**HARDWARE IMPLEMENTATION OF INTERLEAVER IN TURBO DECODER**

**OBJECTIVES**

Architecture had been proposed in [1] to solve the Interleaver Bottleneck problem in Turbo Decoder with parallel decoding unit. This architecture had to be represented by a Generic VHDL model that could be synthesised. A Generic model is a parameterised model to obtain implementation variants. A performance evaluation of the synthesized hardware had to be carried out.

**TASKS CARRIED OUT**

- Designing the controls for a Fixed and a Generic Architecture
- Modelling the architectures with VHDL
- Simulating the Models using ModelSim
- Writing a Test Bench
- Synthesising the Models using Synopsys
- Obtaining synthesis reports for implementation variants by varying the parameters.

**INTRODUCTION**

Turbo Codes are forward error correcting (FEC) channel codes. The decoding is iterative with maximum-a-posteriori (MAP) decoders, interleavers and de-interleavers. Exchanging information between component decoders performs iterative decoding. High data rates can be achieved by using highly parallel architectures. Interleaving data in these architectures consumes enormous area and reduces throughput.

The proposed architecture uses multiple single port RAMs. The data that are produced in parallel are routed using the (de)interleaver table through a sorter to the respective RAMs. Due to the possible many-to-one mapping, these data have to be queued up through a buffer to reduce latency. Two architectures have been thought of. One has one buffer to queue data for each RAM. Another, a variant, has multiple buffers.

**DESIGNING THE CONTROLS FOR A FIXED ARCHITECTURE**

It was found simple to develop controls for the architecture with a fixed set of parameters and later extrapolate the design to develop a generic model. The block diagram in fig.1 shows the architecture for this.

![Fig 1. Block diagram for the fixed module](image)

The parallel sets of data from the producers are fed to the sorting units. The controls for the sorters are derived from the interleaver table entries through the I_leaver table unit. The sorter sends only the relevant data to the corresponding buffer. The buff_access unit writes the sent data in parallel to the registers in the buffer. It keeps track of the number of registers filled and reads out data one at a time. Num_access_generator unit counts the number of data that has to be written into each buffer. This information is required to keep track of the number of registers filled in buffers.
DESIGNING THE CONTROLS FOR A GENERIC ARCHITECTURE

The controls designed for the fixed VHDL model were extrapolated to obtain the generic model. Additional modules were added to take care of both ‘One Buffer per RAM’ and the ‘Multi Buffers per RAM’ architectures. Here is a brief on the modifications adapted in the coding style.

- The various parameters that had to be varied were defined as constants in the package body.
- These constants were used in defining the array ranges, loop ranges and at other places in the code.
- In the top level of the hierarchy the components were instantiated and interconnected using the generate statements supported by VHDL.

The logic of all the units was the same. Two new units were added to support “Multi-buffers per RAM architecture”. A `Count_holder` unit sends the address of the buffer that is filled the most as `buffer_to_be_read` signal. A `Write_RAM` unit selects one of the multi buffer outputs to be written into RAM. This unit adds another stage before the output RAMs.

Synthesis Results
The values of various parameters defined as constants in the package body are:

- Number of parallel producers in the decoding unit: 4
- Number of buffers for each RAM: 1
- Number of registers in each Buffer: 20
- Interleaver Table length: 128

The results are listed in table 1. The slack violation is mainly due to the buffer access unit.

<table>
<thead>
<tr>
<th>Modules</th>
<th>Area (sq. µm)</th>
<th>Timing*(ns)</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorter</td>
<td>31300</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Count Holder</td>
<td>2723</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write RAM</td>
<td>4918</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integration</td>
<td>1489126</td>
<td>9.44 – 13.66 = (-) 4.22 Slack violated</td>
<td></td>
</tr>
</tbody>
</table>

* Data required time – Data arrival time = Slack

Table 1. Brief of Area and Timing Reports

Analysis

Fig. 2 shows the design for this unit that was proposed in [1]. The multiplexer outputs are selected cyclically to write into selected registers. There was a need to have separate controls for selecting multiplexers and registers along with writing and reading out of the registers. The logic of this one unit was large.

The coding style was behavioural. This approach had too much of computation in the loops. This led to the implementation of long combinatorial logic that in turn reduced the maximum clock rate. Though the code depicted the required architecture functionally, the structure of Fig.2. was lost in the synthesised hardware.
Fig. 2. Buffer Access unit

**Modifications**
The following modifications were carried out:
- Performing the multiplexer and register control logic in a separate unit.
- Modifying code to make it more structural in style

**Synthesis results**
A sample of the synthesis results is presented in table 2. Values for Parameters are:
- Number of parallel producers in the decoding unit: 4
- Number of buffers for each RAM: 1
- Number of registers in each Buffer: 64
- Interleaver Table length: 512

<table>
<thead>
<tr>
<th>Modules</th>
<th>Area (sq. um) Before modification</th>
<th>Area (sq. um) After modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorter</td>
<td>23,679</td>
<td>23,679</td>
</tr>
<tr>
<td>Max buff control</td>
<td>-</td>
<td>28,356</td>
</tr>
<tr>
<td>Buffer access</td>
<td>642,455</td>
<td>265,178</td>
</tr>
<tr>
<td>Count Holder</td>
<td>3,542</td>
<td>3,542</td>
</tr>
<tr>
<td>Write RAM</td>
<td>3,443</td>
<td>3,443</td>
</tr>
<tr>
<td>Integration</td>
<td>2,411,613</td>
<td>1,321,257</td>
</tr>
</tbody>
</table>

Table 2. Results after final synthesis

**CONCLUSION**
The fixed and generic models were designed and coded. Test benches, to suit the generic model, were written. The synthesis and performance evaluation has been carried out. The work has to be continued using more design constraints.

**BIBLIOGRAPHY**