Velodyne Lidar Hamming Weight RTL Design
Project Report
Arvind Ramesh

Part 1 : Test Vector Generation :

Wrote a C program to generate the test vectors. The following is the outline of the program :

1. A master for loop whose runs is equal to the number of packets that are to be generated.
2. For each run of the master loop, a random number between 0 to 31 is generated which is the hamming weight of that packet. Let this number be "**number_bits**". This number is written to the **hw.txt** file.
3. A total of "**number_bits**" random numbers been 0 to 1023 are generated for the position of the 1's bit. These positions are set to "1" in the "**bit_vector**" array. These numbers are also stored in the "**bit_position**" array. The entries in the bit_position array are sorted and then written to the **position.txt** file.
4. The bit_vector is written to the the **bitpos.txt** file 8 bits at a time (per line). This aligns with the input data rate to the RTL module

The program generates 3 text files :

- **bitpos.txt** : Contains 8 bit binary vectors per line that is fed to the design on every clock cycle. After 128 such vectors, there is a special "XXXXXXXX" line to indicate the start of a new packet.
- **position.txt** : Contains the positions where the the 1's occur in the binary vectors. These positions are read and compared on every clock cycle when the RTL design produces valid position data.
- **hw.txt :** Contains the Hamming Weights for each of the generated packets. This is used to verify the Hamming Weights that are output from the RTL module.

The C program is found under Test_Vector_Generation/testvector_generator.c. To compile the program use the following command in a terminal :

```
gcc -Wall testvector_generator.c -o testvector
```

Run the resulting binary from the terminal :

```
./testvector
```

Part 2 : RTL Design :

The top module of the RTL design consists of two main components :

- HW_Calculator_V2 : This module takes in the input data and outputs hamming weight and a position vector. The width of the position vector is MAX_POSITIONS (which is 31

since that is the max hamming weight) * POS_WIDTH (which is 10 bits wide to represent data from 0-1023). Hence the vector is 310 bits wide. When a '1' is detected the position is calculated and written to the lowest 10 bits of the position vector. The entire vector is then left shifted by 10 bits to make palace for the next position information. This way the position vector can hold a total of 31 position values (which corresponds to the max hamming weight per packet)

- Position_Output_Module : This module outputs the hamming weight and the positions of the 1's in the data. The output positions are valid as long as the VALID_POSITION_OUT signal is asserted.

The design is synchronous to a global clock input . Each module also has a synchronous reset that is tied to an input pin of the top level module. All the constants in the design are managed from a constants package.

Part 3 : Test Bench :

The test bench Top_Level_Module_V2_TB instantiates the Top_Level_Module_V2 as its UUT (unit under test). It consists of 4 processes as part of its architecture:

- clkproc : Toggles the global clock at 100 Mhz
- feed_data : Populates the **bit_data** signal that is fed to the UUT as the data input. This process also controls the start packet signal.
- verify_data : Verifies the hamming weights and the positions output from the UUT with those from the textfiles.
- stimulus : Creates a reset pulse for 100 ns to initialize the design and then just waits.

A classio package was written to read the bitvectors from the bitpos.txt file.

Simulation and Verification was performed on Modelsim PE Student Edition 10.04a. The modelsim simulation scripts are found under the folder **Modelsim_Script**. In order to run the script, please launch Modelsim and navigate to the folder **Modelsim_Script** and execute the **Top_Level_Module_V2_TB_simulate.do** script. It compiles the design and launches the simulation.

```
do Top_Level_Module_V2_TB_simulate.do
```

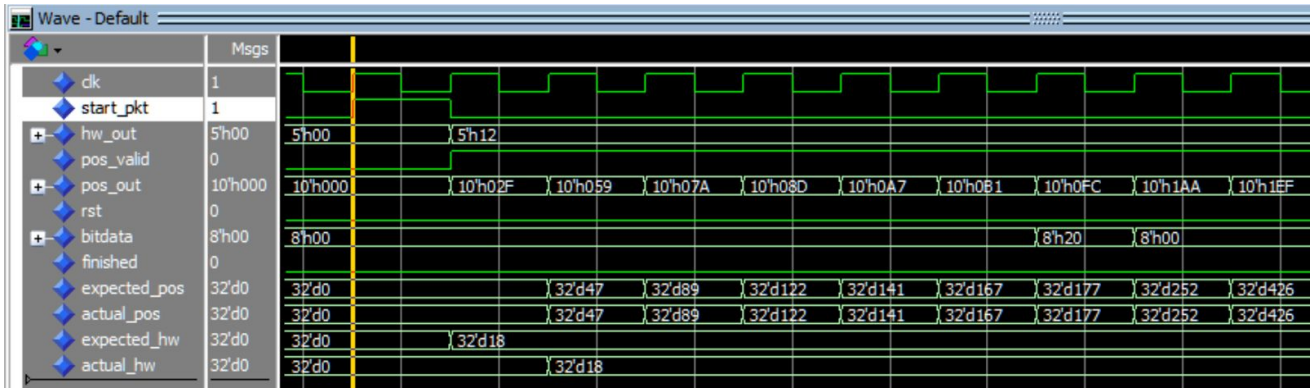
Note : Please make sure the folder structure is maintained in order for the modelsim launch scripts to work. Also please make sure the three txt files that are output from the testvector_generator.c program are in the same folder as the Modelsim launch scripts.

The following is the partial output of the simulation log :

```
# vsim
# Start time: 11:42:45 on Jan 05,2018
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading xil_defaultlib.constants(body)
# Loading xil_defaultlib.classio(body)
# Loading std.env(body)
# Loading xil_defaultlib.top_level_module_v2_tb(rtl)
# Loading xil_defaultlib.top_level_module_v2(rtl)
# Loading xil_defaultlib.hw_calculator_v2(rtl)
# Loading xil_defaultlib.position_output_module(rtl)
# .main_pane.wave.interior.cs.body.pw.wf
# .main_pane.structure.interior.cs.body.struct
# .main_pane.objects.interior.cs.body.tree
# Packet:: 1 Expeccted_HW:: 18 Actual_HW:: 18  !!Matched!!
# Packet:: 2 Expeccted_HW:: 22 Actual_HW:: 22  !!Matched!!
# Packet:: 3 Expeccted_HW:: 29 Actual_HW:: 29  !!Matched!!
# Packet:: 4 Expeccted_HW:: 21 Actual_HW:: 21  !!Matched!!
# Packet:: 5 Expeccted_HW:: 16 Actual_HW:: 16  !!Matched!!
# Packet:: 6 Expeccted_HW:: 18 Actual_HW:: 18  !!Matched!!
# Packet:: 7 Expeccted_HW:: 15 Actual_HW:: 15  !!Matched!!
# Packet:: 8 Expeccted_HW:: 5 Actual_HW:: 5  !!Matched!!
# Packet:: 10 Expeccted_HW:: 25 Actual_HW:: 25  !!Matched!!
.....
.....
.....

# Packet:: 90 Expeccted_HW:: 21 Actual_HW:: 21  !!Matched!!
# Packet:: 91 Expeccted_HW:: 6 Actual_HW:: 6  !!Matched!!
# Packet:: 92 Expeccted_HW:: 22 Actual_HW:: 22  !!Matched!!
# Packet:: 93 Expeccted_HW:: 13 Actual_HW:: 13  !!Matched!!
# Packet:: 94 Expeccted_HW:: 24 Actual_HW:: 24  !!Matched!!
# Packet:: 95 Expeccted_HW:: 25 Actual_HW:: 25  !!Matched!!
# Packet:: 96 Expeccted_HW:: 24 Actual_HW:: 24  !!Matched!!
# Packet:: 97 Expeccted_HW:: 11 Actual_HW:: 11  !!Matched!!
# Packet:: 98 Expeccted_HW:: 27 Actual_HW:: 27  !!Matched!!
# Packet:: 99 Expeccted_HW:: 6 Actual_HW:: 6  !!Matched!!
# Packet:: 100 Expeccted_HW:: 24 Actual_HW:: 24  !!Matched!!
# ** Note: Test Passed for 100 packets.
#    Time: 130305 ns  Iteration: 1  Instance: /top_level_module_v2_tb
# Break in Process verify_data at ../VHDL_Sources/Top_Level_Module_V2_TB.vhd line 120
```

The following is the snapshot showing the exected_pos, actual_pos, expected_hw and actual_hw during a simulation run



Part 4 : Synthesis and Routing : The RTL design was synthesized using Xilinx Vivado 2016.2 using a Kintex 7 FPGA (xc7k160tfbg-484-3). The following is the summary of the design (Post Routing Utilization and Timing Summary) :

Area Summary :

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	2694	0	101400	2.66
LUT as Logic	2694	0	101400	2.66
LUT as Memory	0	0	35000	0.00
Slice Registers	648	0	202800	0.32
Register as Flip Flop	648	0	202800	0.32
Register as Latch	0	0	202800	0.00
F7 Muxes	5	0	50700	<0.01
F8 Muxes	0	0	25350	0.00

2. Slice Logic Distribution

Site Type	Used	Fixed	Available	Util%
Slice	776	0	25350	3.06
SLICEL	476	0		
SLICEM	300	0		
LUT as Logic	2694	0	101400	2.66
using O5 output only	1			
using O6 output only	2484			
using O5 and O6	209			
LUT as Memory	0	0	35000	0.00
LUT as Distributed RAM	0	0		
LUT as Shift Register	0	0		
LUT Flip Flop Pairs	633	0	101400	0.62
fully used LUT-FF pairs	4			
LUT-FF pairs with one unused LUT	627			
LUT-FF pairs with one unused Flip Flop	628			
Unique Control Sets	7			

* Note: Review the Control Sets Report for more information regarding control sets.

Timing Summary :

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.455 ns	Worst Hold Slack (WHS):	0.083 ns	Worst Pulse Width Slack (WPWS):	4.600 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	964	Total Number of Endpoints:	964	Total Number of Endpoints:	649

All user specified timing constraints are met.

Throughput Calculation :

The design can process 8 bits of the input in one clock cycle. With an input clock frequency of 100 MHz, data is processed at 0.8 Gbits/sec

Part 5 : Time Spent :

January 1st : 7 am to 9 am - RTL design in Xilinx Vivado.
January 1st : 5 pm to 9 pm - RTL design in Xilinx Vivado.
January 2nd : 7 am to 9 am - Test Vector Generation C program (spent most of the time fine tuning the required output format to integrate with the vhdl test bench).
January 2nd : 1 pm to 5 pm - Test Bench development for the RTL design.
January 3rd : 7 am to 12 pm - Redesign of the RTL Module (Version 2 design)
January 4th : 7 am to 10 am - Simulation and Verification.
January 5th : 7 am to 11 am - Documentation, Formatting and Report.

Total Time Spent on this Project : **25 hours**

Part 6 : Challenges:

My initial RTL design processed the 8 bit data in 8 clock cycles (1 bit per clock cycle). Hence I had to use an input FIFO to buffer the incoming data. I also used an output FIFO to buffer the bit locations till the next start packet signal is received. I used Xilinx's IP generator to generate these FIFO modules. However I could not proceed with the simulation since the Modelsim student edition did not support mixed language simulation (the xilinx ip files are delivered as encrypted verilog files as secureip).

Hence I redesigned the entire RTL design and did not make use of the Xilinx IP generated modules (hence the version 2 design). In this new design I was able to process all of the incoming 8 bits in a single clock cycle and used a large vector to store the bit positions.