

# Design Contest Report Guidelines

Revision: April 10, 2009



## What Makes a Good Design Report?

As you may know, one of the purposes of the Digilent Design Contest is to identify excellent project designs that are ready to be posted to our website for others to download and replicate. But we also want to help you build an important engineering skill by providing some guidelines for writing a good design report.

Well-conceived designs and well-written documents can help establish engineers as valuable resources in their workplaces.

A design report is written by and for technical personnel, and it must contain enough technical information to allow a fellow engineer to independently recreate the design. It is one of the essential deliverables for any design project, almost equal in importance to the design itself.

A report serves several needs: it provides fellow engineers a convenient means to understand, evaluate and/or reuse work; it allows marketing/business personnel to extract project details that they may need; it allows engineers to recall their work months or years later; and it provides a means for engineers to disseminate their work.

A typical design report includes a description of the design objectives, technical descriptions of all major design blocks, a statement of the final success and/or utility of the design, a list of all hardware and software tools required to complete the design, locations of all source files, and references to external sources (like data sheets, specifications, or other design reports) where pertinent information can be found. A design report should present relevant design information in a clear and concise manner, without any narrative discussions or stories about how the design proceeded (other documents, like laboratory notebooks, can be used to capture lessons learned during the design, or other relevant thoughts that might be of general use). Component data sheets from semiconductor manufacturers often provide good examples of concise design information.

This document provides guidelines that can be helpful in preparing a suitable report for most design projects, but it cannot address all possible design environments. Different reports will require different amounts of detail, and perhaps even different content and organization altogether. Some reports may be a few pages long, and some may require dozens of pages. Every report must be tailored to the design at hand, but in general, these guidelines can serve as a starting point for most situations.

The following section provides an outline of a design report. In general, the major headings should be included in your report. A table of contents is generally not required unless the report is longer than 15 to 20 pages, and a list of figures is not required at all.

For samples of good (though not definitive) contest reports, see the *Digital Synth* and *FPGA for Life* reports, available from the Design Contest page at <http://www.digilentinc.com/events/ro2010/>. At the same location you can download a pre-formatted Word template that you can use for your report, if you wish.

Appendix A gives some guidelines for your presentation. For an excellent sample of a PowerPoint® presentation, see the *DigitalSynth Presentation* available from the same location.

## Report Outline

### Title Page

### Product Marketing Sheet

### Introduction

- Objectives
- Features-in-Brief
- Project Summary
- Tools Required
- Design Status

### Background

- Why This Project?
- Reference Material

### Design

- Features and Specifications
- Design Overview
- Detailed Design Description

### Discussion

- Problems Encountered
- Engineering Resources Used

### References

## Report Sections in Detail

### Title Page

Your report should have a title block that occupies at least half of the first page. Use an entire title page if your report is 15 pages or longer. The title block/page should include the project name, the design engineer's name and affiliation (e.g., your university), submission date, project supervisor (or instructor), and the reason the document was prepared (e.g., the Diligent Design Contest).

Project names should be descriptive, but not too wordy; usually just a few words. The project name should use a larger point size than the document text, and be distinguished with bold text, underlining, or other means. If a team of engineers was involved in the design, they should all be listed with their credentials, e.g., Electrical Engineer (student), Computer Engineer, etc.

### Product Marketing Sheet

This is a one-page “flyer” or “sell sheet” that can be used to market your “product”. Be creative and look to similar sell sheets, product advertisements, and even product boxes for inspiration. The goal here is to list the major features and specifications of your product, along with a top-level block diagram, photo, and other graphics.

For just one example, see the Nexys2 product page at [www.diligentinc.com](http://www.diligentinc.com).

### Introduction

The *Introduction* should be limited to one or two pages for larger designs, and one or two paragraphs for smaller designs. Don't hesitate to use marketing language in the *Introduction* to promote interest in your design. Keep in mind that reports are often used by engineers looking for reusable content from completed projects. The *Introduction* should help such readers find what they might be looking for.

### Objectives

Describe the intent and motivation behind your design. Present the overall design requirements or specifications, as well.

### Features-in-Brief

For larger designs, consider including a *Features-in-Brief* subsection to further orient the reader. It could be a table or list that details the design's major features. Use the same information density as a typical product advertisement or product box.

### Project Summary

Present a brief synopsis of the design, including the major features.

Describe the major design partitions. How is your design broken up? Is it a software design, a hardware design, a VHDL project?

Also describe the effectiveness of the design. Was it a good design? Did it meet objectives? Could all or part of it be used for other purposes, in other designs or contexts?

Explain where and how the design is used, if applicable. Is it used in a product, lab, or class?

Discuss whether the design is intended for particular hardware, or is generically useful.

### **Tools Required**

List all of the hardware and software tools that were necessary for implementing your project (e.g., equipment, operating system, CAD software, etc.)

### **Design Status**

It's OK for the project, or major features, to be incomplete. Please do not withdraw from the contest if your project is incomplete! In a previous contest, there was a team that would have won had they not withdrawn simply because a particular feature was incomplete. In such a case, go ahead and describe the incomplete feature(s).

Describe the status of the project. Was it completed? If not, when do you estimate it will be completed? Was the design abandoned? What would enable completion of the project?

## **Background**

### **Why This Project?**

Explain why you chose to do this project. For example, is it a new product idea? Is it an improvement to an existing product? Or was it just for fun, to gain experience with new products and technologies?

In business and industry, it is often of interest why a design is being built instead of purchased (typically known as the "build-versus-buy" decision). This is the place to provide the data and reasoning that was used to make the "build" decision.

Provide references to other designs, products, or components that your design must interface with.

Finally, discuss why you chose your particular design approach.

### **Reference Material**

In some cases, this section might contain brief theoretical information that is outside the experience of the general reader. For example, if the design is to control the deflection of a cathode ray in a display system, some background information on field strengths, inductor currents, phosphor illumination energies, and so on might be appropriate. Refer readers to other documentation, if necessary.

## Design

### Features and Specifications

The *Design* section is the most important part of the report.

The *Features and Specifications* subsection is to let readers know what the major features are and what specifications constrained the design. We don't want too much detail here, just list what features make your project cool, and, where relevant or important, what specifications those features meet.

### Design Overview

Begin this section with a statement that describes your project's top-level block diagram.

Diagrams are essential to the design process itself and also to creating a good report, so be sure to include a well-conceived top-level block diagram that clearly shows the functional design partition. A properly prepared block diagram serves as a table of contents for the *Design* section, by showing all circuit blocks for which further explanations will be provided. Brief written explanations should then be provided for every block in the diagram in the *Detailed Design Description* subsection.

For simple designs, only a single block is needed, and the report can merge the *Design Overview* and *Detailed Design Description* subsections.

For more complex designs, the block diagram should show the partition used to implement the circuit, and circuit blocks should map to individual source files whenever possible.

Circuit blocks should show "macro" functions like controllers, decoders, data path circuits, etc., but not simpler components like flip-flops, registers, adders, etc. Circuit blocks and all signals should be clearly identified, particularly those signals that interface with sequential controller blocks.

If you use a schematic entry tool for the top-level circuit, then the top-level schematic can be used as the block diagram, provided it is neat, readable, and well-commented.

Discuss the general design methods (e.g., behavioral VHDL, schematic capture, etc.) that were used for the project.

NOTE: For particularly good examples of useful diagrams and tables, see the *Digital Synth and FPGA for Life* project reports, available from the Design Contest section of [www.digilent.ro](http://www.digilent.ro).

### Detailed Design Description

Describe each major function or block in its own subsection. The subsection can be named for the block shown in the top-level block diagram (consider reproducing the individual component block diagram in the relevant subsection).

Begin each subsection with one or two sentences outlining the circuit block's function at a high level, and then describe (with appropriate technical detail) the circuit's function and implementation details.

Include any and all useful state diagrams, timing diagrams, block diagrams, and circuit diagrams or schematics.

For example, if an input needs to be at least 250ns long to start a state machine, but less than 3 $\mu$ s to avoid multiple back-to-back starts, then state that requirement. Or if an input must arrive before a given clock edge by some amount, then state that as well (or better yet, show it in a timing diagram). For more complex designs, important internal signals can also be shown and described in a similar table.

You should include your schematics and/or VHDL code as appendices to the project report, and you can reference sections of those source files in the body of your text (e.g., “line numbers 15-25 in source file xxx implement this function.”)

The bulk of the material in these subsections guides readers through the design source files. If source files are clean, well-written, easy to follow, and generously commented (see Appendix B), the amount of description in the report can be greatly reduced (you can simply refer to the comments in the source file).

Source files should always contain extensive comments that provide insight into circuit behavior. In particular, every source file should contain a comment block that states the file name, the design intent, major architectural features, and lists all I/O signals. Note that modern CAD tools allow comments to be added to schematics as well as HDL source files.

Any design with more than a few I/O signals should show all the signals in a separate table. The table should include the signal name, the signal mode (input, output, or bi-directional), a brief description of signal purpose, logic equations for simple combinational outputs, de-bouncing and/or synchronizing requirements for all inputs, output signal timing requirements where appropriate, any special current drive requirements, and any other pertinent signal requirements or features. Also, note any particular system-level timing dependencies for inputs.

Be sure to specifically list (perhaps in a separate table) all of the input and output signals for the block. For inputs, state precisely what the input does, and any conditions that are placed on the input (e.g., “Input A must be a pulse longer than 5 $\mu$ s, Input B must be de-bounced and synchronous, etc.”) For outputs, state the function or downstream use of each output, and any special timing or other conditions that apply to the output.

When documenting state machines, you should always show a state diagram. Be sure the state diagram shows all inputs and outputs, and that no undefined or ambiguous branches exist. Define any special timing requirements for input or output signals, and state how the circuit implements those requirements. List any unused state codes, and where appropriate, include an analysis of circuit behavior should the machine enter an undefined state.

Consider preparing an outline-style list that describes exactly what happens in each state in the diagram (i.e., describe the inputs, outputs, and system behaviors for each state). Make sure the state diagram exactly matches the circuit’s behavior.

Be sure to draw attention to any programmable or user-modifiable features. For instance, if your design has been created to run at 10Khz, but it is likely that the same design might be re-usable if it could run faster, then describe how that might easily be accomplished (e.g., “the main counter-divider

ratio can easily be changed to modify the controller's frequency"). Or, if your design can be controlled through a programmable interface, specifically list all points of programmability, any default values, all applicable programming options, and what each programming option does (e.g., "writing a 100 to the second register will select the red LEDs as the output.")

Where possible, include information in each subsection that details interesting or useful design metrics. For example, information on power consumption, operating speed, number of components (or logic elements for FPGA designs), etc., would allow other engineers to consider reusing certain parts of your design. If this information is not included in individual subsections, it should be included in the *Discussion* section, or at the start of the *Design* section.

If you refer to other source documents for the provision of technical information, then list proper references to those documents in each subsection. But be sure that the combination of the information provided in this document and in the referred documents give a clear picture of the design.

If it helps to substantiate your description of an individual block, describe the verification methods that you used to provide evidence that demonstrates that the design performs the intended function. Simulator outputs, oscilloscope traces, and logic analyzer timing diagrams are all suitable for this purpose, whether they are for the entire circuit or individual blocks. Also indicate whether simulation or physical implementation and testing were the primary verification methods, and discuss the extent and robustness of verification activities.

## Discussion

### Problems Encountered

Describe any significant problems encountered along the way, and their eventual solution (or lack thereof). If problems were encountered, point out any alternative solutions that were examined and rejected along the way, so that others following in your footsteps don't expend needless effort. It is also important to apply hindsight to your design, and to document your views so that others can benefit from the lessons you learned during the project. For example, if one alternative design path was chosen for implementation, but it became clear later on that a better solution was possible, this should be clearly stated with adequate explanation.

If the project design is not complete, or the design did not meet the original requirements, state clearly what is incomplete or incorrect, why it is not finished, and what resources will be required to get it finished. If the project is only a single phase in a multiphase project, then state that in this section, and describe in one or two sentences just what the next project entails (e.g., "this project represents only the display controller portion of the I/O system. It will be included as a module in the I/O system at a later date.")

### Engineering Resources Used

Provide a detailed account of the engineering resources that were used during the course of the design. At a minimum, define the number of engineering hours required, along with the computing, CAD, and test and measurement equipment resources/hours that were required.

## References

This crucial section provides references for any source files or other materials not incorporated into your report.

- Include a list all electronic design tools used, the computer platforms on which they were used, and their revision levels.
- Include a list of all persons who contributed to the design, and what their contributions were.
- Attach copies (as appendices perhaps) of all schematics (not library parts, only the stuff that you designed), VHDL code, simulator input files, and, where appropriate, simulator outputs.
- Include references to any data sheets or other source documents used in the project.

---

## Appendix A: Presentation Guidelines

The project presentation is the time to show off your design and impress your audience. Pretend that you are making a business presentation to venture capitalists.

A PowerPoint® presentation and a demonstration of your project makes an ideal presentation.

The presentation should include slides with top-level information from each of the major sections of your report. Don't over-formalize the slides, and don't dwell on your design methods. For an excellent example of a PowerPoint® presentation, see the *DigitalSynth Presentation* available from the Design Contest section of [www.digilent.ro](http://www.digilent.ro).

The presentation should strike a balance between a sales presentation and technical concerns. Remember that the report is the place for technical detail. The presentation needs only to highlight technical considerations, but be prepared to discuss technical details if asked. The emphasis of the presentation is the demonstration of how your project works.

Don't be long-winded, but feel free to be creative (e.g., include interesting pictures and anecdotes). Discuss your experience doing this project and anything else you think is relevant.

Conclude with an honest evaluation of the project, including expectations, how the design developed, issues faced, what could have been done better, and what went well and what didn't go well.

Above all, relax and have fun with your presentation. We know that you're nervous about it, but this is not a trial. It is a meeting with your friends from Diligent who want to give away some fabulous prizes and then get together with you for a cocktail reception.

## Appendix B: VHDL Code Sample

There are two main goals for the source file layout style and commenting. The first is to make the code easy to read and understand. The second is to describe how it works so that someone else reading the code doesn't have to infer its operation.

Each source file module should have a header block that provides information about the module. This should include author, description of the purpose of the module and its contents and how it relates to the rest of the project, and a revision history giving a chronology and description of major changes to the module.

Declarations outside of the implementation should be placed near the top of the module and grouped into logically related sets of declarations, with comments describing these groups.

Within the implementation section, the circuit descriptions should be divided into logically-related sections, with comments giving titles to these sections. Within each description, there should be comments describing the how the circuit works.

Indentation should be used to indicate the logical scope of the statements within the various logical constructs.

```
-----
--      modio.vhd -- Digilab Digilab AM188 CPLD Configuration
-----
-- Author:  Gene Apperson
--          Copyright 2003 Digilent, Inc.
-----
-- This module contains the standard configurations for the CPLD on
-- the Digilab AM188 board. This CPLD implements the i/o port logic
-- for connecting the AM188 bus to the module signals on the interface
-- connectors C1 and C2. This allows access to module peripheral boards
-- from the AM188 cpu.
--
-----
-- Revision History:
-- 07/21/2003(GeneA): created
-----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ModIO is
  Port ( dbBus      : inout std_logic_vector(7 downto 0);
        adrBus     : in std_logic_vector(5 downto 0);
        wrBus      : in std_logic;
        rdBus      : in std_logic;
        csBus      : in std_logic;
        int        : out std_logic;
        reset      : in std_logic;
        dbModA     : inout std_logic_vector(7 downto 0);
        astbModA   : in std_logic;
        dstbModA   : in std_logic;
        wrModA     : in std_logic;
        wrstModA   : out std_logic;
        rdstModA   : out std_logic;
        dbModB     : inout std_logic_vector(7 downto 0);
        astbModB   : in std_logic;
        dstbModB   : in std_logic;
        wrModB     : in std_logic;
        wrstModB   : out std_logic;
        rdstModB   : out std_logic;
```

```

        dbPa      : inout std_logic_vector(7 downto 0);
        dbPb      : inout std_logic_vector(7 downto 0));
end ModIO;

```

architecture Behavioral of ModIO is

```

-----
-- Component Declarations
-----

```

```

-----
-- Signal Declarations
-----

```

```

signal  rgbCpuDin      : std_logic_vector(7 downto 0); -- data bus from host CPU
signal  rgbCpuDout     : std_logic_vector(7 downto 0); -- data bus to host CPU
signal  rgbDbbaOut    : std_logic_vector(7 downto 0);
signal  rgbDbbbOut    : std_logic_vector(7 downto 0);

signal  adrReg        : std_logic_vector(2 downto 0);
signal  csl           : std_logic;

-- Registers writable from the host CPU
signal  regStat       : std_logic_vector(7 downto 0);
signal  regCtrl       : std_logic_vector(7 downto 0);
signal  regCpuAdrA    : std_logic_vector(7 downto 0);
signal  regCpuDataA   : std_logic_vector(7 downto 0);
signal  regCpuAdrB    : std_logic_vector(7 downto 0);
signal  regCpuDataB   : std_logic_vector(7 downto 0);
signal  regGpioA      : std_logic_vector(7 downto 0);
signal  regGpioB      : std_logic_vector(7 downto 0);

-- Registers writable from the module ports
signal  regModAdrA    : std_logic_vector(7 downto 0);
signal  regModDataA   : std_logic_vector(7 downto 0);
signal  regModAdrB    : std_logic_vector(7 downto 0);
signal  regModDataB   : std_logic_vector(7 downto 0);

-- Status bits indicating state of contents of data registers
signal  stCpuDataA    : std_logic;    -- set to 1 when data is written to regCpuDataA
signal  stCpuDataB    : std_logic;    -- set to 1 when data is written to regCpuDataB
signal  stCpuAdrA     : std_logic;    -- set to 1 when data is written to regCpuAdrA
signal  stCpuAdrB     : std_logic;    -- set to 1 when data is written to regCpuAdrB
signal  stModDataA    : std_logic;    -- set to 1 when data is written to regModDataA
signal  stModDataB    : std_logic;    -- set to 1 when data is written to regModDataB
signal  stModAdrA     : std_logic;    -- set to 1 when data is written to regModAdrA
signal  stModAdrB     : std_logic;    -- set to 1 when data is written to regModAdrB

signal  oeGpioA       : std_logic;    -- GPIO A output enable
signal  oeGpioB       : std_logic;    -- GPIO B output enable
signal  ienModB       : std_logic;
signal  ienModA       : std_logic;
signal  oenModB       : std_logic;    -- output mode enable for module B signals
signal  oenModA       : std_logic;    -- output mode enable for module A signals

-- Write strobes for updating output device registers
signal  selStat       : std_logic;
signal  selCtrl       : std_logic;
signal  selAdrA       : std_logic;
signal  selDataA      : std_logic;
signal  selAdrB       : std_logic;
signal  selDataB      : std_logic;
signal  selGpioA      : std_logic;
signal  selGpioB      : std_logic;

-- Misc internal signals
signal  irqModA       : std_logic;
signal  irqModB       : std_logic;

```

```

-----
--                               Module Implementation
-----

```

```

begin

```

```

-- General Combinational Logic
-----

csl <= '0' when csBus = '0' and adrBus(5 downto 3) = "110" else '1';
adrReg <= adrBus(2 downto 0);

-- Decode the chip select and address to generate a select
-- signal for register access from the processor bus side.
selStat <= '1' when csl = '0' and adrReg = "000" else '0';
selCtrl <= '1' when csl = '0' and adrReg = "001" else '0';
selAdrA <= '1' when csl = '0' and adrReg = "010" else '0';
selDataA <= '1' when csl = '0' and adrReg = "011" else '0';
selAdrB <= '1' when csl = '0' and adrReg = "100" else '0';
selDataB <= '1' when csl = '0' and adrReg = "101" else '0';
selGpioA <= '1' when csl = '0' and adrReg = "110" else '0';
selGpioB <= '1' when csl = '0' and adrReg = "111" else '0';

-----

-- Map internal signal to i/o pins
-----

rgbCpuDin <= dbBus;
dbBus <= rgbCpuDout when (csl = '0' and rdBus = '0') else "ZZZZZZZ";

regStat(0) <= stModDataA;
regStat(1) <= stCpuDataA;
regStat(2) <= stModAdrA;
regStat(3) <= stCpuAdrA;
regStat(4) <= stModDataB;
regStat(5) <= stCpuDataB;
regStat(6) <= stModAdrB;
regStat(7) <= stCpuAdrB;

wrstModA <= stModDataA or stModAdrA;
rdstModA <= stCpuDataA;

wrstModB <= stModDataB or stModAdrB;
rdstModB <= stCpuDataB;

-- Select cpu read data based on address.
with adrReg select
  rgbCpuDout <=
    regStat      when "000",
    regCtrl      when "001",
    regModAdrA   when "010",
    regModDataA  when "011",
    regModAdrB   when "100",
    regModDataB  when "101",
    dbPa         when "110",
    dbPb         when "111",
    "-----"    when others;

-- Module data input/output busses
dbModA <= rgbDbaOut when wrModA = '1' else "ZZZZZZZ";
rgbDbaOut <= regCpuAdrA when astbModA = '0' else regCpuDataA;

dbModB <= rgbDbbOut when wrModB = '1' else "ZZZZZZZ";
rgbDbbOut <= regCpuAdrB when astbModB = '0' else regCpuDataB;

-- General purpose i/o port data busses
dbPa <= regGpioA when oeGpioA = '1' else "ZZZZZZZ";
dbPb <= regGpioB when oeGpioB = '1' else "ZZZZZZZ";

-- Interrupt request to host CPU
irqModA <= '1' when (ienModA = '1') and (stModDataA = '1' or stModAdrA = '1')
           else '0';
irqModB <= '1' when (ienModB = '1') and (stModDataB = '1' or stModAdrB = '1')
           else '0';
int <= irqModA or irqModB;

-----

-- Registers Writable from the Module Side
-----

-- Module A address register and address ready status bit
process (astbModA, reset, rdBus)
begin
  if (reset = '0') or (rdBus = '0' and selAdrA = '1') then

```

```

        stModAdrA <= '0';
    else
        if astbModA = '1' and astbModA'Event then
            if wrModA = '0' then
                regModAdrA <= dbModA;
                stModAdrA <= '1';
            end if;
        end if;
    end if;
end process;

-- Module A Data Register
process (dstbModA)
begin
    if (oenModA = '0') or (wrModA = '0' and dstbModA = '0') then
        regModDataA <= dbModA;
    end if;
end process;

-- Module A data ready status bit
process (dstbModA, reset, rdBus)
begin
    if (reset = '0') or (rdBus = '0' and selDataA = '1') then
        stModDataA <= '0';
    else
        if dstbModA = '1' and dstbModA'Event then
            if wrModA = '0' then
                stModDataA <= '1';
            end if;
        end if;
    end if;
end process;

-- Module B Address Register
process (astbModB, reset, rdBus)
begin
    if (reset = '0') or (rdBus = '0' and selAdrB = '1') then
        stModAdrB <= '0';
    else
        if astbModB = '1' and astbModB'Event then
            if wrModB = '0' then
                regModAdrB <= dbModB;
                stModAdrB <= '1';
            end if;
        end if;
    end if;
end process;

-- Module B Data Register
process (dstbModB)
begin
    if (oenModB = '0') or (wrModB = '0' and dstbModB = '0') then
        regModDataB <= dbModB;
    end if;
end process;

-- Module B data ready status bit
process (dstbModB, reset, rdBus)
begin
    if (reset = '0') or (rdBus = '0' and selDataB = '1') then
        stModDataB <= '0';
    else
        if dstbModB = '1' and dstbModB'Event then
            if wrModB = '0' then
                stModDataB <= '1';
            end if;
        end if;
    end if;
end process;

-----
-- Registers Writable from the Processor Bus Side
-----

-- Control Register
process (wrBus, reset)

```

```

begin
  if reset = '0' then
    oeGpioB <= '0';
    oeGpioA <= '0';
    ienModB <= '0';
    ienModA <= '0';
    oenModB <= '0';
    oenModA <= '0';
  else
    if wrBus = '1' and wrBus'Event then
      if selCtrl <= '1' then
        oeGpioA <= rgbCpuDin(7);
        oeGpioB <= rgbCpuDin(6);
        ienModB <= rgbCpuDin(3);
        ienModA <= rgbCpuDin(2);
        oenModB <= rgbCpuDin(1);
        oenModA <= rgbCpuDin(0);
      end if;
    end if;
  end if;
end process;

-- Host Processor Address Register A
process (wrBus, astbModA, reset)
begin
  if (reset = '0') or (wrModA = '1' and astbModA = '0') then
    stCpuAdra <= '0';
  else
    if wrBus = '1' and wrBus'Event then
      if selAdra = '1' then
        regCpuAdra <= rgbCpuDin;
        stCpuAdra <= '1';
      end if;
    end if;
  end if;
end process;

-- Host Processor Data Register A
process (wrBus, dstbModA, reset)
begin
  if (reset = '0') or (wrModA = '1' and dstbModA = '0') then
    stCpuDataA <= '0';
  else
    if wrBus = '1' and wrBus'Event then
      if selDataA = '1' then
        regCpuDataA <= rgbCpuDin;
        stCpuDataA <= '1';
      end if;
    end if;
  end if;
end process;

-- Host Processor Address Register B
process (wrBus, astbModB, reset)
begin
  if (reset = '0') or (wrModB = '1' and astbModB = '0') then
    stCpuAdrb <= '0';
  else
    if wrBus = '1' and wrBus'Event then
      if selAdrb = '1' then
        regCpuAdrb <= rgbCpuDin;
        stCpuAdrb <= '1';
      end if;
    end if;
  end if;
end process;

-- Host Processor Data Register B
process (wrBus, dstbModB, reset)
begin
  if (reset = '0') or (wrModB = '1' and dstbModB = '0') then
    stCpuDataB <= '0';
  else
    if wrBus = '1' and wrBus'Event then
      if selDataB = '1' then
        regCpuDataB <= rgbCpuDin;
        stCpuDataB <= '1';
      end if;
    end if;
  end if;
end process;

```

```
        end if;
    end if;
end if;
end process;

-- GPIO Port A Register
process (wrBus)
begin
    if wrBus = '1' and wrBus'Event then
        if selGpioA = '1' then
            regGpioA <= rgbCpuDin;
        end if;
    end if;
end process;

-- GPIO Port B Register
process (wrBus)
begin
    if wrBus = '1' and wrBus'Event then
        if selGpioB = '1' then
            regGpioB <= rgbCpuDin;
        end if;
    end if;
end process;

-----
end Behavioral;
```