IP Generators - A Better Reuse Methodology

Amanjyot Kaur, Agnisys
Problem Statement

● Designers spend a huge amount of time creating standard IPs.
  ○ These IPs are very rigid and brittle at the same time.
● To control the flow and reuse of such IPs, engineers use a parameter-based flow,
  ○ but they generally break when used in a different environment because changes in a port list or customization of these IPs is difficult.
● Also, when the RTL changes, it becomes of utmost importance to vary the corresponding application programming interfaces (APIs) and sequences already created for it.
Proposed Methodology

• Configuring the IPs using parameters:
  o Generate time
  o Instance/elaboration time
• Customizations required, such as additional fields or registers added to the register map (regmap) of the IP
• Creating interconnections between different IPs
• Automatic generation of configuration APIs in UVM and C format
• Creation of test sequences for different platforms like firmware, validation, verification, and Automatic Test Equipment (ATE)
• The choice of the bus used to access the IP, such as, AHB, APB, AXI, etc, is abstracted out, to avoid the design becoming too brittle
Implementation Details

- General Purpose Input/Output (GPIO)
- Advanced Encryption Standard (AES)
- Programmable Interrupt Controller (PIC)
- Serial Peripheral Interface (SPI)
- Pulse Width Modulation (PWM)
- Direct Memory Access (DMA)
- Inter-Integrated Circuit (I2C)
- Integrated Inter-IC Sound Bus (I2S)
- Universal Asynchronous Receiver/Transmitter (UART)
- Timer

<table>
<thead>
<tr>
<th>IP generator approach</th>
<th>Parameterizable IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>Since IPs are being created on the fly at generation time, there is additional flexibility in controlling how to create the IP</td>
</tr>
<tr>
<td>Ability to change ports</td>
<td>Yes</td>
</tr>
<tr>
<td>Ability to add registers or fields to the IP’s regmap</td>
<td>Yes</td>
</tr>
<tr>
<td>Impact on other aspects of IPs</td>
<td>Along with RTL, design verification environment, firmware, software, and documentation can also be generated</td>
</tr>
<tr>
<td>Development resources required</td>
<td>Lower</td>
</tr>
</tbody>
</table>

Table 1. Comparison of IP generator approach and parameterizable approach

Figure 1. Block Diagram of IP Generators
Application

- Trigger Word Detector (TWD)
- Design can detect one of the four unique words in the audio sample and then drive the appropriate LED, LED0-LED3, using the GPIO interface
- If none of the words is detected, LED4 glows
- Each set of weights is trained to detect four unique words
- mem1_csr block is connected to a dual port memory
- On pressing the button[0] the first set of weights is loaded by the DMA into the sample regmap from the mem2_csr
- The neural net logic uses these weights and input audio sample spectrogram values to generate an output which in turn drives the LEDs
- Similarly, on pressing button[1] the second set of weights is loaded
Figure 3. Generated Verilog and UVM for regmap
Result

Figure 4. HTML and Aggregation Logic of TWD

```vhdl
module ids_top_csr_apb_aggregation(
    clk, //Bus clock
    reset, //Reset
    sel, //Select : It indicates that the slave device is selected and a data transfer is in progress.
    enable, //Enable : This signal indicates the second and subsequent cycles of a pwrite.
    prot, //Protection type : This signal indicates the common, privileged, or secure address range.
    wdata, //Write strobe : This signal indicates which byte lanes to update during a pwrite.
    addr, //Address bus
    rdy, //Ready : The slave uses this signal to extend an APB transfer.
    wdata, //Read data
    werror, //Error signal : This signal indicates a transfer failure.
);

begin
    // please note that this is just a simple example
end
```

HTML

---

**Figure 4. HTML and Aggregation Logic of TWD**
Figure 5. Sample of Configuration APIs of GPIO in C and UVM
### Result

IP related generated files (RTL) with number of lines

<table>
<thead>
<tr>
<th>S.No.</th>
<th>File Name</th>
<th>Line of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Configuration Bus File</strong></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>apb_widget.v</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td><strong>GPIO IP (Config bus : APB, Number of sources : 2, Number of outputs : 5)</strong></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>sync_ffiv</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>edge_detect.v</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>gpioo_top.v</td>
<td>352</td>
</tr>
<tr>
<td>5</td>
<td>gpio.v</td>
<td>743</td>
</tr>
<tr>
<td></td>
<td><strong>DMA IP (Config bus : APB, Bus type : APB, Number of channels : 2)</strong></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>dma_regmap_orbitor.v</td>
<td>227</td>
</tr>
<tr>
<td>7</td>
<td>dma_sob_master.v</td>
<td>132</td>
</tr>
<tr>
<td>8</td>
<td>dma_regmap_core.v</td>
<td>407</td>
</tr>
<tr>
<td>9</td>
<td>dma_regmap_tkn.v</td>
<td>379</td>
</tr>
<tr>
<td>10</td>
<td>fifo.v</td>
<td>102</td>
</tr>
<tr>
<td>11</td>
<td>dma.v</td>
<td>742</td>
</tr>
<tr>
<td></td>
<td><strong>I2S IP (Config Bus: APB, Interrupt generation with enable)</strong></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>i2s_master.v</td>
<td>310</td>
</tr>
<tr>
<td>13</td>
<td>preselerv</td>
<td>82</td>
</tr>
<tr>
<td>14</td>
<td>i2s_csr_block.v</td>
<td>970</td>
</tr>
<tr>
<td>15</td>
<td>tnx_fifo.v</td>
<td>116</td>
</tr>
<tr>
<td>16</td>
<td>i2s_core.v</td>
<td>521</td>
</tr>
</tbody>
</table>
Conclusion

● Fully configurable
● Easily customizable
● IPs are generated from the command line or on a click of a button
● Unencrypted code
● Ability to target multiple platforms (Design, Verification, Firmware, Software, and Documentation)
● For about 10 different IPs, and about 50 different instantiations, users can easily generate 100,000 lines of RTL code, UVM, and C test environment
Questions