The GameOfLifeArray stream module shown below constructs the Game-of-Life array.

```
class GameOfLifeArray: public streamModule
{
    cell array[nROWS][nCOLS];         // Array of cells
                                      // Cell initial states
    bool initialState[nROWS][nCOLS] = {{ 0, 0, 0, 0, 0, 0, 0, 0 },
                                        { 0, 0, 1, 0, 0, 0, 0, 0 },
                                        { 0, 0, 0, 1, 0, 0, 0, 0 },
                                        { 0, 1, 1, 1, 0, 0, 0, 0 },
                                        { 0, 0, 0, 0, 0, 0, 0, 0 },
                                        { 0, 0, 0, 0, 0, 0, 0, 0 },
                                        { 0, 0, 0, 0, 0, 0, 0, 0 },
                                        { 0, 0, 0, 0, 0, 0, 0, 0 }};
public:

    GameOfLifeArray()                 // Constructor
    {                                 // Stream-domain code
        int r, c;

        for (r = 0; r < nROWS; ++r)   // Initialize cell states
        {
            for (c = 0; c < nCOLS; ++c)
            {
                array[r][c].setState( initialState[r][c] );
            }
        }
                                      // Interconnect array of cells
        interconnect2D( nROWS, nCOLS, tsNS_WRAP, tsWE_WRAP );
        end();                        // Housekeeping
    }
};
```

Some observations about GameOfLifeArray:

a. The array data-member

   cell array[nRows][nCols];

   is the Game-of-Life grid of cells.

b. The array data-member

   bool initialState[nRows][nCols]

   provides initial states for cells in the Game-of-Life grid. An initial state of 0 represents a *dead* cell, while an initial state of 1 represents a *live* cell.

c. In the GameOfLifeArray constructor, the statement

   interconnect2D(nRows, nCols, tsNS_WRAP, tsWE_WRAP);

   calls the streamModule member function

```
int interconnect2D( int              nRows,
                    int              nCols,
                    TsNorthSouthWrap NSwrap,
                    TsWestEastWrap   WEwrap )
```

where:

nRows is the number of rows in the (unique) two-dimensional-module-array data member of the streamModule.

nCols is the number of columns in the (unique) two-dimensional-module-array data member of the streamModule.

TsNorthSouthWrap is the enumeration type

```
typedef enum { tsNS_NOWRAP = 0, tsNS_WRAP = 1 } TsNorthSouthWrap;
```

The argument determines whether outward-facing inputs and outputs on the north and south sides of the array *wrap around* (NSwrap == tsNS_WRAP) or not (NSwrap == tsNS_NOWRAP).

TsNorthSouthWrap is the enumeration type

```
typedef enum { tsWE_NOWRAP = 0, tsWE_WRAP = 1 } TsWestEastWrap;
```

The argument determines whether outward-facing inputs and outputs on the west and east sides of the array *wrap around* (WEwrap == tsWE_WRAP) or not (WEwrap == tsWE_NOWRAP).

The call to interconnect2D causes each cell in the *unconnected* GameOfLifeArray data member array[nROWS][nCOLS], illustrated in Figure 1 (below), to be connected to its 8 nearest neighbors as illustrated in Figure 2 (below)

interconnect2D accomplishes this feat by performing the following steps for each input-stream inStrm of each module M in array[nROWS][nCOLS]:

(1) If (a) inStrm is not connected to a stream (i.e., inStrm is a dangling input)

(b) Module M has an inStrm.direction neighbor N

(c) Neighbor N has output-stream outStrm such that:

(i) The data-values sizes for inStrm and outStrm are the same

(ii) outStrm is facing Module M or outStrm.direction == tsALLDIRECTIONS

(2) Then create a TruStream connecting outStrm to inStrm

d. In the GameOfLifeArray constructor, the statement

```
end();
```

signals to the TruStream compiler that the stream-domain code in the constructor has ended. It is REQUIRED in ALL stream-module constructors, including default constructors in which there are no other statements.

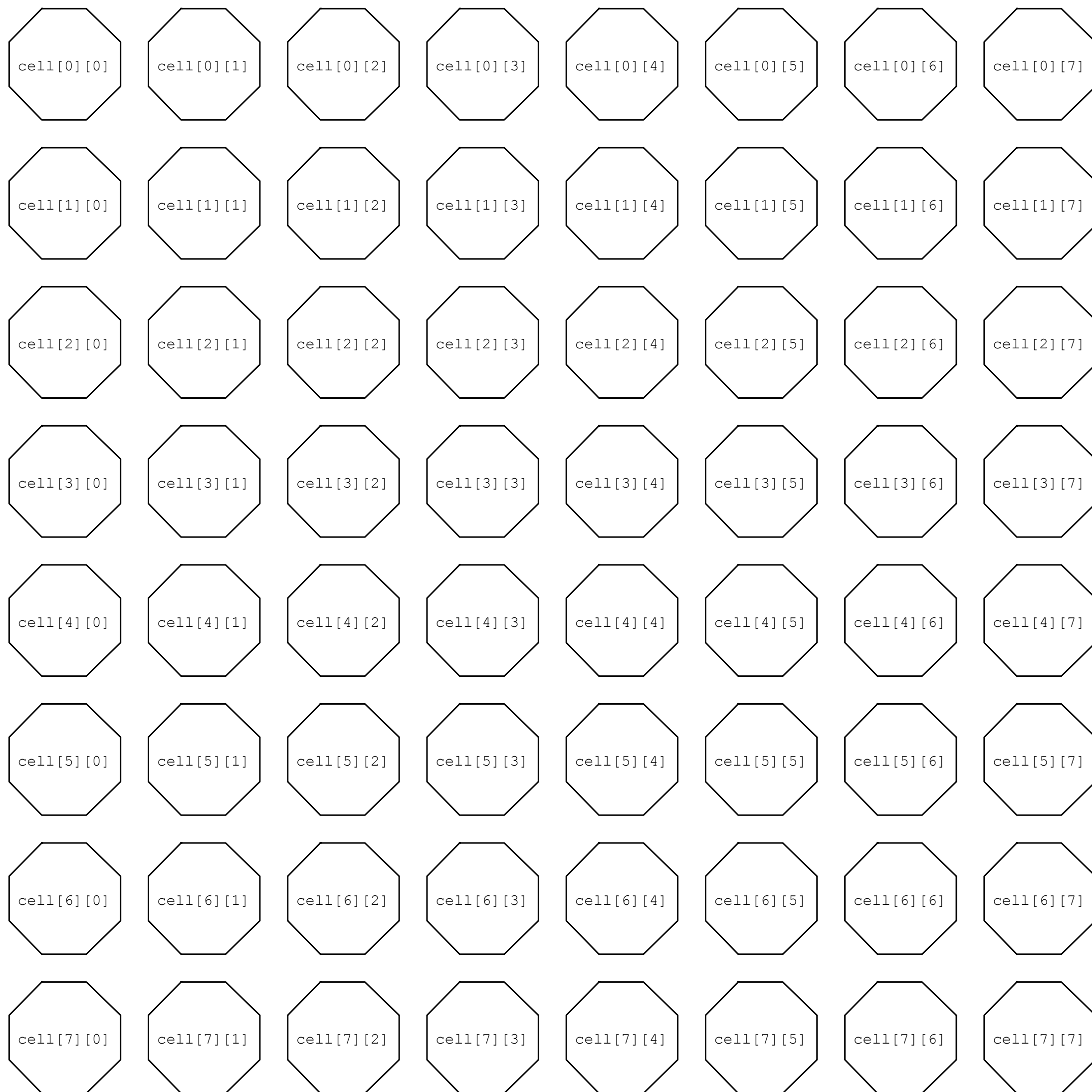

Figure 1. Pre-Interconnect2D Systolic Array of Cells

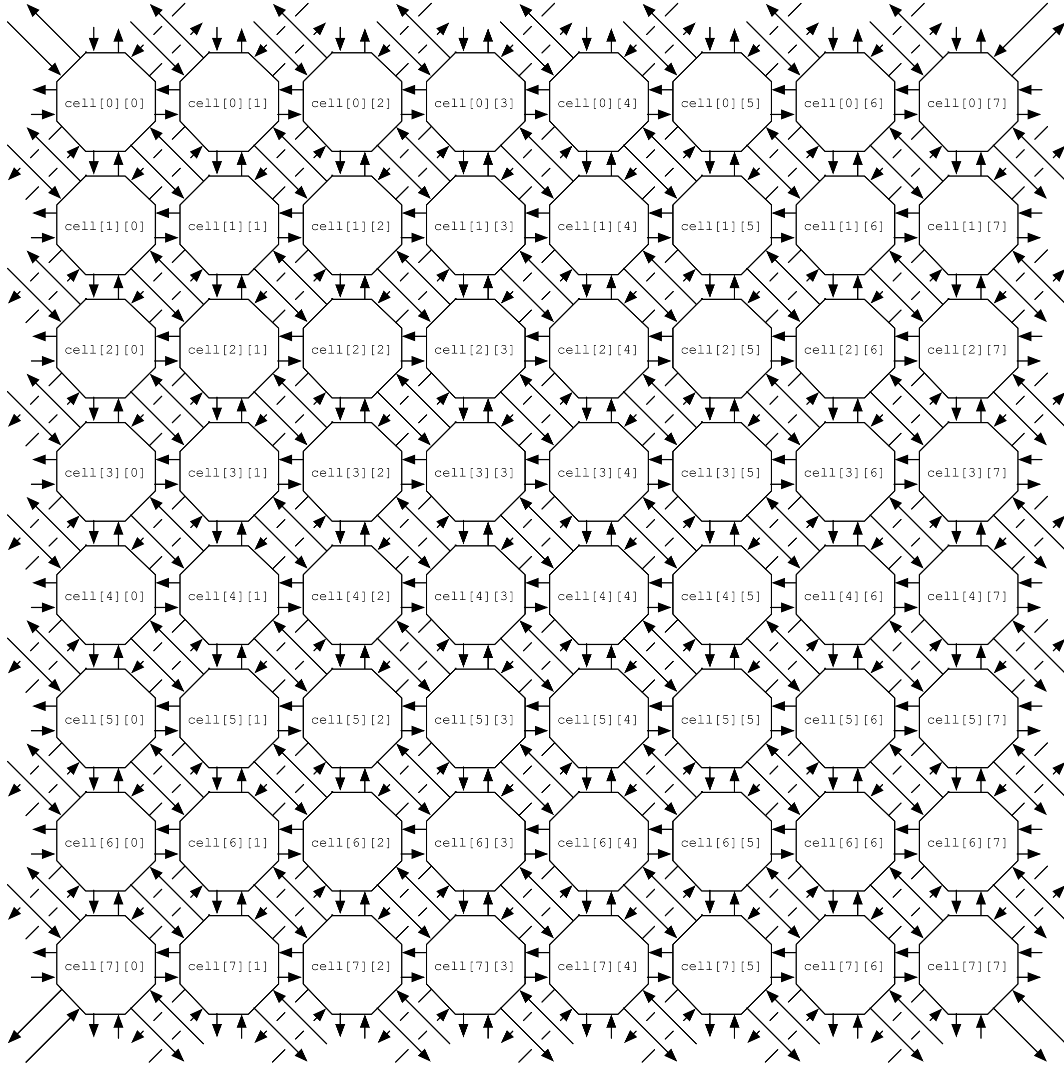

Figure 2. Post-Interconnect2D Systolic Array of Cells