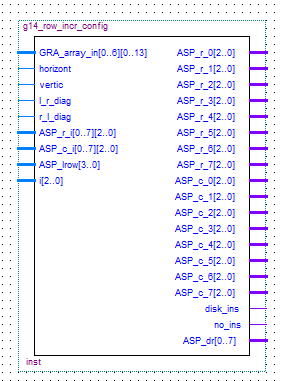
**Lab 3 Report**

***g14\_row\_increase\_config.vhd***

**Top-level I/O**

**Inputs**

GRA\_array\_in – gra\_array

Horizt, vertic, l\_r\_diag, r\_l\_diag – bit

ASP\_r\_i, ASP\_c\_i – asp\_reg

ASP\_lrow – 4-bit std\_logic\_vector

i – 3 bit std\_logic\_vector

**Outputs**

ASP\_r\_0 through ASP\_r\_7 – 2-bit std\_logic\_vector

ASP\_c\_0 through ASP\_c\_7 – 2-bit std\_logic\_vector

Disk\_ins, no\_ins – bit

ASP\_dr – 8-bit std\_logic\_vector

**NB: the gra\_array data type is a 7 by 14 array of std\_logic\_vector and the ASP\_reg data type is an 8 by 3 array of std\_logic\_vector**

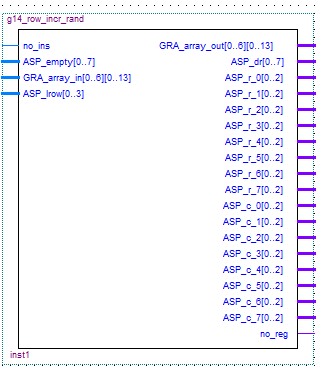
**Description of Circuit**

This circuit takes in a GRA array representation the state of the playing field prior to a piece being placed, a signal representing the directional configuration of the ASP’s longest row, the length of that row, the registers in which the row is stored, and a register index. Based on the directional configuration of the longest row, the circuit determines which end of the row the next piece should be placed by a predetermined formula. If the longest row cannot be extended in either direction, the “no\_ins” bit is activated. The output of the circuit is the modified ASP registers containing the location of the new piece placed to extend the longest row. The appropriate ASP\_dr bit is activated when there is not space to place a piece.

**Testing**

The circuit was tested by creating sample situations by implementing the GRA\_array and ASP\_r\_i and ASP\_c\_i configurations described in the Lab 3 Instruction Guide. 2 cases were considered (one for each end of the longest row) for each directional configuration. The outputs were verified by checking the output registers to ensure that the appropriate cells were added to the longest row. Finally, The empty register case was tested to prove that no\_ins would be high and disk\_ins would be low.

******

***g14\_row\_incr\_rand.vhd***

**Top-level I/O**

**Inputs**

No\_ins – bit

ASP\_empty – 8-bit std\_logic\_vector

ASP\_lrow – 4-bit std\_logic\_vector

GRA\_array\_in – gra\_array

**Outputs**

GRA\_array\_out – gra\_array

ASP\_dr – 8-bit std\_logic\_vector

ASP\_r\_0 through ASP\_r\_7 – 3-bit std\_logic\_vector

ASP\_c\_0 thorugh ASP\_c\_7 – 3-bit std\_logic\_vector

No\_reg – bit

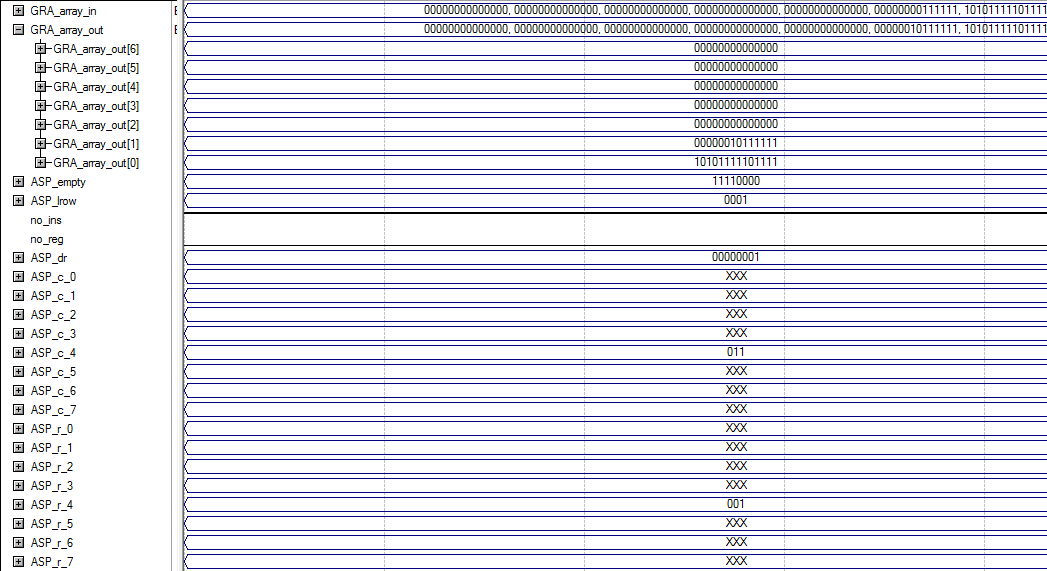
**NB: the gra\_array data type is a 7 by 14 array of std\_logic\_vector**

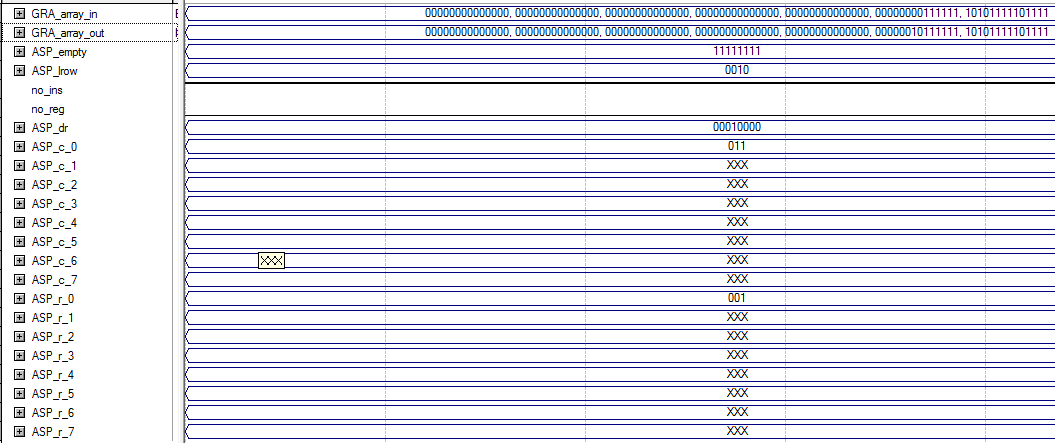
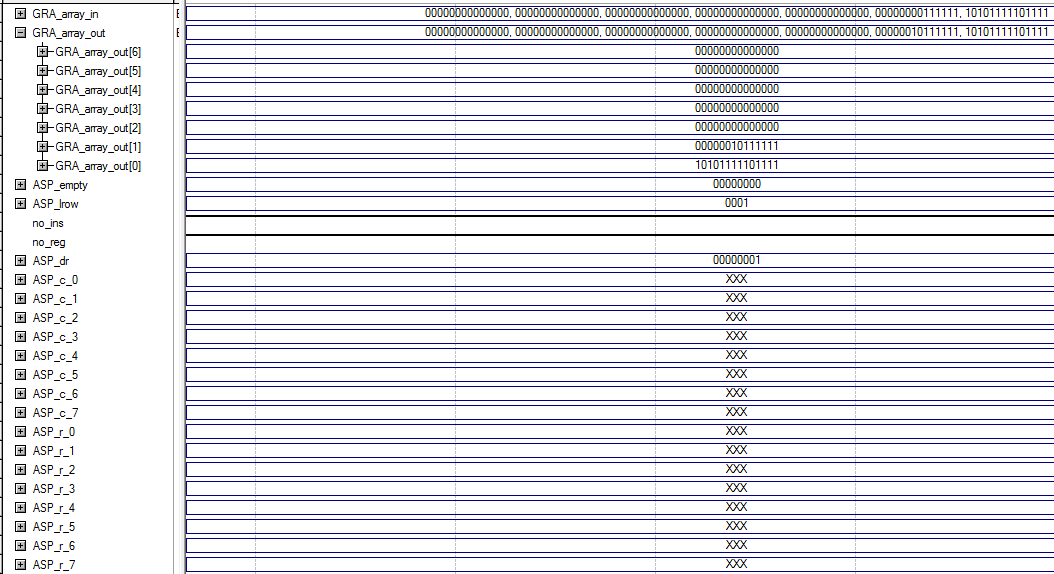
**Description of Circuit**

This circuit takes in a GRA array representing the state of the playing field prior to a piece being placed, ASP\_empty which indicates the status of empty register spaces, ASP\_lrow which indicates the length of the current longest row, and no\_ins which means that no disk has yet been inserted. The circuit is used when the longest row cannot be extended due to space limitations on the GRA array (external of this circuit). Thus, the piece is placed in the next available position by “snaking” through the GRA array. Once the next available spot is found, the piece is placed there, and the ASP registers and GRA array are updated to reflect the change.

**Testing**

The circuit was tested by creating a sample GRA array and register configurations and observing the next available cell chosen along the the postion in the register where the cell’s address was saved. Output verification requires the inspection of GRA\_array\_out to show the correct location and of the output registers to show correct address and saving those values in the proper register location.

Save to register 4

All registers are fullnext piece in row 2