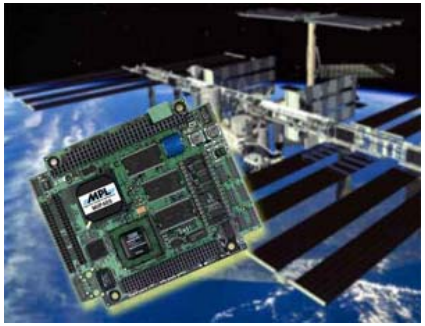
CSCE 313: Embedded System Design

Introduction

Instructor: Jason D. Bakos

UNIVERSITY OF SOUTH CAROLINA

Embedded Systems



Desktop vs. Embedded CPU

- General purpose (desktop and server) CPUs have a more complex structure
  - Execute up to 4 instructions per cycle
  - Instructions executed out-of-order
  - Big, complex caches
  - All code runs at the same speed, since the processor finds parallelism at runtime
    - Exception: single core code will still only use one core, SIMD instructions often require "intrinsics"

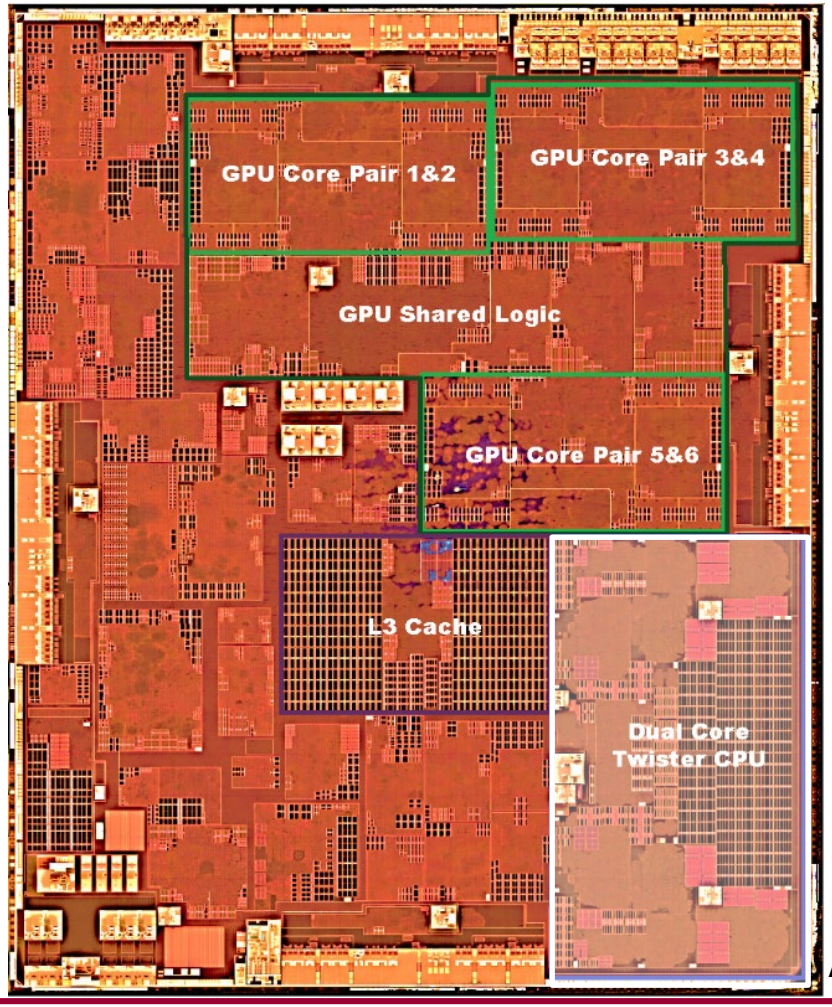
- Embedded CPUs:
  - Generally not reprogrammable except by vendor
  - One or two instructions per cycle
  - Instructions executed in order, so instructions that depend on previous ones will "hold up the line"
  - Small, simple caches
  - Performance is dependent on code efficiency
  - Sometimes do not run an OS ("bare metal") or limited OS support
  - Tightly coupled with peripherials (system-on-chip)

Desktop vs. Embedded CPU

- This class vs. 240 and 274:
  - Write code in C (vs. Java)
  - Write and debug code on a PC that runs on different processor
  - Write code that communicates with hardware

- This class vs. 274:
  - Write code in C (vs. Python)
  - Write code that runs on bare metal
  - Write code containing features to improve performance
  - Code is more performance- and graphics-oriented

System-on-a-Chip

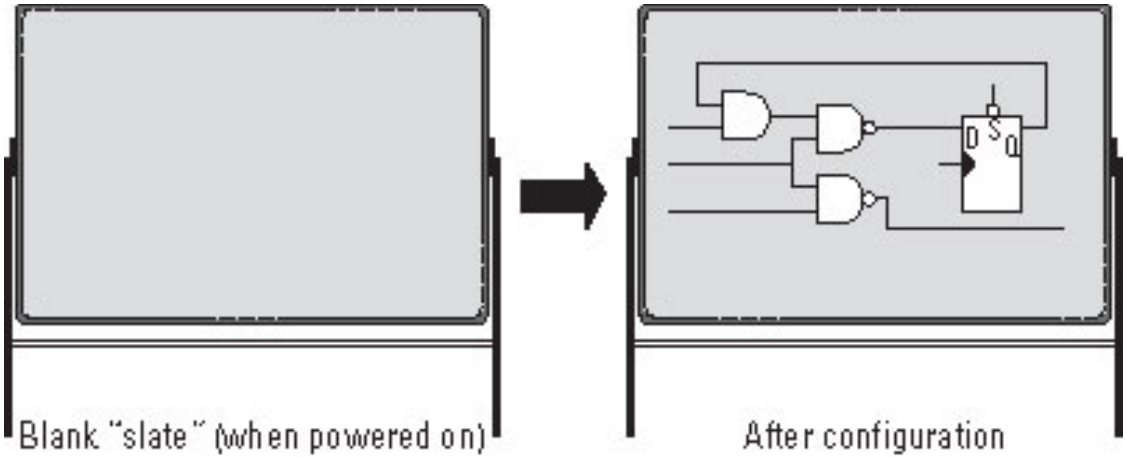


Apple A9

- Most embedded processors contain multiple CPUs and integrated peripherals:
  1. I/O
  2. Coprocessors
  3. Memory

Field Programmable Gate Arrays

- Programmable logic device



- Contains:
  - Ability to implement "soft logic": programmable logic gates (CLBs) with programmable interconnect
  - "Hard cores": RAMs, multipliers, IOs, PCIe interface, etc.

Field Programmable Gate Arrays

- Originally developed for "glue logic"

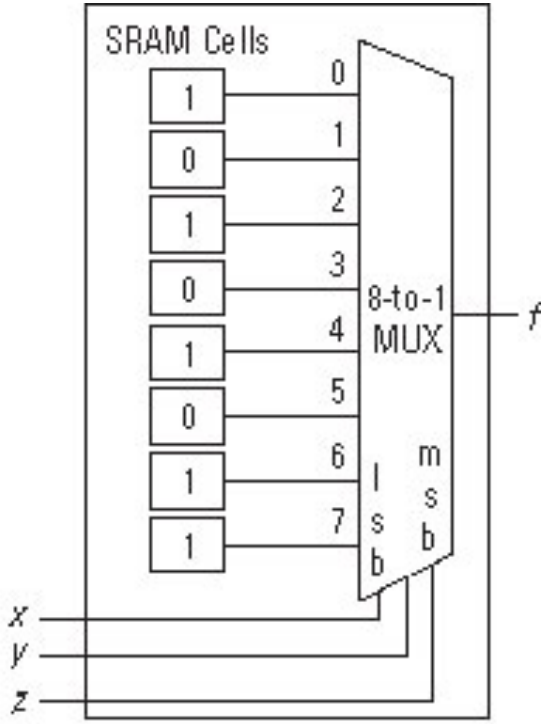
- Now used as system-on a-programmable chip (SoPC)
  - Customized "softcore" processor,
  - Memory/cache subsystem,
  - I/O interfaces,
  - Off-chip memory interfaces

FPGA Lookup Table

- Function generator:

x	y	z	$xy + z'$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

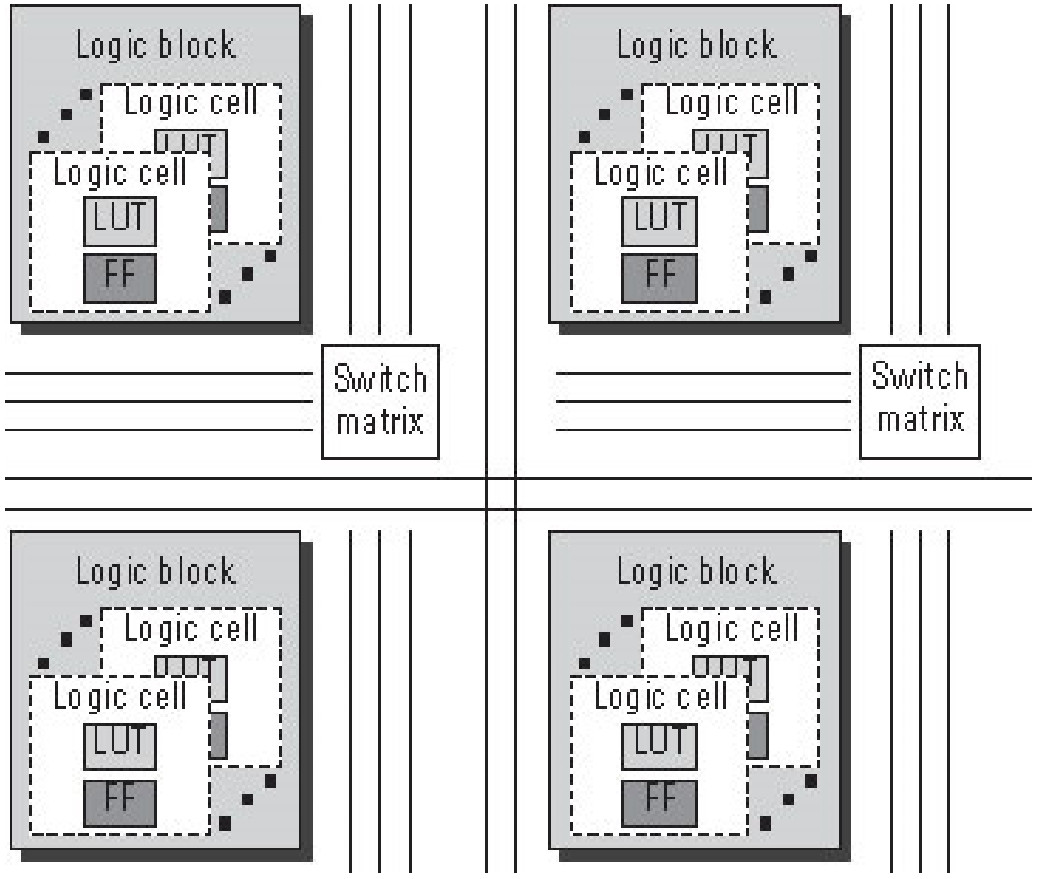
(a)



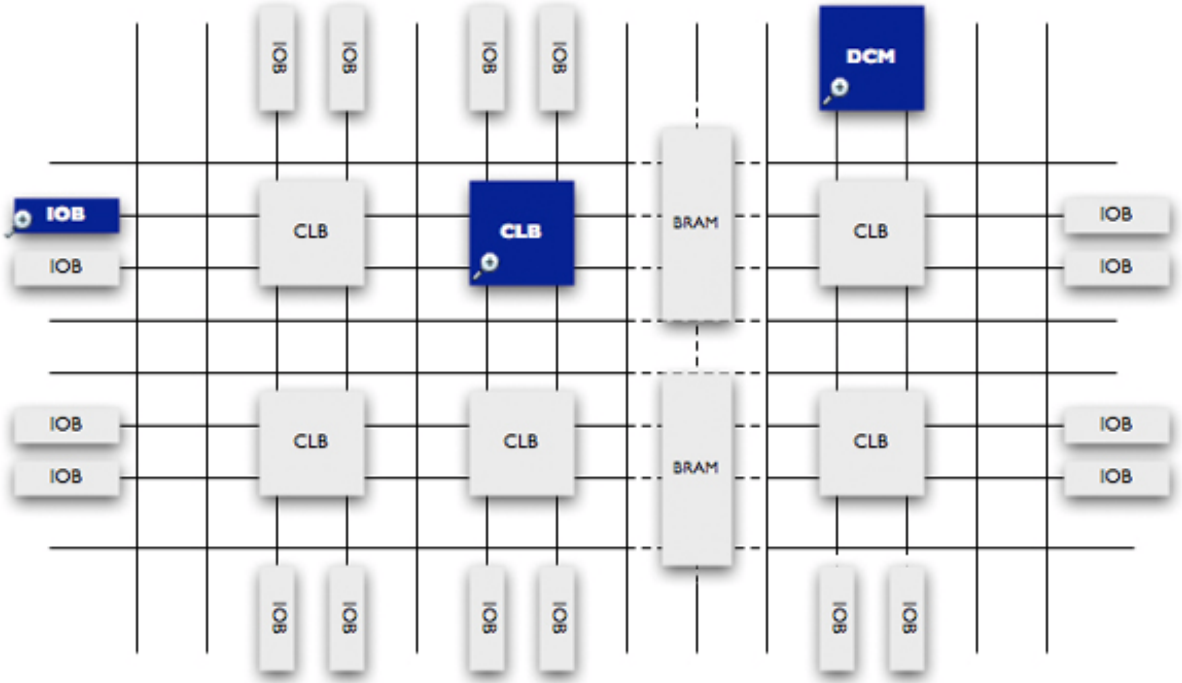
(b)

FPGA Fabric

- FPGA fabric:

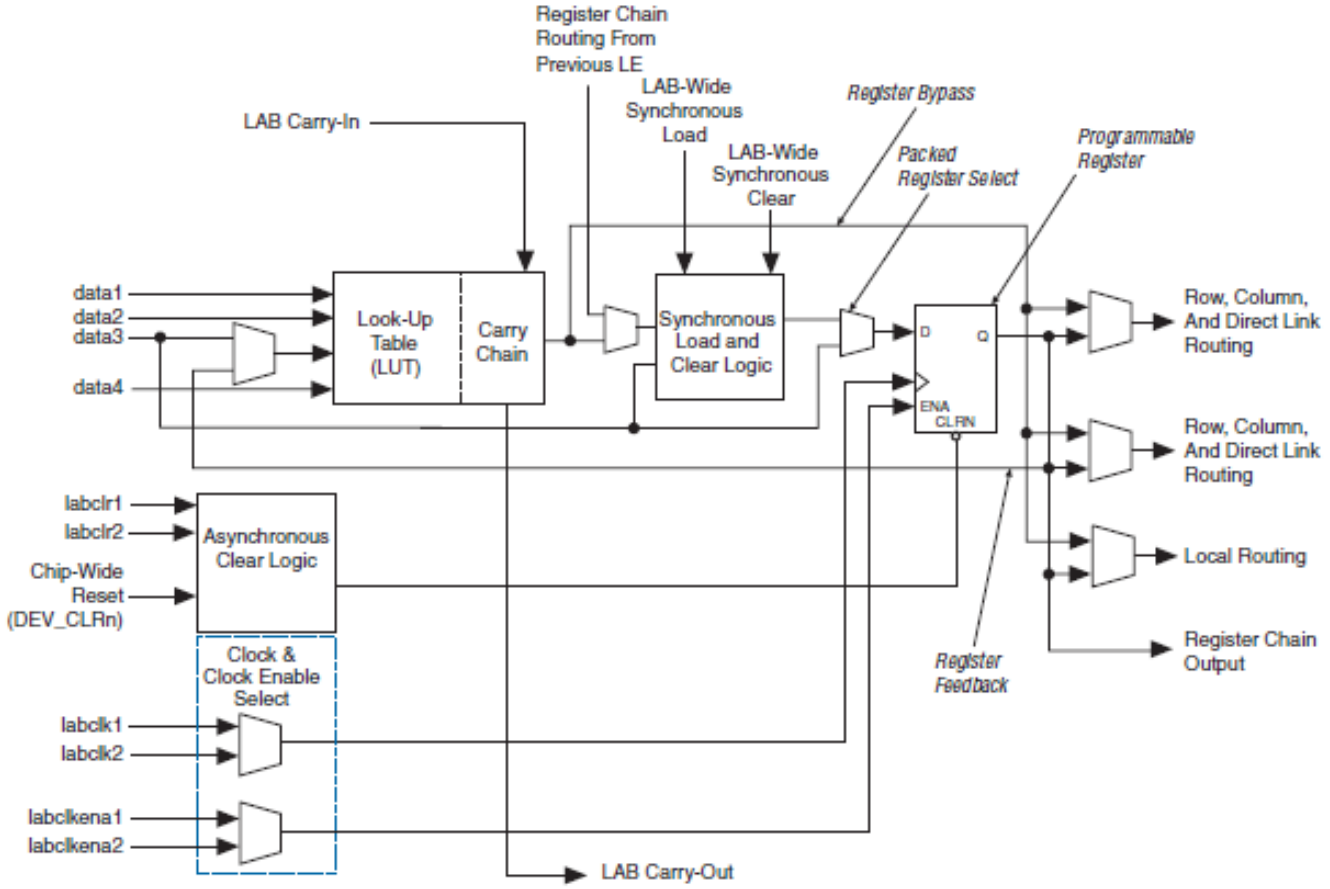


Field Programmable Gate Arrays



- On chip resources:
  - Logic Elements (LEs)
    1. LUT
    2. Register
  - Onchip memories (M20Ks)
  - Multlipliers
  - PLLs

Cyclone 2 Logic Element



Verilog Example

- Full adder:

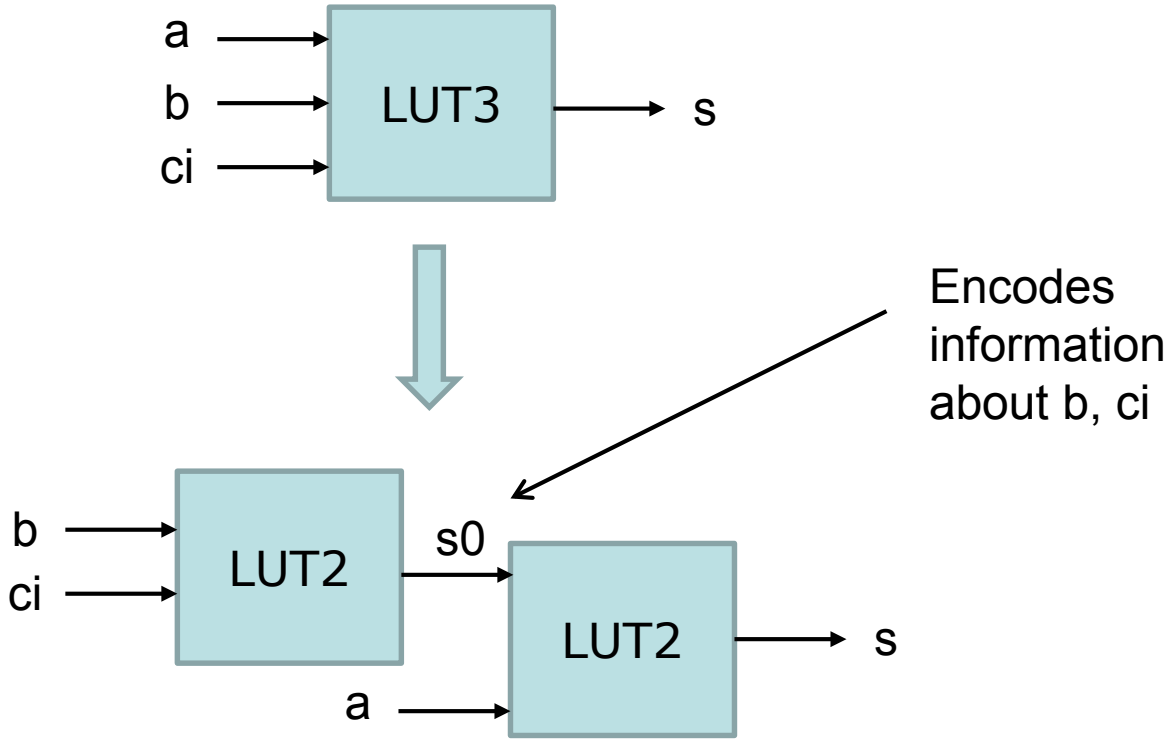
```
module full_adder (input a,b,ci,output s,co);
  assign s = a ^ b ^ ci;
  assign cout = (a & b) | (a & ci) | (b & ci);
endmodule
```

  - Synthesize: (Compile)

a	b	ci	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Mapping

- Assume our target FPGA has LUT2s
  - Can't map an 3-input function to one LUT2…



Mapping

- s = a xor b xor c

- Equivalent to…

s = (a)(~b)(~c)+ (~a)(b)(~c) + (~a)(~b)(c) + (a)(b)(c)

- Transform:

s = (~a)[(b)(~c) + (~b)(c)] + (a)[(~b)(~c) + (b)(c)]
s = (~a)[(b)(~c) + (~b)(c)] + (a)(~[(b+c) (~b+~c)])
s = (~a)[(b)(~c) + (~b)(c)] + (a)(~[(b)(~b)+(b)(~c)+(~b)(c)+(c)(~c)])
s = (~a)[(b)(~c) + (~b)(c)] + (a)(~[(b)(~c)+(~b)(c)])

- Set s0 = (b)(~c) + (~b)(c)
- s = (~a)(s0) + (a)(~s0)

Verilog Example

a	b	ci	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

a	b	ci	s0	s
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

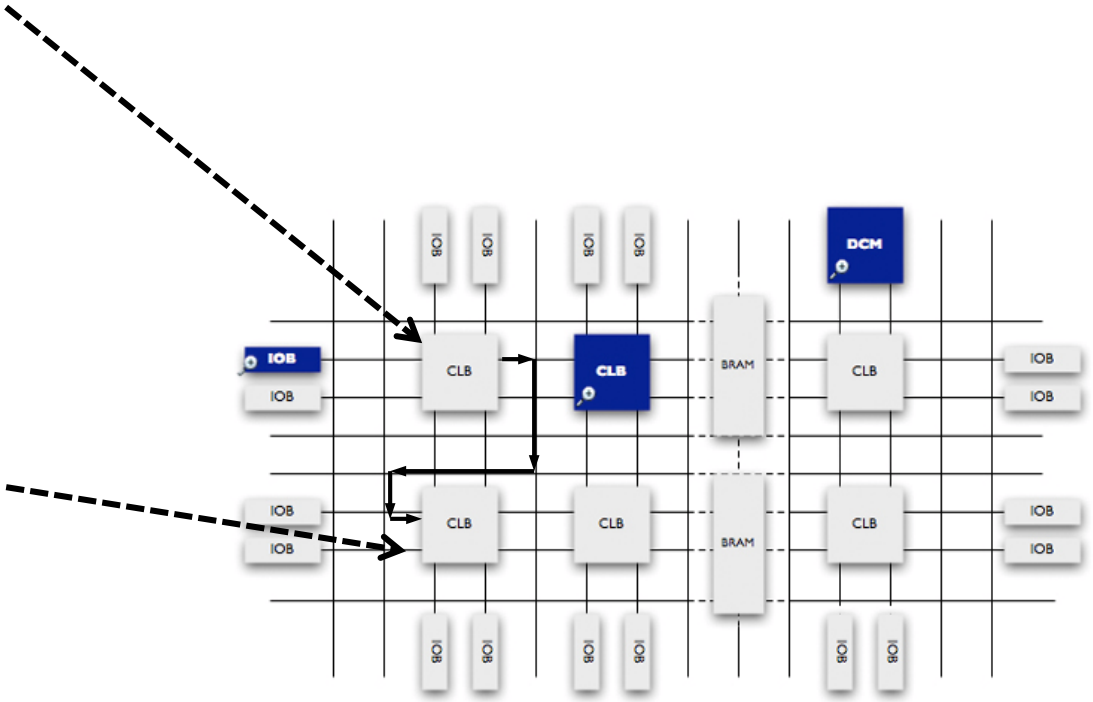
a	b	ci	s0
X	0	0	0
X	0	1	1
X	1	0	1
X	1	1	0

a	b	ci	s0	s
0	X	X	0	0
0	X	X	1	1
1	X	X	0	1
1	X	X	1	0

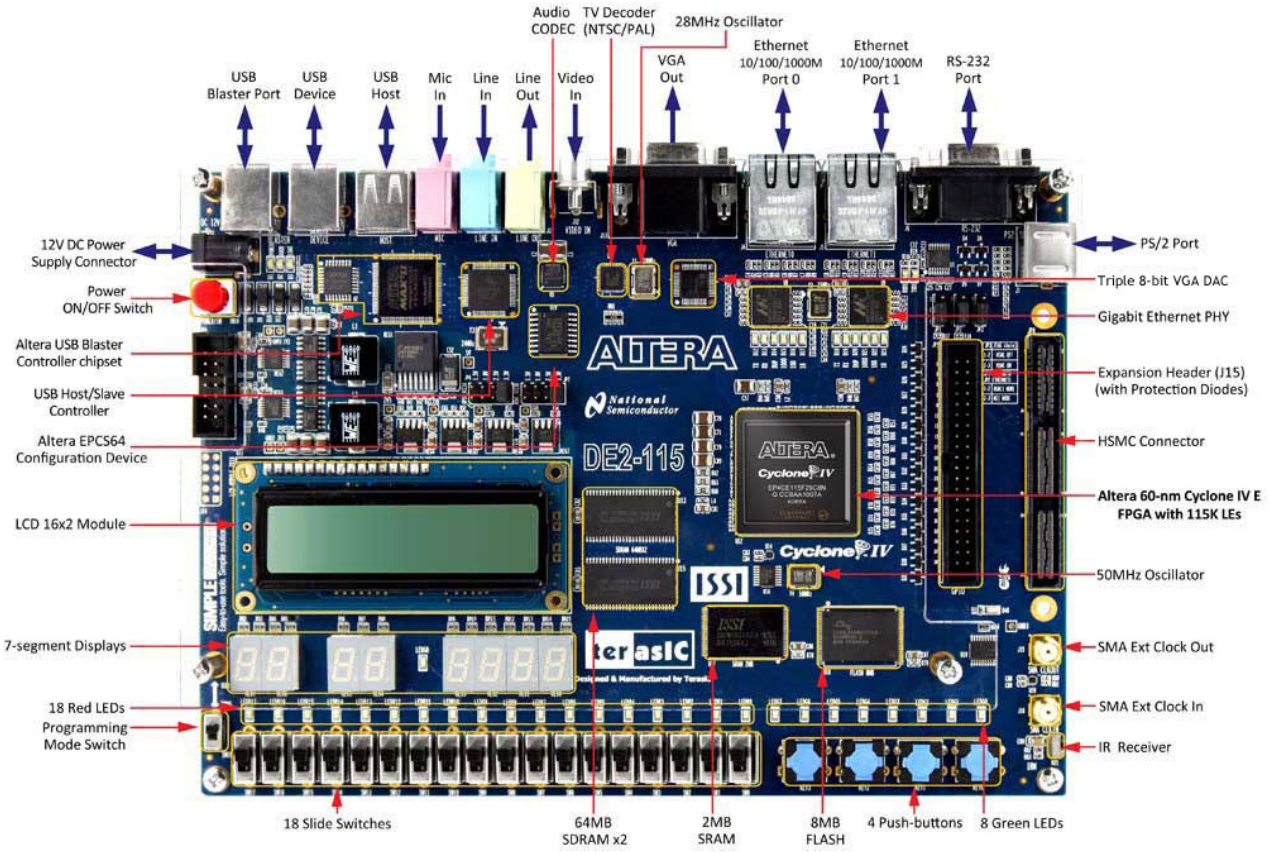
Place and Route

b	ci	s0
0	0	0
0	1	1
1	0	1
1	1	0

a	s0	s
0	0	0
0	1	1
1	0	1
1	1	0



Terasic DE2-115



- Altera Cyclone 4 FPGA with 115K gates

TODO ASAP

1. Make sure you can log into Dropbox:

   http://dropbox.cse.sc.edu

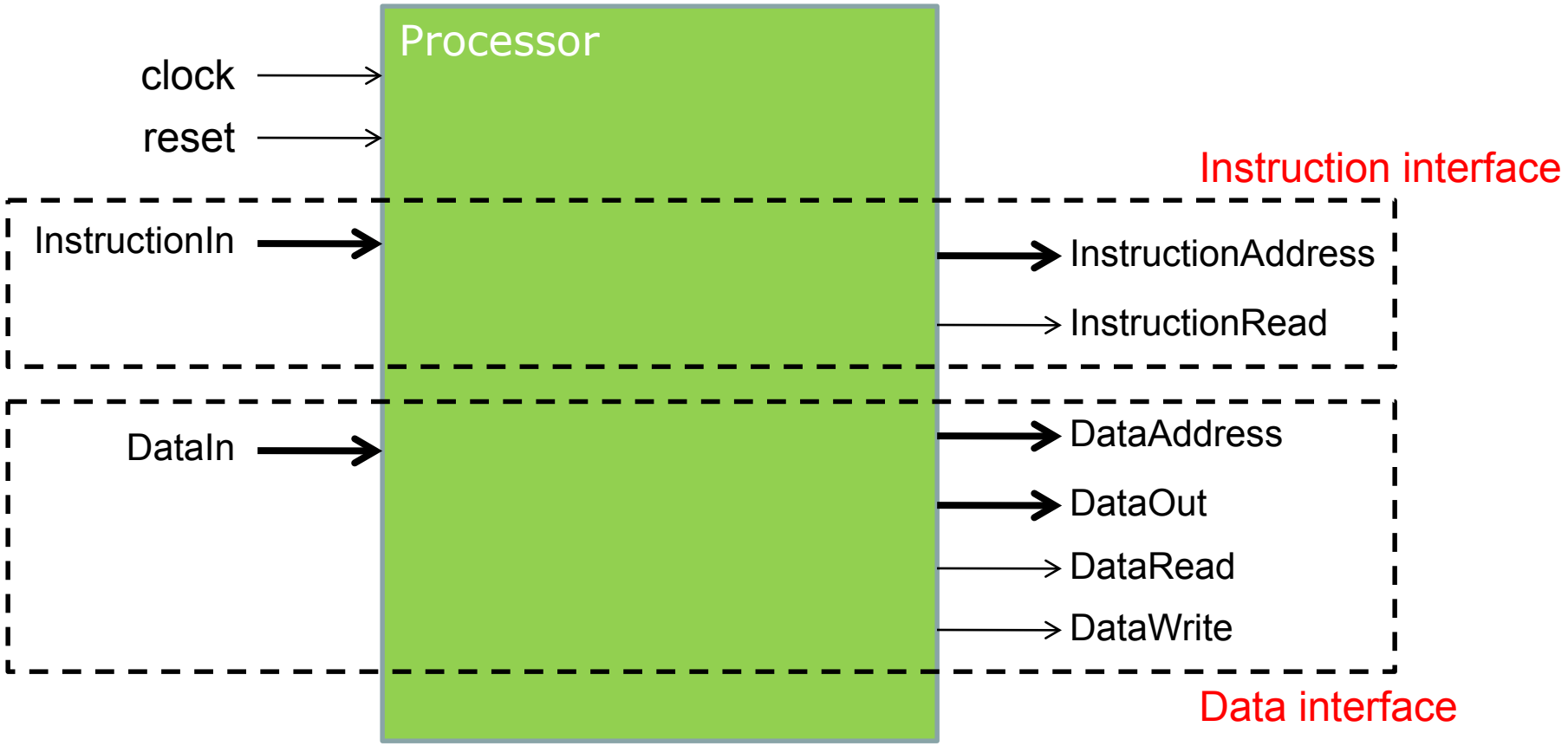
2. Select your lab partner

3. One lab partner: e-mail jbakos@sc.edu with names

System Design

- Processors communicate with the outside world using a simple transactional model:
  - READ:
    - Processor says READ and provides an address
    - Operations that depend on this data WAIT until data is returned
  - WRITE:
    - Processor says WRITE and provides an address and data
- These operations correspond to the LOAD and STORE instructions
- In this case, we assume that CPU is the master and devices responding to these operations are slaves

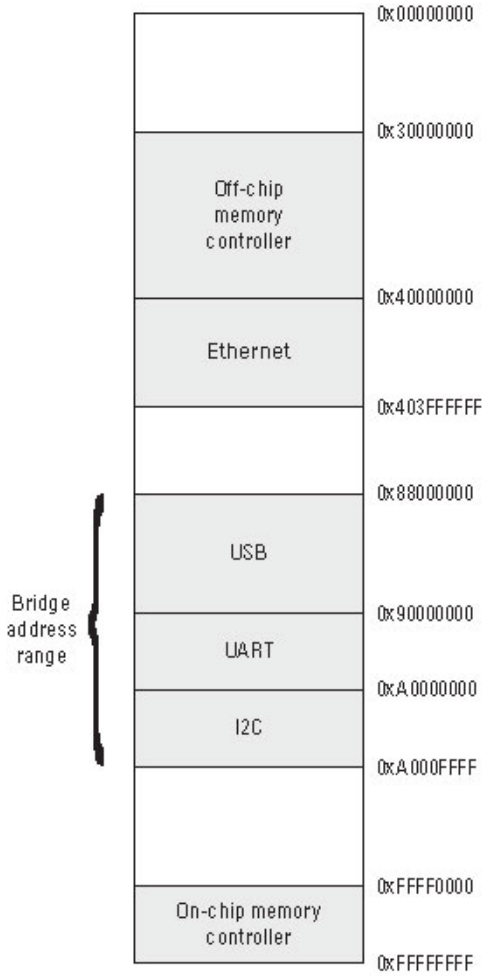
Processor Interface



Programmed I/O

- Loads and stores to specially-mapped address ranges can be used to:
  - Read a word from a **status register**
    - Used to poll the state of a peripheral
  - Write a word to a **control register**
    - Used to send an "instruction" to a peripheral

Sample Address Map



Altera Tools

- Quartus II
  - Starting point for all designs (create and open *projects*)
  - Contains simple editors for HDL design and constraint files
  - Has a makefile-like design flow manager for synthesis, map, place and route, bitstream generation, and programming
- Platform Designer
  - Allows for drag-and-drop creations of platform designs (processors, busses, peripherals)
- Command line tools
  - Compiler and run code on NIOS2 processor

Platform Designer

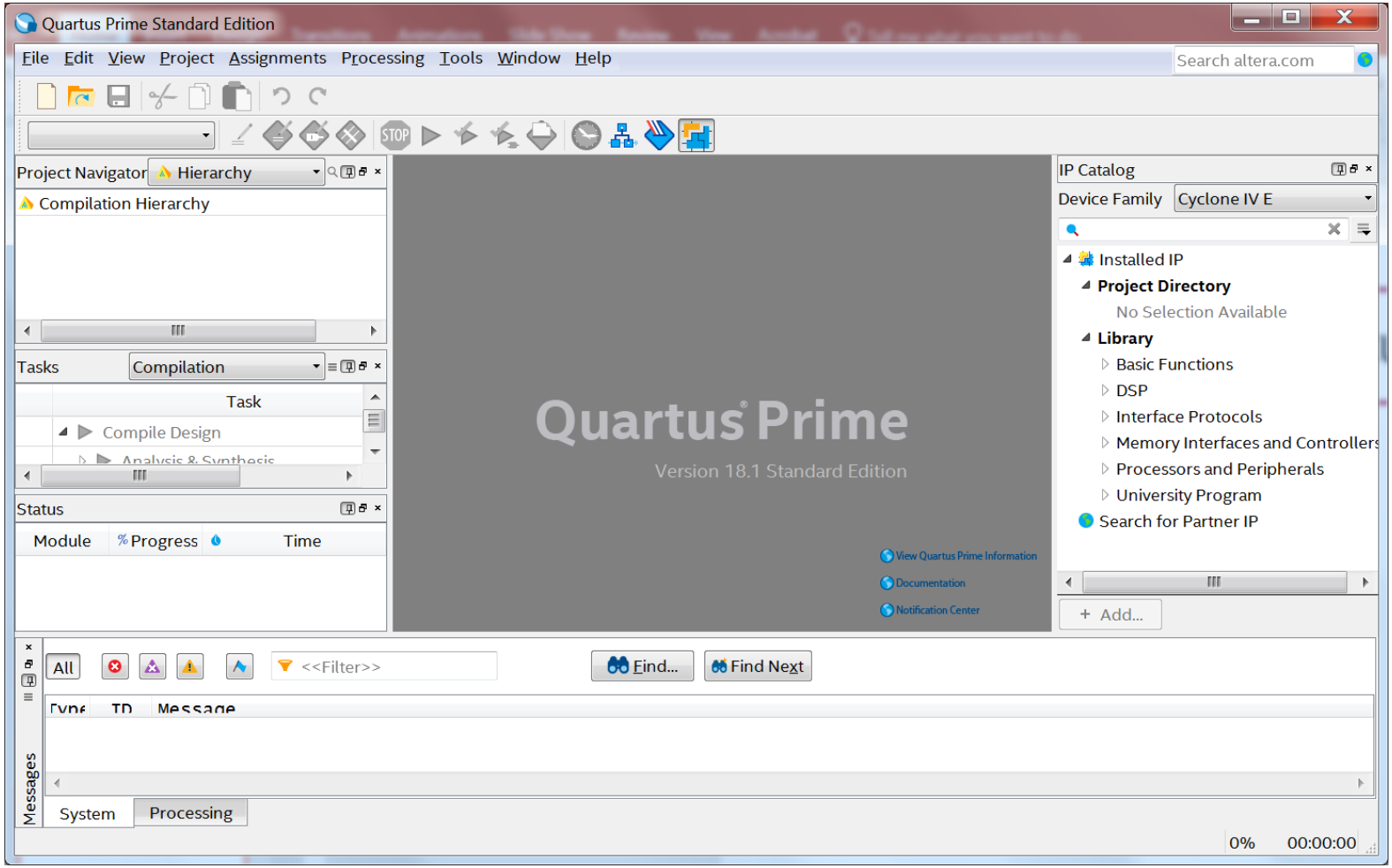
- Platform Designer allows you to design the portion of your embedded system that is implemented on the FPGA
- Using this information, the command-line tools can generate a BSP that corresponds to your system
- The BSP includes the interface code for the peripherals that you add in SOPC Builder
  - As such, it must be regenerated each time you make a change in your system design

Setup Your Environment

- Launch Quartus:

```
quartus&
```

Quartus

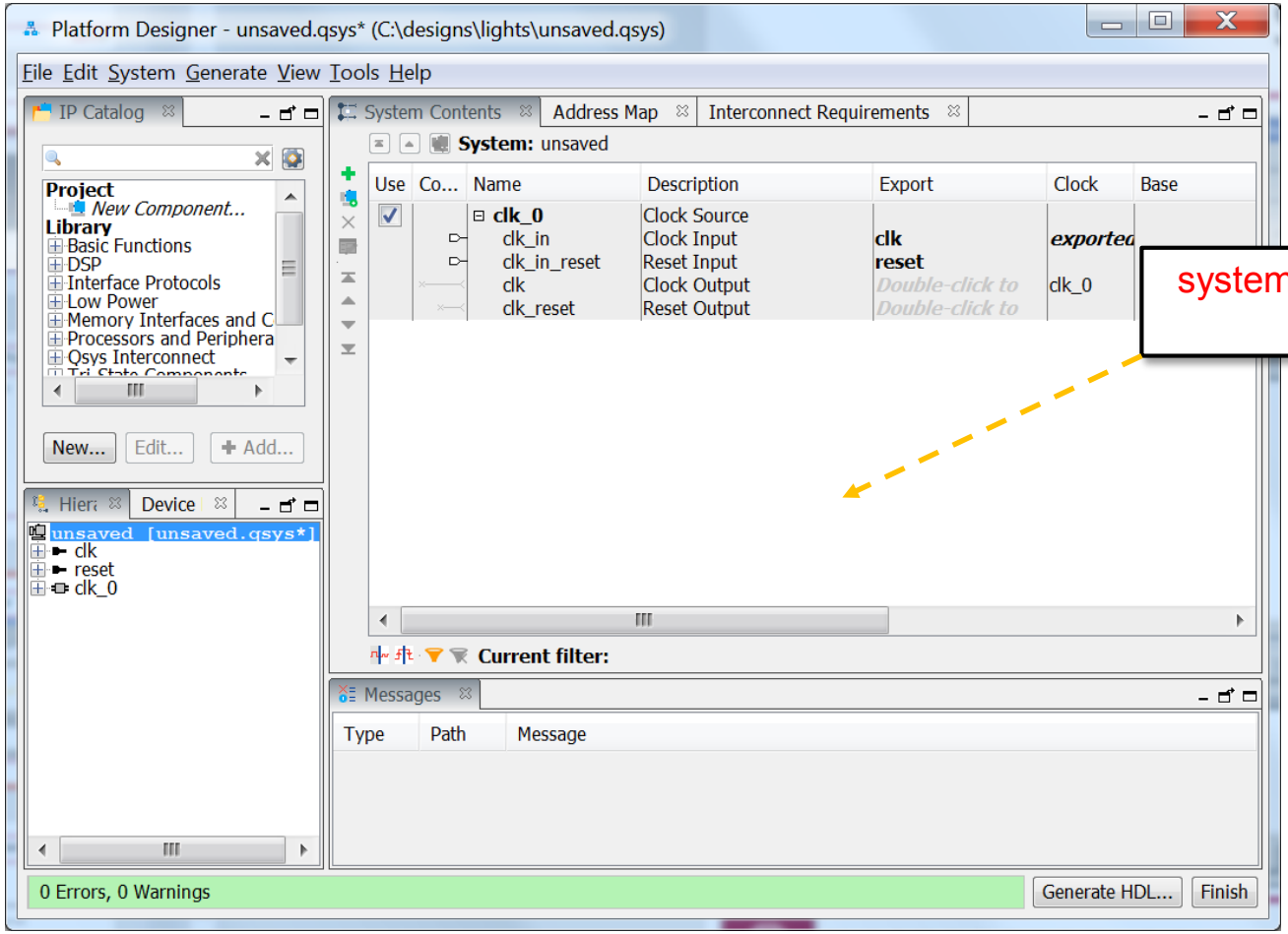

Quartus Prime Standard Edition
File Edit View Project Assignments Processing Tools Window Help
Search altera.com
Project Navigator
Hierarchy
Compilation Hierarchy
Tasks
Compilation
Task
Compile Design
Status
Module
Progress
Time
Quartus Prime
Version 18.1 Standard Edition
View Quartus Prime Information
Documentation
Notification Center
IP Catalog
Device Family
Cyclone IV E
Installed IP
Project Directory
No Selection Available
Library
Basic Functions
DSP
Interface Protocols
Memory Interfaces and Controllers
Processors and Peripherals
University Program
Search for Partner IP
Add...
All
<<Filter>>
Find...
Find Next
Type
ID
Message
Messages
System
Processing
0%
00:00:00

Creating a New Project

- File | New | New Quartus II Project…
- Working directory = /acct/<username>/lights
- Project name = "lights"
- Top-level design entity = "lights"
- "Empty project"
- Skip the "Add Files" page
- For device, choose
  - **Family**: Cyclone IV E
  - **Package:** FBGA
  - **Pin count:** 780
  - **Speed grade:** 7
  - **Device:** EP4CE115F29C7
- Click Finish

- Go to Tools | Platform Designer

Platform Designer



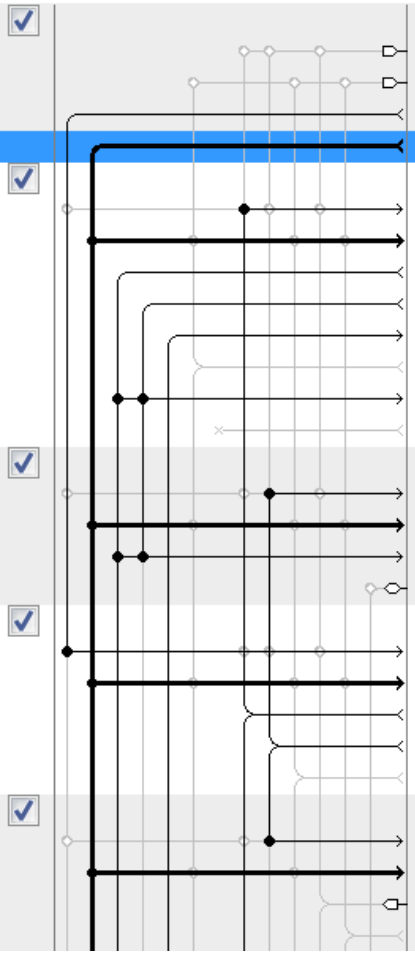
Adding Components

- Add a processor
  - In the component library, search for "nios2"
  - Double-click "Nios II Processor"
  - Select **Nios II/f**, then FINISH
- Add an interface to the SDRAM
  - In the component library, search for "sdram"
  - Double-click "SDRAM Controller Intel FPGA IP"
  - Presets=
    - Bits=32, chip select=1, banks=4, row=13, column=10, then FINISH
- Add a clock manager
  - In the component library, search for "clocks"
  - Double-click "System and SDRAM Clocks for DE-series Boards"
- Add another clock source
  - In the component library, search for "clock source"
  - Double-click "clock source"

Connecting Clock and Reset

- Start with the clk_0 component, connect:
  - "clock output" to "ref_clk" on sys_sdram_pll_0
  - "clk_reset" to "ref_reset" on sys_sdram_pll_0
  - "clk_reset" to "reset" on nios2 and sdram_controller
- From sys_sdram_pll_0:
  - "sys_clk" to "clk" on nios2_gen2_0
  - "sdram_clock" to "clk" on new_sdram_controller
- From nios2_gen2_0
  - "data_master" and "instruction_master" to "s1" on sdram
- From clk_1
  - "clk_in" to "sdram_clk" on "sys_sdram_pll_0"
  - Double-click the "Export" column for clk
- Double-click nios2
  - Under "Vectors"
    - Reset and Exception vector memory: sdram_controller
- System | Assign Base Addresses
- File | Save As "nios_system"

Platform Design



⊟ **clk_0**	Clock Source		
clk_in	Clock Input	**clk**	***exported***
clk_in_reset	Reset Input	**reset**	
clk	Clock Output	*Double-click to export*	clk_0
clk_reset	Reset Output	*Double-click to export*	
⊟ **nios2_gen2_0**	Nios II Processor		
clk	Clock Input	*Double-click to export*	**sys_sdram_pll_0_sys_clk**
reset	Reset Input	*Double-click to export*	[clk]
data_master	Avalon Memory Mapped ...	*Double-click to export*	[clk]
instruction_master	Avalon Memory Mapped ...	*Double-click to export*	[clk]
irq	Interrupt Receiver	*Double-click to export*	[clk]
debug_reset_request	Reset Output	*Double-click to export*	[clk]
debug_mem_slave	Avalon Memory Mapped ...	*Double-click to export*	[clk]
custom_instruction_m...	Custom Instruction Master	*Double-click to export*	
⊟ **new_sdram_controll...**	SDRAM Controller Intel F...		
clk	Clock Input	*Double-click to export*	**sys_sdram_pll_0_sdram_clk**
reset	Reset Input	*Double-click to export*	[clk]
s1	Avalon Memory Mapped ...	*Double-click to export*	[clk]
wire	Conduit	**new_sdram_controller_0_wire**	
⊟ **sys_sdram_pll_0**	System and SDRAM Cloc...		
ref_clk	Clock Input	*Double-click to export*	**clk_0**
ref_reset	Reset Input	*Double-click to export*	[ref_clk]
sys_clk	Clock Output	*Double-click to export*	sys_sdram_pll_0_sys_clk
sdram_clk	Clock Output	*Double-click to export*	sys_sdram_pll_0_sdram_clk
reset_source	Reset Output	*Double-click to export*	
⊟ **clk_1**	Clock Source		
clk_in	Clock Input	*Double-click to export*	**sys_sdram_pll_0_sdram_clk**
clk_in_reset	Reset Input	*Double-click to export*	
clk	Clock Output	**sdram_clk**	clk_1
clk_reset	Reset Output	*Double-click to export*	

Platform Design

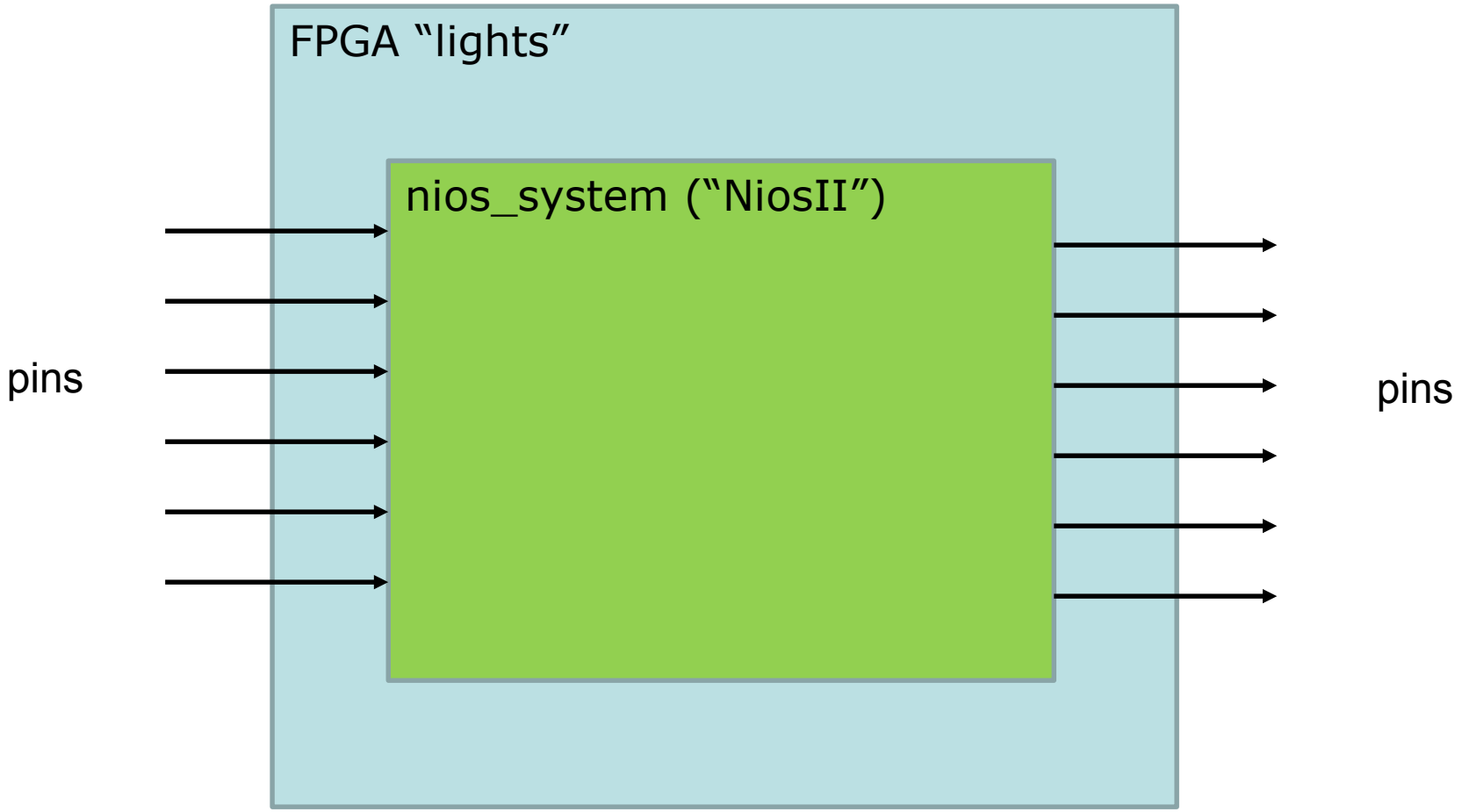
- Click "Generate HDL"
  - Accept default settings

- Go back to Quartus
  - Assignments | Import Assignments
    - "/usr/local/3rdparty/csce611/CPU_support_files/DE2_115_pin_assignments.qsf"
  - Assignments | Settings
    - Files | Add | "nios_system.qsys"
  - File | New | Synopsys Design Constraint File
    - Contents:

      create_clock -name CLOCK_50 -period 20 [get_ports CLOCK_50]

      derive_pll_clocks -create_base_clocks
    - Save as SDC1.sdc
  - File | New | Verilog HDL File
    - Contents:

Top-Level Design

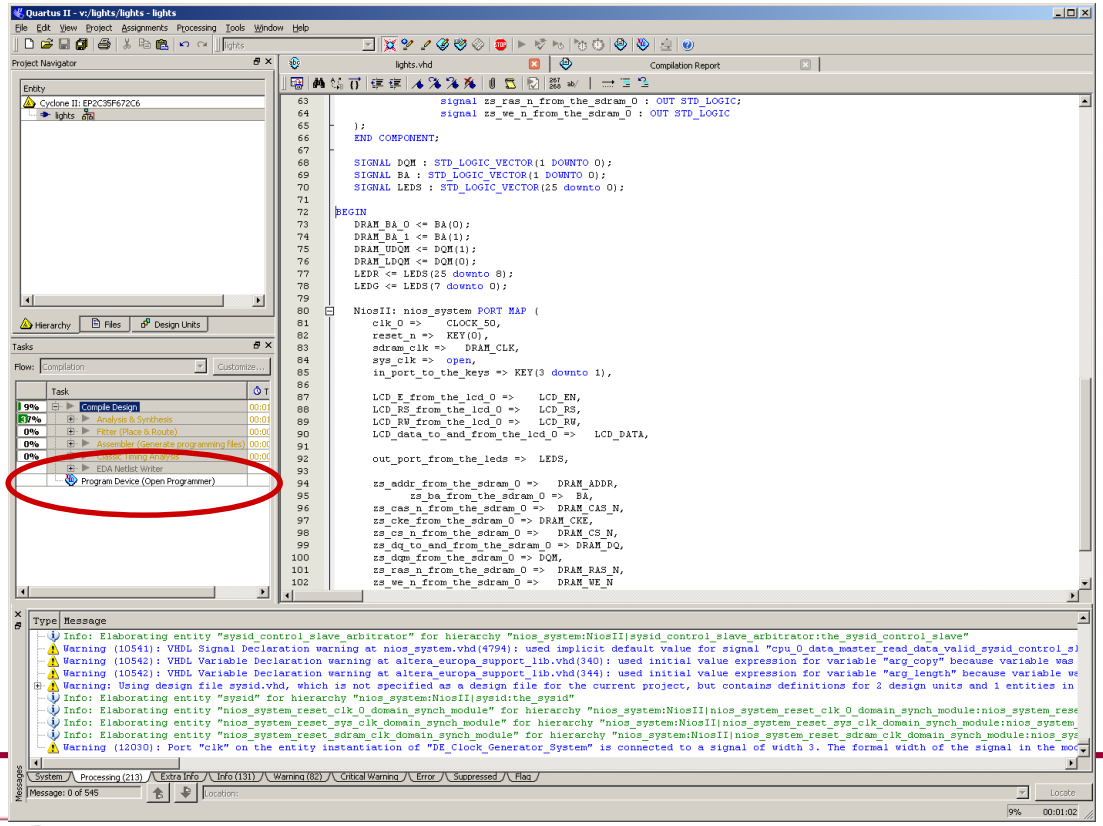


lights.v

```
module lights (input CLOCK_50,
               input [3:0] KEY,
               output [12:0] DRAM_ADDR,
               output [1:0] DRAM_BA,
               output DRAM_CAS_N,
               output DRAM_CKE,
               output DRAM_CS_N,
               inout [31:0] DRAM_DQ,
               output [3:0] DRAM_DQM,
               output DRAM_RAS_N,
               output DRAM_WE_N,
               output DRAM_CLK);

               nios_system u0 (
                        .clk_clk                       (CLOCK_50),                    //                    clk.clk
                        .reset_reset_n                 (KEY[0]),                      //                  reset.reset_n
                        .new_sdram_controller_0_wire_addr  (DRAM_ADDR),  // new_sdram_controller_0_wire.addr
                        .new_sdram_controller_0_wire_ba    (DRAM_BA),    //                            .ba
                        .new_sdram_controller_0_wire_cas_n (DRAM_CAS_N), //                            .cas_n
                        .new_sdram_controller_0_wire_cke   (DRAM_CKE),   //                            .cke
                        .new_sdram_controller_0_wire_cs_n  (DRAM_CS_N),  //                            .cs_n
                        .new_sdram_controller_0_wire_dq    (DRAM_DQ),    //                            .dq
                        .new_sdram_controller_0_wire_dqm   (DRAM_DQM),   //                            .dqm
                        .new_sdram_controller_0_wire_ras_n (DRAM_RAS_N), //                            .ras_n
                        .new_sdram_controller_0_wire_we_n  (DRAM_WE_N),  //                            .we_n
                        .sdram_clk_clk                 (DRAM_CLK)
               );

endmodule
```

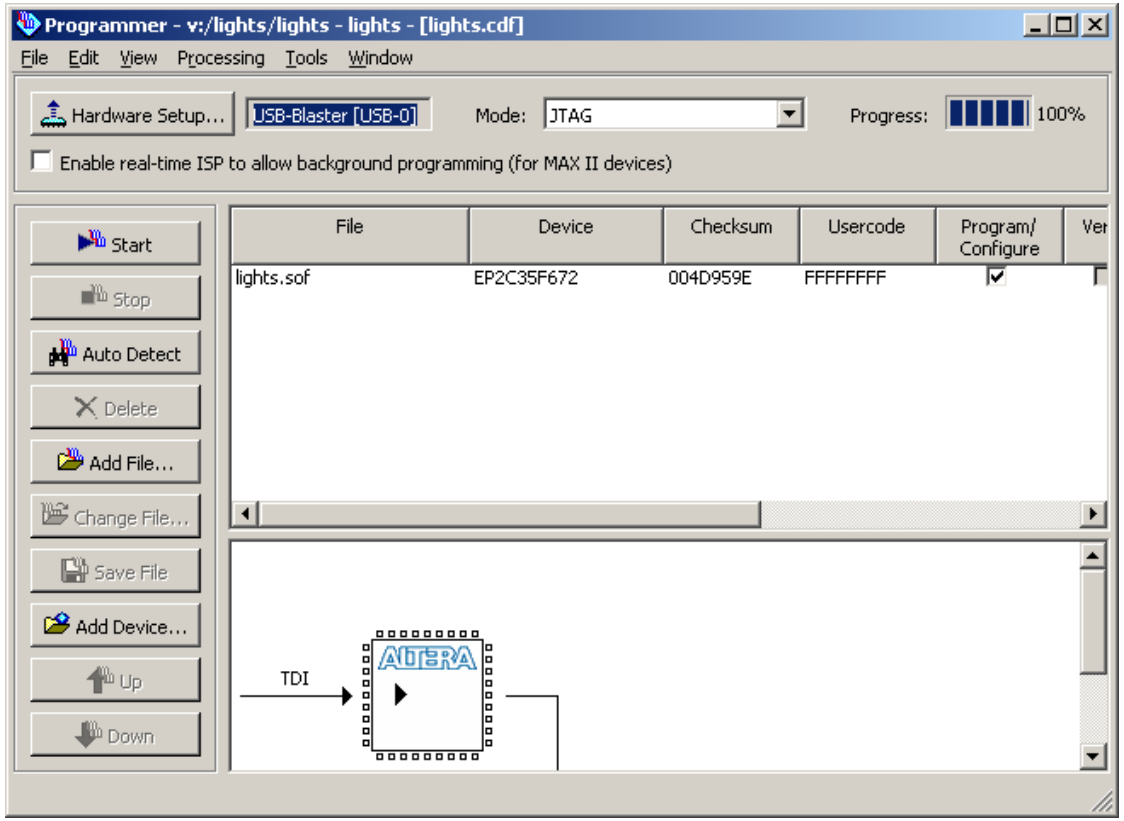
Hardware

- Re-compile the design…
- Program the FPGA…
  - Double-click on Program Device



Hardware

- Click Start



Building and Running Software

- Go back to Quartus and compile your design

- Open terminal
  - cd lights
  - mkdir software
  - cd software
  - Type:

  nios2-swexample-create --name=lights --sopc-file=../nios_system.sopcinfo --type=hello_world --cpu-name=nios2_gen2_0 --app-dir=lights --bsp-dir=lights_bsp
  - cd lights
  - ./create-this-app

hello_world.c

- Open hello_world.c

```
#include <stdio.h>

int main()
{
  double new_val = 3.0;
  double val = 0;
  double start = 2.0;
  double temp;
  int i=0,toggle=0;

  while (val != new_val) {
          val = new_val;
    temp = start * (start + 1.0) * (start + 2.0);
    if (!toggle)
      new_val += 4.0 / temp;
    else
      new_val -= 4.0 / temp;
    toggle = !toggle;
    start = start + 2.0;
    i++;
  }

  printf ("value is %0.8f\n",val);

  return 0;
}
```

Debugging

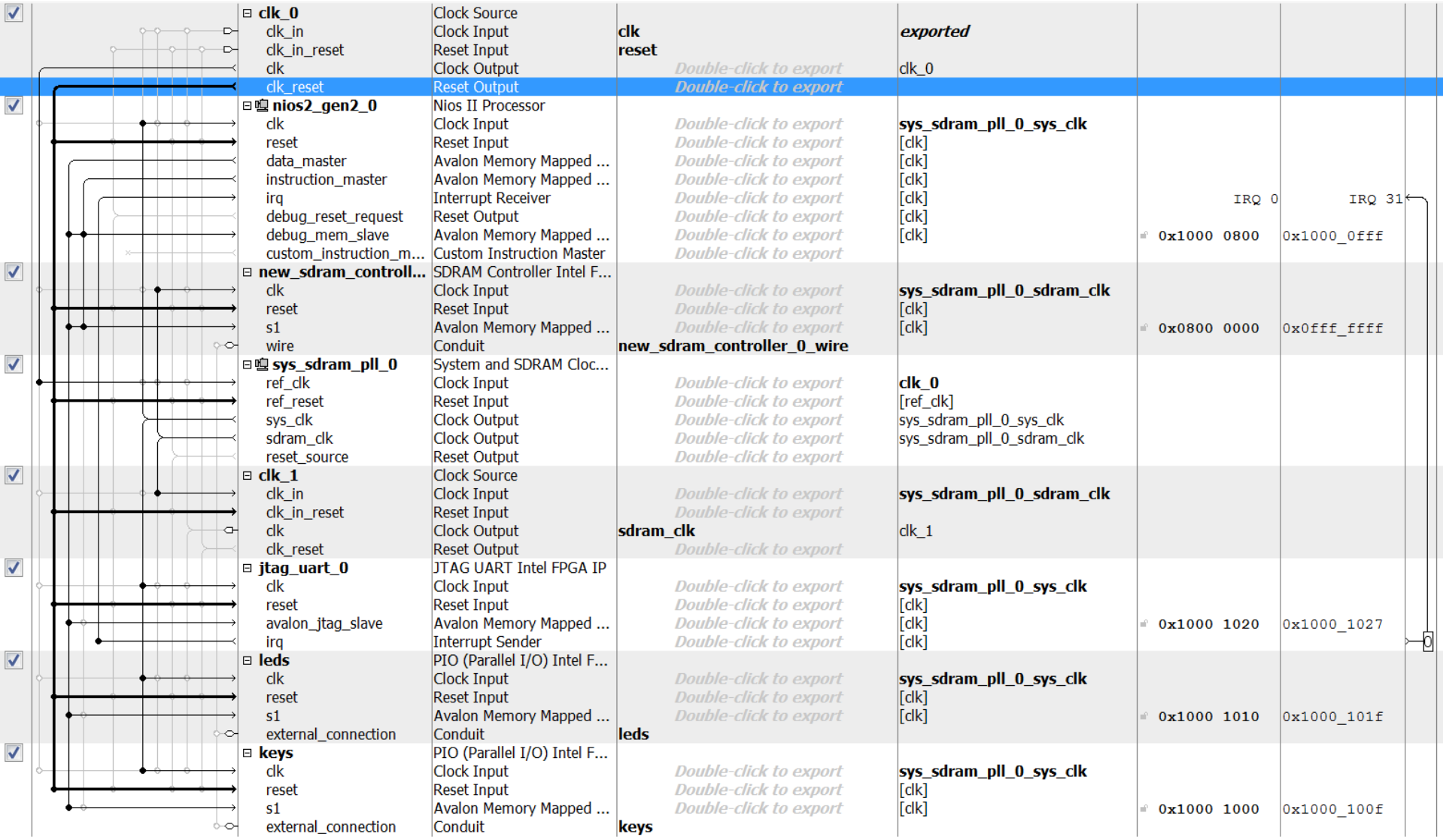
- make
- nios2-gdb-server --tcpport 8888 --tcppersist&
- nios2-elf-gdb lights.elf
  - target remote localhost:8888
  - load
  - b hello_world.c:39
  - c
  - p val (0)
  - c
  - p val (3)
  - c
  - p val (3.1667)
  - c
  - p val (3.1333)
  - c
  - del
  - b 39
  - c
  - p val (3.1416)
  - p i (131072)

Common GDB Commands

- p <print expression>
- bt (backtrace)
- f (frame)
- c (continue)
- s (step over)
- n (step into)
- fin (step out)
- b (breakpoint)
- del (delete breakpoint)

Adding Components

- Add a JTAG UART for the console
  - use sys_clk and global reset, connect control slave to CPU
  - Search "uart", add "JTAG UART Intel FPGA IP"
  - Accept default settings, connect as above
  - Also: connect irq to CPU
- Add parallel I/O for the LEDs
  - Search "pio", add "PIO (Parallel I/O) Intel FPGA IP"
  - Width=26, output ports only, FINISH, rename it as "leds", then put it on the sys_clk
  - Connect as above
  - Double-click export column and rename to "leds"
- Add parallel I/O for the keys (buttons)
  - Same as above, but 3 bits, input only
  - Under "Input Options", turn on "Synchronously capture", then FINISH
  - Rename as "keys" and put it on the sys_clk


clk_0 Clock Source
clk_in Clock Input clk exported
clk_in_reset Reset Input reset
clk Clock Output Double-click to export clk_0
clk_reset Reset Output Double-click to export
nios2_gen2_0 Nios II Processor
clk Clock Input Double-click to export sys_sdram_pll_0_sys_clk
reset Reset Input Double-click to export [clk]
data_master Avalon Memory Mapped ... Double-click to export [clk]
instruction_master Avalon Memory Mapped ... Double-click to export [clk]
irq Interrupt Receiver Double-click to export [clk] IRQ 0 IRQ 31
debug_reset_request Reset Output Double-click to export [clk]
debug_mem_slave Avalon Memory Mapped ... Double-click to export [clk] 0x1000 0800 0x1000_0fff
custom_instruction_m... Custom Instruction Master Double-click to export
new_sdram_controll... SDRAM Controller Intel F...
clk Clock Input Double-click to export sys_sdram_pll_0_sdram_clk
reset Reset Input Double-click to export [clk]
s1 Avalon Memory Mapped ... Double-click to export [clk] 0x0800 0000 0x0fff_ffff
wire Conduit new_sdram_controller_0_wire
sys_sdram_pll_0 System and SDRAM Cloc...
ref_clk Clock Input Double-click to export clk_0
ref_reset Reset Input Double-click to export [ref_clk]
sys_clk Clock Output Double-click to export sys_sdram_pll_0_sys_clk
sdram_clk Clock Output Double-click to export sys_sdram_pll_0_sdram_clk
reset_source Reset Output Double-click to export
clk_1 Clock Source
clk_in Clock Input Double-click to export sys_sdram_pll_0_sdram_clk
clk_in_reset Reset Input Double-click to export
clk Clock Output sdram_clk clk_1
clk_reset Reset Output Double-click to export
jtag_uart_0 JTAG UART Intel FPGA IP
clk Clock Input Double-click to export sys_sdram_pll_0_sys_clk
reset Reset Input Double-click to export [clk]
avalon_jtag_slave Avalon Memory Mapped ... Double-click to export [clk] 0x1000 1020 0x1000_1027
irq Interrupt Sender Double-click to export [clk]
leds PIO (Parallel I/O) Intel F...
clk Clock Input Double-click to export sys_sdram_pll_0_sys_clk
reset Reset Input Double-click to export [clk]
s1 Avalon Memory Mapped ... Double-click to export [clk] 0x1000 1010 0x1000_101f
external_connection Conduit leds
keys PIO (Parallel I/O) Intel F...
clk Clock Input Double-click to export sys_sdram_pll_0_sys_clk
reset Reset Input Double-click to export [clk]
s1 Avalon Memory Mapped ... Double-click to export [clk] 0x1000 1000 0x1000_100f
external_connection Conduit keys

lights.v

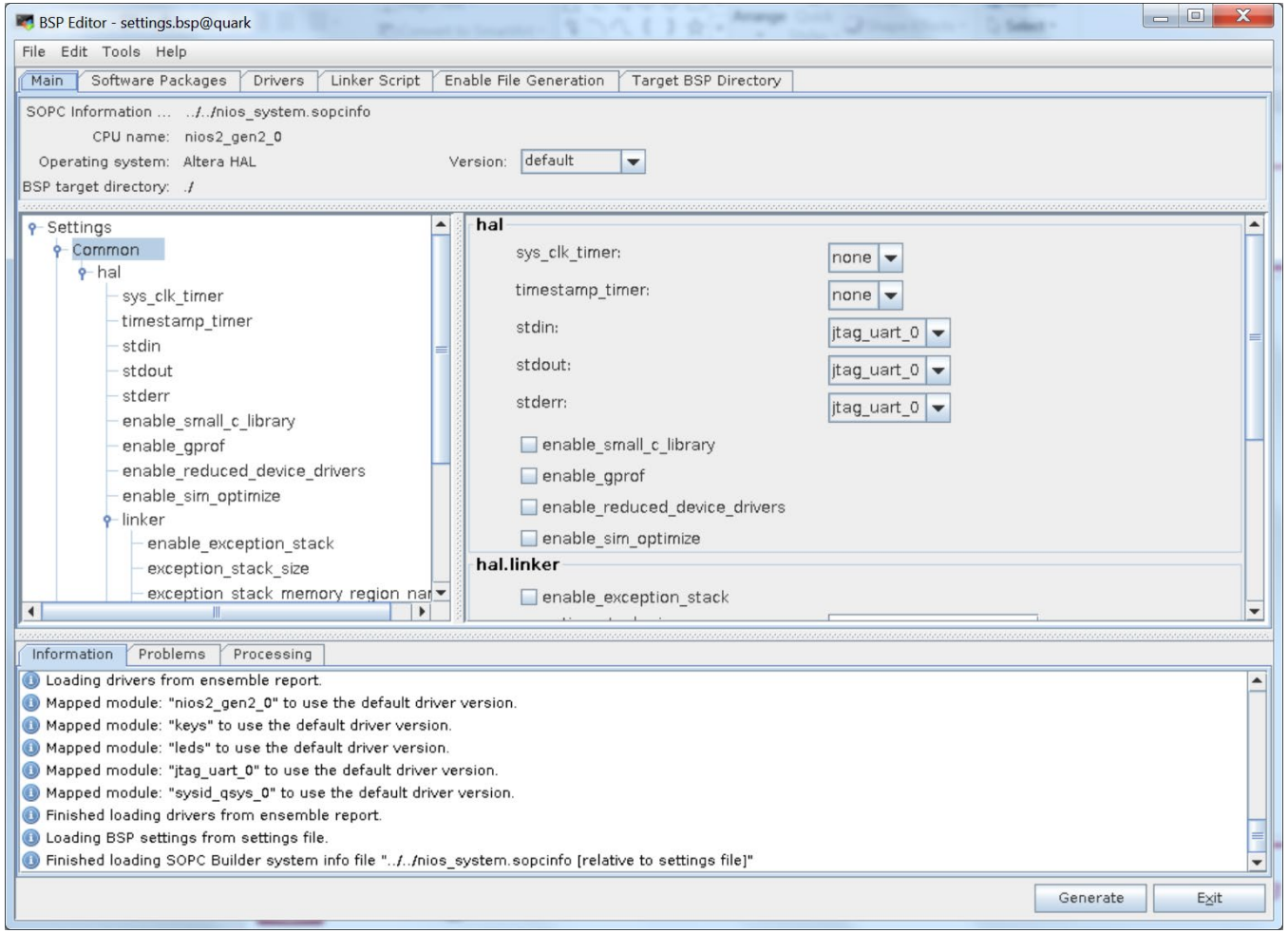
```
module lights (input CLOCK_50,
                    input [3:0] KEY,
                    output [12:0] DRAM_ADDR,
                    output [1:0] DRAM_BA,
                    output DRAM_CAS_N,
                    output DRAM_CKE,
                    output DRAM_CS_N,
                    inout [31:0] DRAM_DQ,
                    output [3:0] DRAM_DQM,
                    output DRAM_RAS_N,
                    output DRAM_WE_N,
                    output DRAM_CLK,
                    output [17:0] LEDR,
                    output [7:0] LEDG
);

                    nios_system u0 (
                                .clk_clk                       (CLOCK_50),                    //                 clk.clk
                                .reset_reset_n                 (KEY[0]),                //                 reset.reset_n
                                .new_sdram_controller_0_wire_addr  (DRAM_ADDR),  // new_sdram_controller_0_wire.addr
                                .new_sdram_controller_0_wire_ba    (DRAM_BA),    //                           .ba
                                .new_sdram_controller_0_wire_cas_n (DRAM_CAS_N), //                           .cas_n
                                .new_sdram_controller_0_wire_cke   (DRAM_CKE),   //                           .cke
                                .new_sdram_controller_0_wire_cs_n  (DRAM_CS_N),  //                           .cs_n
                                .new_sdram_controller_0_wire_dq    (DRAM_DQ),    //                           .dq
                                .new_sdram_controller_0_wire_dqm   (DRAM_DQM),   //                           .dqm
                                .new_sdram_controller_0_wire_ras_n (DRAM_RAS_N), //                           .ras_n
                                .new_sdram_controller_0_wire_we_n  (DRAM_WE_N),  //                           .we_n
                                .sdram_clk_clk                 (DRAM_CLK),
                                .keys_export                   (KEY[3:1]),                    //                 keys.export
                                .leds_export                   ({LEDR,LEDG}),                 //                 leds.export
                    );

endmodule
```

lights

- Note: if you get the error:
  - "XXX (XXX) overlaps XXX (XXX)"
  - Select System | Assign Base Addresses

- Stop GDB:
  - pkill -9 .*gdb.*
- Download new SOF:
  - nios2-configure-sof ../../output_files/lights.sof
- Update bsp:
  - cd ../lights_bsp
  - nios2-bsp-generate-files --settings="settings.bsp" --bsp-dir=.
  - cd lights_bsp
  - nios2-bsp-editor

BSP Editor


BSP Editor - settings.bsp@quark
File Edit Tools Help
Main
Software Packages
Drivers
Linker Script
Enable File Generation
Target BSP Directory
SOPC Information ... ../../nios_system.sopcinfo
CPU name: nios2_gen2_0
Operating system: Altera HAL
Version: default
BSP target directory: ./
Settings
Common
hal
sys_clk_timer
timestamp_timer
stdin
stdout
stderr
enable_small_c_library
enable_gprof
enable_reduced_device_drivers
enable_sim_optimize
linker
enable_exception_stack
exception_stack_size
hal
sys_clk_timer: none
timestamp_timer: none
stdin: jtag_uart_0
stdout: jtag_uart_0
stderr: jtag_uart_0
enable_small_c_library
enable_gprof
enable_reduced_device_drivers
enable_sim_optimize
hal.linker
enable_exception_stack
Information
Problems
Processing
Loading drivers from ensemble report.
Mapped module: "nios2_gen2_0" to use the default driver version.
Mapped module: "keys" to use the default driver version.
Mapped module: "leds" to use the default driver version.
Mapped module: "jtag_uart_0" to use the default driver version.
Mapped module: "sysid_qsys_0" to use the default driver version.
Finished loading drivers from ensemble report.
Loading BSP settings from settings file.
Finished loading SOPC Builder system info file "../../nios_system.sopcinfo [relative to settings file]"
Generate
Exit

NIOS Terminal

- Open a new terminal
- nios2-download -g lights.elf
- see results in terminal window!

New hello_world.c

- Open hello_world.c
- Add header files:

```
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
```

Software

- New main () code:

```
alt_u32 current_value;
alt_u32 current_state;
alt_u8 current_direction;
alt_u32 keys;

current_state=3;
current_value=1;
current_direction=0;

printf ("Program running (UART)...\n");
```

Software

```
while (1) {
  // read the current state of the keys
  keys=IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE);
  // switch speed if necessary
  if ((keys != 7) && (keys != current_state)) {
    if (keys == 3)  printf ("speed set to 250 ms\n");
    else if (keys == 5) printf ("speed set to 150 ms\n");
    else if (keys == 6) printf ("speed set to 50 ms\n");
    current_state=keys;
  }
  // switch direction if necessary
  if ((current_direction==0) && (current_value==(1 << 25))) current_direction=1;
  else if ((current_direction==1) && (current_value==1)) current_direction=0;
  // move light
  else if (current_direction==0) current_value = current_value << 1;
  else current_value = current_value >> 1;
  // update lights
  IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,current_value);
  // wait
  if (current_state==3) usleep (250000);
  else if (current_state==5) usleep (125000);
  else usleep (50000); }
```

Quartus

- Back to Quartus…
- Now we need to write a top-level Verilog HDL file for lights
- File | New | Verilog HDL File