Setting up ESP-IDF for ESP32 Development

ESP-IDF (**Espressif IoT Development Framework**) is the official development framework for ESP32 and ESP32-S series microcontrollers. It is specifically designed to work with ESP32 devices and provides APIs, tools, and libraries to simplify development.

Steps to Set Up ESP-IDF

1. **Install Prerequisites**

   - For **Linux/Mac**: Install dependencies via terminal:

     ```
     bash
     Copy code
     sudo apt-get install git wget flex bison gperf python3 python3-pip
     cmake ninja-build ccache libffi-dev libssl-dev dfu-util
     ```

   - For **Windows**: Download and install Python 3, Git, and other dependencies.

2. **Download ESP-IDF**

   - Clone the ESP-IDF repository:

     ```
     bash
     Copy code
     git clone --recursive https://github.com/espressif/esp-idf.git
     ```

   - Navigate to the ESP-IDF directory:

     ```
     bash
     Copy code
     cd esp-idf
     ```

3. **Set Up Environment**

   - Run the setup script:

     ```
     bash
     Copy code
     ./install.sh
     ```

   - Add ESP-IDF tools to your environment:

     ```
     bash
     Copy code
     . ./export.sh
     ```

4. **Create a New Project**

   - Use the provided template to create a new project:

     ```
     bash
     Copy code
     cp -r examples/get-started/hello_world ~/esp32_projects/hello_world
     cd ~/esp32_projects/hello_world
     ```

5. **Build and Flash**

   - Configure the project:

```
bash
Copy code
idf.py menuconfig
```

- Build the project:

```
bash
Copy code
idf.py build
```

- Flash the firmware to your ESP32:

```
bash
Copy code
idf.py flash
```

- Monitor the serial output:

```
bash
Copy code
idf.py monitor
```

FPGA Programming Sequence

The standard sequence to program an FPGA:

1. **Assert PROG_B**: Reset the FPGA.
2. **Send Bitstream**: Provide the bitstream file through the `DATA0` pin, synchronized with the clock (`CCLK`).
3. **Check DONE Pin**: Verify that the FPGA is successfully configured.

---

1. Code to Upload Bitstream

Here is the complete **ESP-IDF-based code** to upload a bitstream stored on an SD card to the FPGA:

2. Explanation of the Code

Key Points

1. **GPIO Configuration**:
   - `FPGA_PROG_B_PIN`: Resets the FPGA to prepare it for configuration.
   - `FPGA_DATA0_PIN`: Sends data bits to the FPGA.
   - `FPGA_CCLK_PIN`: Toggles the clock to synchronize data transfer.
   - `FPGA_DONE_PIN`: Monitors the FPGA's configuration status.
2. **Bitstream Upload**:

    - Reads the bitstream file byte by byte from the SD card.
    - Sends each bit of the byte to the FPGA via `DATA0`, synchronized with `CCLK`.
3. **SD Card Handling**:
    - Mounts the SD card to read the bitstream file.
    - Unmounts the SD card after programming.
4. **Error Handling**:
    - Ensures the file is accessible and the `DONE` pin confirms a successful configuration.

---

3. Hardware Setup

Ensure the following connections between the ESP32 and FPGA:

- **GPIO22 (FPGA_PROG_B_PIN) → PROG_B Pin on FPGA**
- **GPIO23 (FPGA_CCLK_PIN) → CCLK Pin on FPGA**
- **GPIO25 (FPGA_DATA0_PIN) → DATA0 Pin on FPGA**
- **GPIO21 (FPGA_DONE_PIN) → DONE Pin on FPGA**

Additionally:

- Insert an SD card containing the bitstream file (`bitstream.bin`) into the SD card slot on the board.

4. Build and Flash

1. Open your ESP-IDF terminal.
2. Configure your project:

    ```
    bash
    Copy code
    idf.py menuconfig
    ```

    - Ensure the correct pins and SD card slot are configured.
3. Build the project:

    ```
    bash
    Copy code
    idf.py build
    ```

4. Flash the firmware to your ESP32:

    ```
    bash
    Copy code
    idf.py flash
    ```

5. Monitor the serial output:

    ```
    bash
    Copy code
    idf.py monitor
    ```

---

5. Output

You should see output similar to:

```
arduino
Copy code
SD card mounted successfully.
Starting FPGA configuration...
FPGA configuration successful!
FPGA programming completed successfully.
SD card unmounted.
```

Next Step

This code **only uploads the bitstream** to the FPGA. Once this is working:

1. Add functionality for SPI or UART communication with the FPGA.
2. Test the communication post-configuration.