GPIO Verification Plan

This plan tries to verify the functionality of GPIO and simulate all the use cases and driving scenarios.

The following functions of the GPIO are to be verified:

1. Writing to GPIOUT and HRDATA
2. Parity generation and checking
3. Reset operation
4. Verify that the block is controlled correctly with all the input signals

Assertions in the GPIO verify the functionality 1. and 2. :

1. If gpio_dir==0, on the next cycle HRDATA[15:0]==$past(GPIOIN[15:0])
2. If conditions for write to GPIOOUT were satisfied, on the next cycle GPIOOUT[15:0]==$past(HWDATA[15:0])
3. GPIOOUT changes only when writing conditions are met on previous cycle (similar to 2.)
4. If conditions for write to gpio_dir were satisfied, on the next cycle gpio_dir[15:0]==$past(HWDATA[15:0])
5. gpio_dir changes only when writing conditions are met on previous cycle (similar to 4.)
6. If GPIOOUT changes, it's parity is correct depending on PARITYSEL in previous cycle
7. If gpio_dir==0, on next cycle checks whether PARITYERR is correct using previous cycle values of GPIOIN and PARITYSEL
8. PARITYERR changes only if on previous cycle gpio_dir==0

Driving scenarios that check the full GPIO functionality, output of GPIO is evaluated with a monitor and scoreboard:

1. Drive on a 2 cycle basis, one for address phase, one for data phase
2. Same as 1. but with random delay between every two cycles
3. Same as 1. but drive both GPIOIN and HWDATA to check only one of them was inserted
4. Try to drive new data every cycle, do an address phase only if needed
5. Perform random resets that happen asynchronously to test reset functionality

Random stimulus:

1. PARITYSEL
2. write_cycle -> 1 will drive GPIOOUT, 0 will drive GPIOIN
3. inject_parity_error -> when high, GPIOIN will have an incorrect parity bit (injected in post randomise function)
4. command_signals[2:0] -> [0]->HREADY, [1]->HSEL, [2]->HTRANS[1]
5. dir_inject -> when high, driver will inject wrong value to direction register
6. HWDATA_dir_inject -> the wrong value injected to direction register
7. HWDATA_data -> data that will be put into GPIOOUT
8. HWDATA_upper_bits -> upper 16 bits are not used in GPIO, so most of the time they are zero, but sometimes they are not zero
9. GPIOIN -> least significant 16 bits are randomised and in post randomise, the parity bit is calculated
10. inject_wrong_address[1:0] -> LSB for direction phase, MSB for data phase,
11. HADDR_inject -> value to inject if either bits of inject_wrong_address are high

Finally, functional coverage will be sampled in the interface and code coverage is automatically generated by QuestaSim. Functional coverage is checked through the following covergroups:

1. parity_injection -> checks the cross coverage of PARITYERR and PARITYSEL, but also has a coverpoint that checks whether PARITYERR has the correct value. If PARITYERR is incorrectly flagged, then an illegal bin will be hit, which would show up in the coverage report.
2. Same as 1. but checks that a parity_error was injected during a reset. This checks that the error will not be recognised if HRESETn is low
3. parity_gen_and_check -> cross coverage between PARITYSEL and GPIOIN; and between PARITYSEL and GPIOOUT
4. Separate coverpoints (no cross coverage) for GPIOIN, HWDATA, HRDATA, GPIOOUT, HADDR, gpio_dir

- Sampling of HWDATA and HADDR (has illegal bin for invalid GPIO addresses) is done only when HSEL is high because that is when the peripheral is selected.
- Sampling of gpio_dir happens on the next cycle after a value is written to it.
- Sampling of GPIOIN, GPIOOUT and HRDATA happens every cycle if HRESETn is high.
- Sampling of parity injection is done on next cycle after gpio_dir is 0 (keeps sampling if gpio_dir stays 0).
- Sampling of parity injection during reset happens every cycle.
- Sampling of parity_gen_and_check happens every cycle if HRESETn is high.