If you prefer email support, we can be reached at support@dallaslogic.com. We will generally reply within 24hrs of receiving an inquiry or request for help with problems which are related to the operation of your SOCkit board. Before contacting us with inquiries, please make sure you have:

1. Verified the fuse (F1) and checked the voltage outputs at connector J1.

2. Removed any test circuits from connection to the SOCkit PCB and restored it to its original design state (remove and restore any modifications).

3. Re-programmed and tested your SOCkit board with the original reference design files.

For questions related specifically to Altera software tools or FPGA devices, we can sometimes be of assistance, but will generally refer you to the Altera online knowledge base or online support web pages.

Warranty Information

Your un-modified SOCkit assembly is guaranteed to be free of defects in material and workmanship for a period of ninety days from the date of purchase. If your SOCkit board stops working during the ninety day period, and is in its original, un-modified state, first contact us at support@dallaslogic.com. If the problem cannot be resolved, you must return the PCB assembly and power supply, postage prepaid to Dallas Logic Corporation. All repairs and return ship will be made within 12 days of receipt of package. During the warranty period, Dallas Logic will, at its option, repair, replace, or refund the purchase price. Products that have been damaged or modified from their original design state are not covered by warranty. No warranty is implied with respect to the software or firmware files.

Disclaimer