trenz electronic

Dünner Kirchweg 77
32257 Bünde
Germany
www.trenz-electronic.de

# Spartan-II Development System

2002-May-4

Application Note: Game of Life

## Introduction

The Spartan-II Development System is designed to provide a simple yet powerful platform for FPGA development, which can be easily expanded to reflect your application's requirements.

The following application note was developed to give the engineer a quick hands-on experience, and to demonstrate the board's features and their application.

## General Overview

The *TE-BL Expansion Board* provides a set of standard functionality which is commonly used by most applications:

- 7-segment displays
- LEDs
- Push buttons
- VGA output
- USB type "B" receptacle
- 48MHz oscillator

To speed-up application development, a standard VHDL module (*entity tebl*) was created, encapsulating the following functions:

- encoding of 7-segment displays
- multiplexing of LEDs
- debouncing of push buttons.
- emulation of switches
- generation of VGA timing
- encoding of USB signals

Based on the above mentioned VHDL module, this application note describes an FPGA implementation of Conway's Game of Life, demonstrating the following tasks:

- using the 7-segment displays
- using the LEDs
- using the push buttons
- emulating switches
- using the VGA output

This application note concentrates on the creation of VGA images. For a more detailed discussion of the application of 7-segment displays, LEDs and buttons, refer to the *Buttons & Lights* application note.

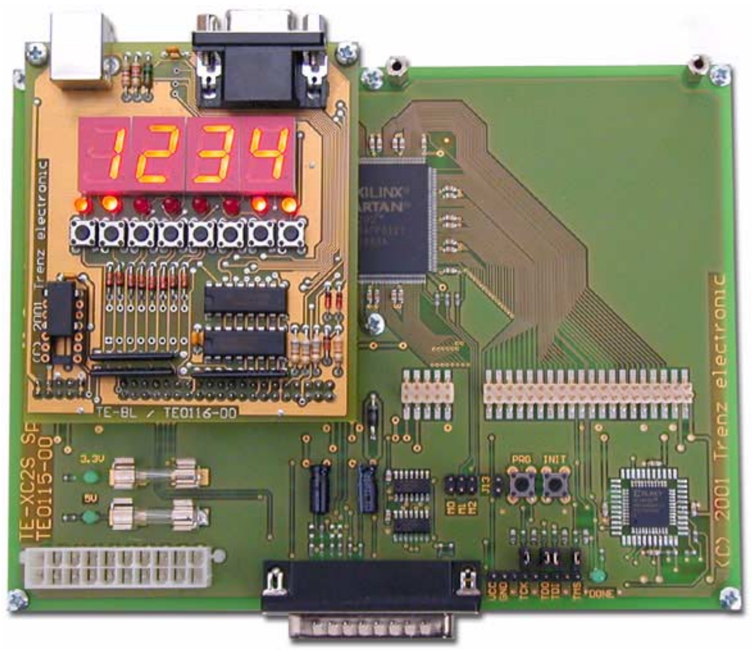

**Figure 1: TE-XC2S Base Board with TE-BL Expansion.**

## Architectural Description

Conway's Game of Life is a mathematical game, simulating the growth of a population of cells in a toy universe. It is played on a two-dimensional grid. For every generation, the state of each square is computed according to the following rules:

- on squares with two or three alive neighbors, existing cells survive
- on squares with three alive neighbors, a new cell is born
- on the remaining squares, the cells die

The remarkable thing about Life is the unexpected chaos that results from its simple rules.

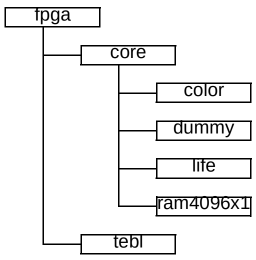

**Figure 2: Design hierarchy**

The design is partitioned into several entities, each of them serving a well-defined purpose. *Figure 2* visualizes the design hierarchy.

### Entity core

The entity *core* implements the Game of Life functionality by combining a set of building blocks. *Figure 3* illustrates the structure.

The instance *Ulife* implements the control logic required to compute the next generation of cells and to display the game on the VGA output.

The instances *Uram_a* and *Uram_b* are two RAM banks with 4096x1 bits each. The control logic toggles these two banks, so that one bank serves as a data source holding the current generation of cells, while the other bank is used to store the next generation of cells being computed.

The instance *Ucolor* maps the single bit output from the control logic to an RGB plus intensity tuple for VGA display. The color mapping may be selected from four different palettes.

Finally, the instance *Utebl* contains the readily provided *TE-BL Expansion Board* interface circuitry.

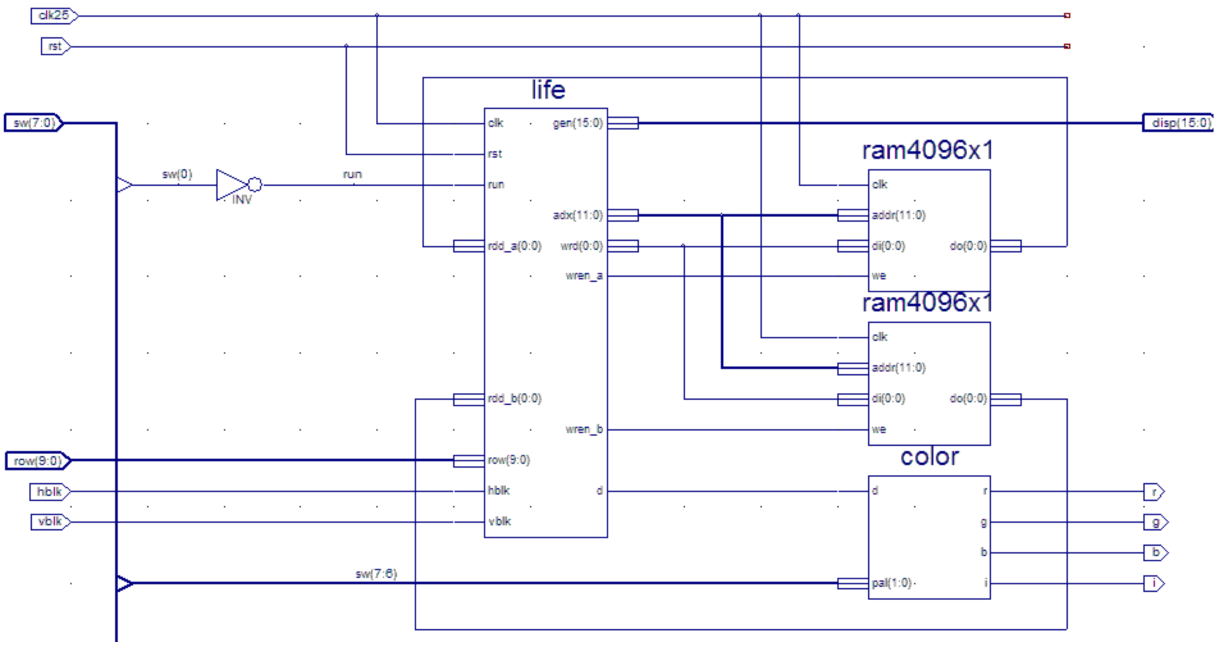

**Figure 3: Entity core**

## Entity life

The instance *Ulife* implements the control logic required to compute the next generation of cells and to display the game on the VGA output.

The entity *life* is clocked with the VGA dot clock, as the entity works in close interaction with entity *tebl* creating the VGA timing.

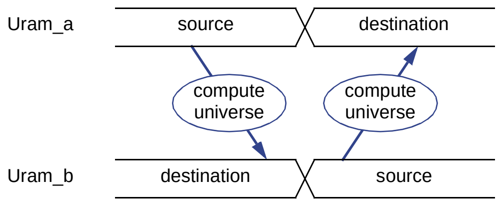

**Figure 4: RAM bank toggle**

The universes dimensions are 64x64 cells, which results in 10x8 pixels per cell at VGA resolution of 640x480 pixels. The 64x64 cell universe is stored in two RAM banks of 4096x1 each. Two banks are required, as the Game of Life algorithm cannot be computed in-place. Therefore one bank serves as the data source to VGA and cell computation, while the other bank is used as the destination. After completing computation of the next generation, the two banks are toggled. Figure 4 illustrates this.

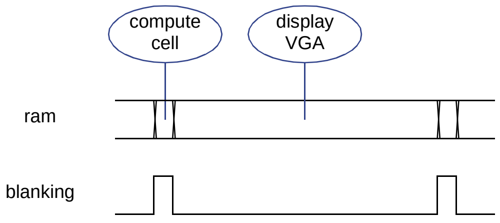

**Figure 5: Compute during blanking**

As VGA display and computation of the next cell generation are assumed to be performed simultaneously, a mechanism is required, to share the memory between VGA output and cell computation. To avoid using dual-ported RAM, the computation of cells is performed during the blanking intervals, i.e. when the screen is black. Refer to Figure 5 for further details. To keep things simple, one cell is computed during each horizontal or vertical blanking, resulting in a computation time of approximately 140ms for the complete 64x64 universe.

## Entity ram4096x1

The instances *Uram_a* and *Uram_b* of entity *ram4096x1* are used to store a complete universe of cells at a given time. To achieve the best possible implementation density, Block-RAMs are used. As the *WebPACK ISE* software package does not include the CoreGenerator, manually instantiated *RAMB4_S1* instances are used. After FPGA configuration, the RAMs contain the initial cell population. To do so, the RAMs are initialized using the VHDL attribute *INIT_xx*. The following code snippet details this:

```
architecture Xilinx of ram4096x1 is
  attribute INIT_00: string;
  ...
  attribute INIT_0F: string;
  attribute INIT_00 of Uram: label is
                 "0000000000000000
                  0000000000000000
                  0000000000000000
                  0000000000000000";
  ...
  attribute INIT_0F of Uram: label is
                 "0000000000000000
                  0000000000000000
                  0000000000000000
                  0000000000000000";
begin
  Uram: ramb4_s1 port map (
    addr=> addr,
    clk => clk,
    rst => zero,
    di  => di,
    en  => one,
    we  => we,
    do  => do);
end Xilinx;
```

The initial value is given in hex, please note that there is no preceding ‘x’ used, as attribute *INIT_xx* is of type *string*.

## Entity fpga

The entity *fpga* is the top level of the design. It performs the following functions:

- create a reset signal
- lock all I/Os to their pad locations
- specify timing constraints

While Xilinx recommends to assign design constraints using the *Constraints Editor*, this application note uses VHDL attributes to pass the constraints to the implementation tools. This is especially useful for the novice user, as the design contains less files and is more consistent.

The mechanism used here was proven with Xilinx' *WebPACK ISE 3.3WP8.x*. In case your tool chain does not support this method, the constraints me be additionally entered into a *.ucf* file.

A detailed description of these mechanism may be found in the *Buttons & Lights* application note.

## Entity tebl

A VGA image displays a rectangular area which is composed of square pixels. The dimensions of the image are 640x480 pixels. The image is surrounded by a dark blanking area. *Figure 6* illustrates this.

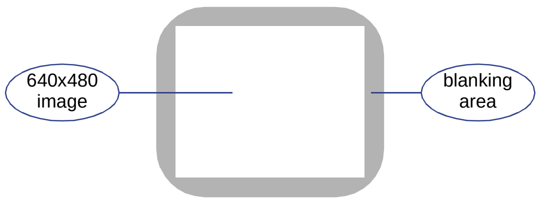

**Figure 6: VGA image**

The image is generated from a sequence of frames which are repeated quickly enough, so that the human eye does not notice the flicker. The frames are identified by vertical sync pulses. Each frame displays 480 scan lines. Above and below these lines is a blanking area, which stays dark. See *Figure 7* for further details.

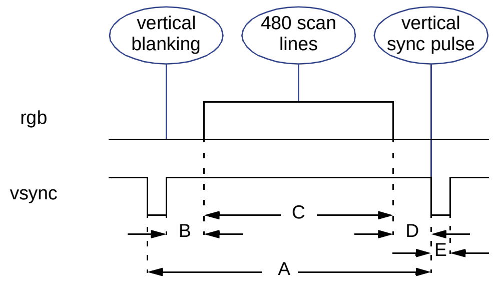

**Figure 7: Vertical refresh cycle**

The 480 scan lines are identified by horizontal sync pulses. Each line displays 640 pixels. To the left and to the right of these pixels is a blanking area, which stays dark. See *Figure 8* for further details.

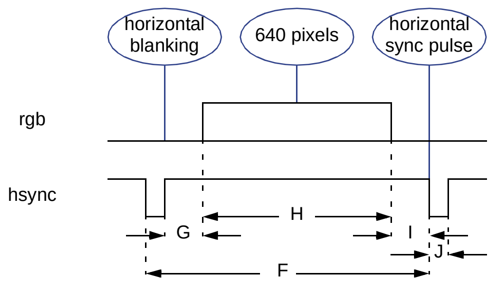

**Figure 8: Horizontal refresh cycle**

The timing of these parameters is standardized, so that signal sources and monitors may be combined freely. *Table 1* summarizes the timing parameters.

| Parameter | Description | Time |
|---|---|---|
| A | vertical refresh | 16.6ms (60Hz) |
| B | top blanking | 1.02ms |
| C | vertical image | 15.25ms |
| D | bottom blanking | 0.35ms |
| E | vsync width | 64us |
| F | horizontal refresh | 31.77us (31.5kHz) |
| G | left blanking | 1.89us |
| H | horizontal image | 25.17us |
| I | right blanking | 0.94us |
| J | hsync width | 3.77us |

**Table 1: VGA timing parameter**

All these timing parameters are derived from the pixel clock, which is typically 25.175MHz. Due to the requirement of using a 48MHz oscillator for USB, the *tebl* entity was adapted to use a 24MHz oscillator instead. This results in slightly shortened blanking intervals, while the other parameters remain unaltered.

The entity *tebl* is provided as VHDL source code and extensively commented. It is left up to the interested engineer, to review this source for more detailed information.

## Project files

The project files for this application note are provided in *WebPACK ISE* format with all synthesis options set up to achieve a push button flow. Furthermore, the resulting *.mcs* file to program the Flash PROM and a *.bit* file to program the FPGA via JTAG are provided.

To answer questions regarding synthesis, implementation and download of projects, our *Tutorial on WebPACK ISE* is highly recommended.

## References

- *Spartan-II Development System*
  *Product Specification*
  Trenz Electronic
  September 12, 2001
- *Spartan-II Development System*
  *Tutorial on WebPACK ISE*
  Trenz Electronic
  September 2001
- *Spartan-II Development System*
  *Application Note: Buttons & Lights*
  Trenz Electronic
  September 20, 2001

## Revisions History

| Version | Date | Who | Description |
|---|---|---|---|
| 1.0 | 2001sep17 | FB | Created |
| 1.1 | 2002may04 | FB | ISE 4.2 |

**Table 2: Revisions History**