RGB LED Blink on VSDSquadron FM Board

Introduction

This project involves the implementation of an RGB LED blink functionality on the VSDSquadron FM board using Verilog HDL. The VSDSquadron FM board is a versatile and compact FPGA development platform designed for embedded systems, IoT applications, and hardware prototyping. It features a Lattice ICE40UP5K FPGA, which is known for its low power consumption and high performance, making it an ideal choice for projects requiring efficient hardware control and signal processing.

The primary objective of this project is to demonstrate the control of an RGB LED using Verilog HDL. The RGB LED is driven by its red, green, and blue channels, each controlled with specific timing and intensity to create a blinking effect. The design leverages the internal oscillator of the FPGA and a frequency counter to generate precise clock signals, ensuring accurate timing for the LED blink patterns.

Verilog Code Review

The Verilog code is structured as a single module that interfaces with the hardware components of the VSDSquadron FM board. Below is a breakdown of the module and its functionality.

Module Declaration

The module is declared with the following input and output ports:

```
module top (
        output wire led_red ,
        output wire led_blue ,
        output wire led_green ,
       input wire hw_clk,
        output wire testwire
);
```

Output Ports :

- **output wire led_red**

- **Purpose:** This output port controls the red channel of the RGB LED.

- **output wire led_blue**
  - **Purpose:** This output port controls the blue channel of the RGB LED.

- **output wire led_green**
  - **Purpose:** This output port controls the green channel of the RGB LED.

- **output wire testwire**
  - **Purpose:** This is a test signal output used for debugging or verification purposes.

Input Ports :

- **input wire hw_clk**
  - **Purpose:** This is the hardware oscillator clock input, which serves as the primary clock signal for the design.

Variable Declaration

The provided code snippet declares two variables:

```
wire int_osc;
reg [27:0] frequency_counter_i;
```

These variables play a critical role in the internal logic of the Verilog module, particularly in generating the clock signal and controlling the timing for the RGB LED blink functionality. Below is a detailed explanation of each variable and its purpose:

- **- wire int_osc**

  Purpose: This is a signal wire used to connect the output of the internal oscillator (SB_HFOSC) to the rest of the design. Functionality :

  - The int_osc wire carries the clock signal generated by the internal oscillator (SB_HFOSC).
  - This signal serves as the primary clock for the frequency counter and other timing-related logic in the design.

- **reg [27:0] frequency_counter_i**

  Purpose: This is a 28-bit counter register used to divide the high-frequency clock signal from the oscillator into a lower frequency suitable for controlling the RGB LED blink rate.

  Functionality:

  - The frequency_counter_i register increments on every clock cycle of the int_osc signal.

- Once the counter reaches its maximum value (2^28 - 1), it overflows and resets to zero, creating a periodic signal.
- The overflow signal or specific bits of the counter can be used to control the timing of the RGB LED blink. For example, the most significant bit (MSB) of the counter can toggle at a much lower frequency, creating a visible blink rate for the LED.

Counter Functionality :

```
assign testwire = frequency_counter_i[5];
always @(posedge int_osc) begin
        frequency_counter_i <= frequency_counter_i + 1'b1;
end
```

Purpose: This block defines the behavior of the frequency_counter_i register, which increments on every positive edge of the int_osc clock signal.

- **Functionality:**
  - The always block is triggered on the rising edge (posedge) of the int_osc signal, which is the clock signal generated by the internal oscillator (SB_HFOSC).
  - On each clock cycle, the frequency_counter_i register is incremented by 1 (1'b1).
  - Since frequency_counter_i is a 28-bit register, it will count from 0 to 2^28 - 1(268,435,455) and then overflow back to 0, creating a periodic signal.
- **Usage:**
  - The counter is used to divide the high-frequency int_osc clock into a lower-frequency signal.
  - Specific bits of the counter (e.g., frequency_counter_i[27]) can be used to control the timing of the RGB LED blink or other time-dependent logic in the design.

Internal Oscillator

```
SB_HFOSC #(.CLKHF_DIV ("0b10")) u_SB_HFOSC (
   .CLKHFPU(1'b1),  // Power-up enable
   .CLKHFEN(1'b1),  // Clock enable
   .CLKHF(int_osc)  // Output clock signal
);
```

This code segment instantiates an internal high-frequency oscillator (SB_HFOSC) available in Lattice FPGAs. It generates a clock signal (int_osc) for use in the design.

- **SB_HFOSC** → Lattice-provided internal oscillator module.
- **#(.CLKHF_DIV ("0b10"))** → Sets the clock division factor ("0b10" = divide by **4**).
- .**CLKHFPU(1'b1)** → Powers up the oscillator.
- .**CLKHFEN(1'b1)** → Enables the oscillator.
- **.CLKHF(int_osc)** → Outputs the generated clock signal (int_osc).

Instantiation of RGB Primitive

This section of the code instantiates an RGB driver module (SB_RGBA_DRV) that controls an RGB LED. Let's break down the code to understand how the RGB LED is being controlled:

SB_RGBA_DRV Instantiation:

```
SB_RGBA_DRV RGB_DRIVER (
   .RGBLEDEN(1'b1),
   .RGB0PWM (1'b0), // red
   .RGB1PWM (1'b0), // green
   .RGB2PWM (1'b1), // blue
   .CURREN (1'b1),
   .RGB0 (led_red), // Actual Hardware connection
   .RGB1 (led_green),
   .RGB2 (led_blue)
);
```

This is the instantiation of the SB_RGBA_DRV module, which is used to control an RGB LED. Here's what each parameter means:

- .RGBLEDEN(1'b1): This enables the RGB LED. When this is set to 1'b1, the RGB driver is active, and the LEDs can be controlled.
- .RGB0PWM(1'b0): This parameter controls the red LED (RGB0). A value of 1'b0 means the red LED will either be fully off or will not be dynamically pulsed (based on PWM control). The value 1'b0 here indicates that no pulse-width modulation (PWM) is applied to the red LED, and it is either fully on or off.
-
-

- .RGB1PWM(1'b0): This parameter controls the green LED (RGB1). Similar to the red LED, the green LED is controlled by this PWM signal. A value of 1'b0 means the green LED is also either fully off or without PWM control.
- .RGB2PWM(1'b1): This parameter controls the blue LED (RGB2). Setting this to 1'b1 means that the blue LED will always be on (since no PWM is being applied to control brightness). In a PWM-controlled system, this would typically mean the LED is driven continuously at full brightness.
- .CURREN(1'b1): This parameter is used to enable the current driving capability for the RGB LEDs. Setting it to 1'b1 ensures that the current is supplied to the RGB LEDs, allowing them to be powered and illuminated.
- .RGB0(led_red), .RGB1(led_green), .RGB2(led_blue): These are the actual hardware connections for the red, green, and blue LEDs. These signals (led_red, led_green, led_blue) are connected to the physical pins that control the RGB LEDs.

defparam Statements:

```
defparam RGB_DRIVER.RGB0_CURRENT = "0b000001";
defparam RGB_DRIVER.RGB1_CURRENT = "0b000001";
defparam RGB_DRIVER.RGB2_CURRENT = "0b000001";
```

These defparam statements set the current for each of the RGB LEDs. The current values are typically defined in terms of 6-bit values to select different levels of brightness or current strength. Here, all three LEDs (red, green, and blue) are assigned a current value of 0b000001, which is a low value, indicating low current (and possibly low brightness) for each LED.

Final Observations from Verilog Code

This Verilog code demonstrates the way to control an RGB LED and generate a blinking effect using an FPGA. The internal oscillator provides the clock signal, the frequency counter creates a periodic signal, and the RGB driver controls the LED states. By adjusting the counter size or the clock division factor, the blinking frequency can be modified. This code serves as a foundational example for more complex LED control and timing applications in FPGA designs.

Creation of pcf file

PCF stands for “Physical Constraints File”. The PCF is a critical component in FPGA design, and for the projects on VSDSquadron FM board. It defines how the logical signals in your design are mapped to the physical pins of the FPGA.

Need for pcf file

- The VSDSquadron FM board has specific GPIO pins connected to the RGB LED. To control the LED, you need to tell the FPGA which physical pins correspond to the red, green, and blue channels of the LED.
- Without a PCF file, the FPGA tools won't know how to map your design's logical signals (e.g., `led_red`, `led_green`, `led_blue`) to the actual hardware pins.

Purpose of PCF File

- **Pin Mapping**: The PCF file specifies the exact FPGA pins connected to the RGB LED on the board. For example:
    - Pin for the red LED channel
    - Pin for the green LED channel
    - Pin for the blue LED channel
- **Hardware-Specific Configuration**: It ensures your design aligns with the VSDSquadron FM board's hardware layout, preventing errors like incorrect signal routing or pin conflicts.

PCF file for led blink

Below is the pcf file defined for the led blink project on VSD Squadron FM board:

```
set_io led_red 39
set_io led_blue 40
set_io led_green 41
set_io hw_clk 20
set_io testwire 17
```

Here's a brief overview of the pin assignments and their significance:

Signal	FPGA Pin	Function
led_red	39	Controls the red channel of the onboard RGB LED.
led_blue	40	Controls the blue channel of the onboard RGB LED.
led_green	41	Controls the green channel of the onboard RGB LED.
hw_clk	20	Receives the external hardware clock signal for FPGA timing.
testwire	17	General-purpose I/O pin for debugging or additional interfacing needs.

Significance in Verilog Code and Board Hardware:

- **RGB LED Control (led_red, led_blue, led_green)**:
  In the Verilog code, these signals are defined as outputs and connected to the RGB primitive (SB_RGBA_DRV). The assignments in the PCF file map them to pin 39 (red), pin 40 (blue), and pin 41 (green). The SB_RGBA_DRV module enables control of LED brightness and color, where RGB2PWM is set to 1 (blue on), and RGB0PWM and RGB1PWM are set to 0 (red and green off).
- **Hardware Clock (hw_clk)**:
  Assigned to pin 20, this external clock is declared as an input in Verilog. However, the provided code does not use hw_clk directly; instead, it employs an internal oscillator (SB_HFOSC) to drive the frequency counter.
- **Test Wire (testwire)**:
  Assigned to pin 17, this signal outputs a divided clock derived from the internal oscillator. The code assigns testwire to frequency_counter_i[5], making it a low-frequency signal useful for debugging or external monitoring.

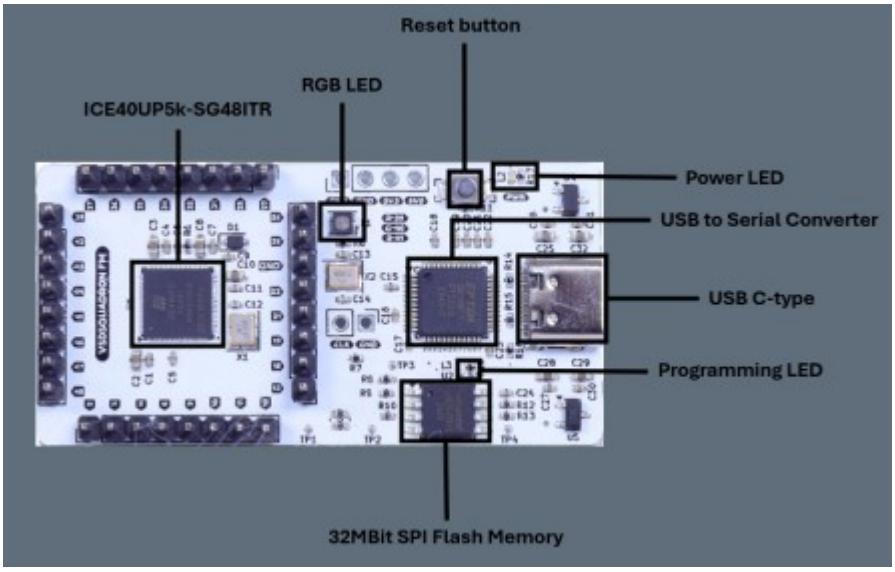
These assignments align with the VSDSquadron FM board’s hardware configuration, ensuring correct functionality and avoiding conflicts.

Integration with VSD Squadron FPGA Mini board

The **VSDSquadron FM** is a compact **FPGA development board** built around the **Lattice iCE40UP5K FPGA**, designed for **low-power, high-efficiency embedded applications**. It is widely used in digital logic design, IoT, and hardware prototyping due to its small form factor and rich set of features.

VSD Squadron FPGA Mini board features

- 48-lead QFN package
- 5.3K LUTs for flexible logic design
- 1Mb SPRAM and 120Kb DPRAM for efficient memory usage
- Onboard FTDI FT232H USB-to-SPI interface for programming and communication
- All 32 FPGA GPIO accessible for rapid prototyping
- Integrated 4MB SPI flash for configuration and data storage
- RGB LED for user-defined signaling
- Onboard 3.3V and 1.2V power regulators, with the ability to supply 3.3V externally



Development Tools for VSDSquadron FM

1. Project IceStorm

Project IceStorm is an **open-source toolchain** for Lattice iCE40 FPGAs. It enables bitstream generation, allowing users to program FPGAs without proprietary tools.

- **Components**: `icepack` (bitstream packer), `iceprog` (flasher), `icetime` (timing analysis).
- **Purpose**: Converts synthesized netlists into bitstreams for FPGA configuration.

2. Yosys

Yosys is an **open-source synthesis tool** for Verilog designs. It translates HDL code into a gate-level representation.

- **Purpose**: Logic synthesis, optimization, and technology mapping for FPGAs..

3. NextPNR

NextPNR is a **place-and-route tool** that maps synthesized logic onto FPGA resources. It supports multiple FPGA architectures, including iCE40.

- **Purpose**: Assigns logic gates to physical FPGA locations and routes interconnections.
- **Key Features**: Timing-driven placement, constraint management, and multi-architecture support.

These tools collectively form a fully open-source FPGA design flow, enabling efficient FPGA development.

Automating FPGA Workflow using Makefile

A Makefile is a script used for automating repetitive tasks in software and hardware development. It simplifies project compilation, synthesis, place-and-route, bitstream generation, and flashing the FPGA. Instead of manually executing multiple commands, a Makefile ensures a structured and efficient workflow.

Purpose of Makefile in this Project

The Makefile automates the entire FPGA development process for the VSD Squadron FPGA Mini board. It handles the following tasks:

- Synthesis of Verilog code using Yosys
- Placement and routing with NextPNR
- Timing analysis using Icetime
- Bitstream generation with Icepack
- Flashing the FPGA using Iceprog
- Serial communication setup via Picocom