|  |
| --- |
| Cornell University, School of Electrical and Computer ENgineering |
| One-Dimensional Cellular Automaton |
| ECE5760 |
|  |
| **Jeremy Blum, Sima Mitra, Jason Wright** |
| **Thursday Lab Section** |

|  |
| --- |
| A binary, nearest-neighbor, one dimensional cellular automaton was calculated by a state machine running on an FPGA to be displayed on a connected VGA monitor at a resolution of 640x480 pixels. |

# Introduction

The objective of this lab was to develop a 1-D cellular automaton that could be displayed on a VGA monitor at a resolution of 640x480. An Altera DE-2 board with a Cyclone II FPGA was used to build a hardware state machine paired with a VGA driver and appropriate memory management to permit on-the-fly calculation of the cellular automaton states. The cellular automaton rule is entered using the slider switches; the seed row can be set to a single pixel in the middle or a random row of pixel values. An “advance” button shows the next 480 lines of evolution, and a reset button starts a new evolution with the settings specified by the switches. The 7 segment displays are used to show the current rule setting.

# Design and Testing Methods

Below, we describe each of the key components to the larger system in detail, and how we tested each individual component before piecing it together into the complete cellular automaton system.

## Inputs and Outputs

We used a total of nine switches. Switches zero through seven are used to input an 8-bit binary value to serve as the rule for the cellular automaton. Switch 17 is used to toggle between using a single pixel starting point or a random distribution of pixels in the top row. To confirm that the switches were being read properly by the program, the seven segment LED displays and the RED LEDs were configured to show the state of the switches and the state of the program. The seven segment displays show the binary value that has been set by the rule switches, and some of the red LEDs were configured to show which state the program is current executing. Figure 1 below shows a rule selected on the switches and confirmed on the seven segment displays:

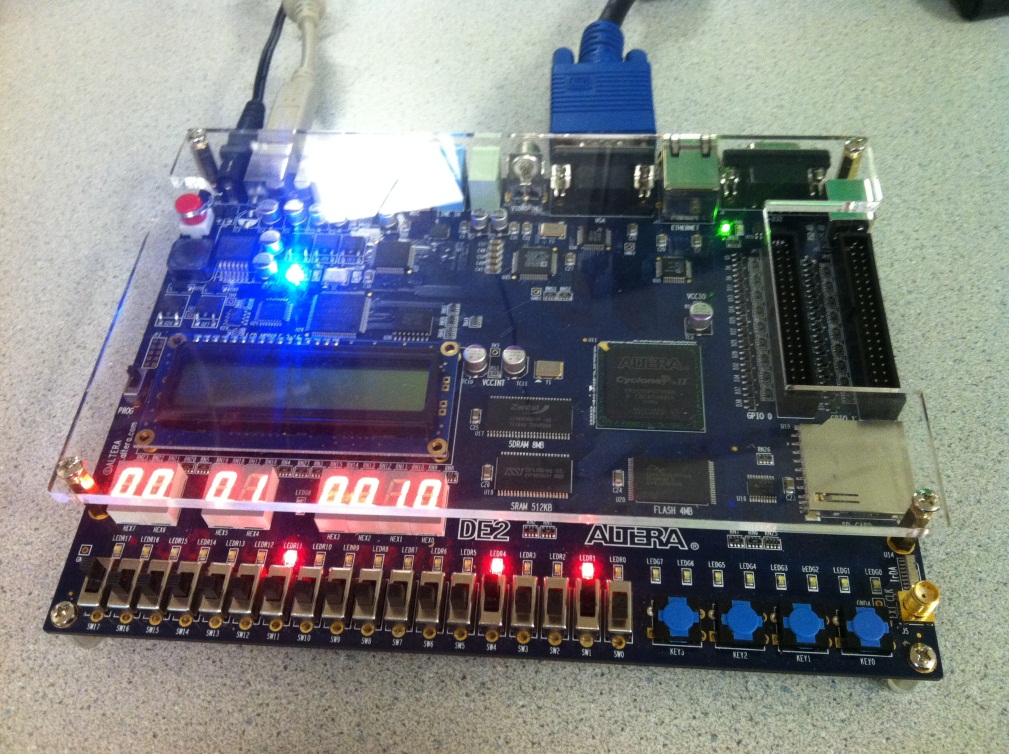


Figure 1: Switch States shown on Seven Segment Displays

Key 0 was used to reset the state machine and Key 3 was used to advance the VGA display to the next line. Since the buttons on the DE-2 board sometimes do not work well, they were configured to illuminate the green LEDs above them when pressed. This provided useful confirmation that the button presses were successful.

## High-Level Structure

Figure 2 below shows the basic high-level system architecture:

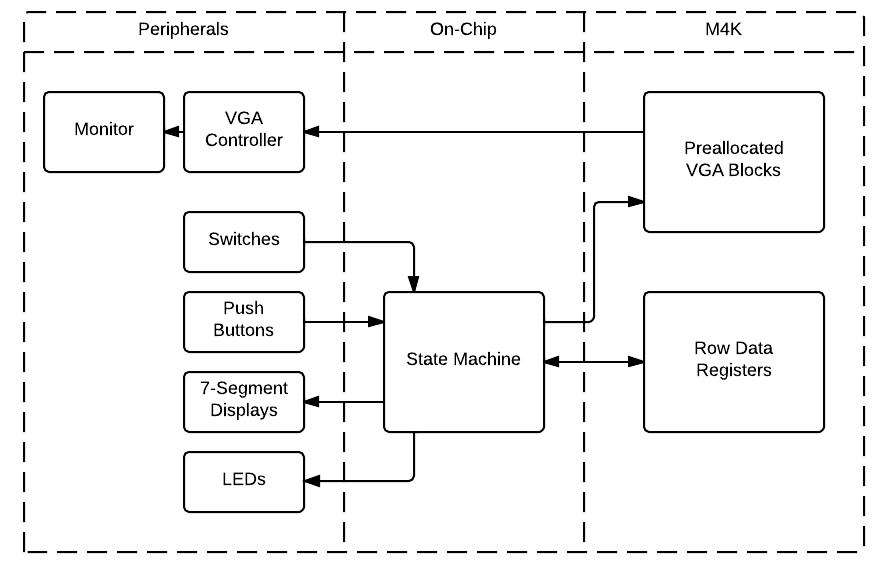


Figure 2: High-Level System Organization Diagram

The state machine is written in Verilog and is built into the FPGA hardware. Our design calculates the entire next row of pixels at once based on the previous row and the cellular automaton rule using the M4K memory. This parallel computation is implemented by using 640 single automaton hardware units to compute the value of each pixel using data from the previous row. The VGA image buffer is stored in M4K blocks, which are just large enough to hold a 640x480 image in addition to the cellular states. The state machine is wired to the pushbutton and switch inputs for controlling the rule selection, random/single pixel state, reset, and “advance” functions.

## State Machine

Figure 3 shows the state machine that was implemented in hardware using Verilog. The state machine is responsible for responding to keypresses, reading the automaton rule, and generating the contents to be sent to the VGA display adapter. The three print cases are necessary to insert a timing delay for the VGA writes, and the “chill” states are where the program sits when it is waiting for user input to either reset the program or advance it to the next screen. Any time key 0 is pressed, the state machine will be reset to the initial state.

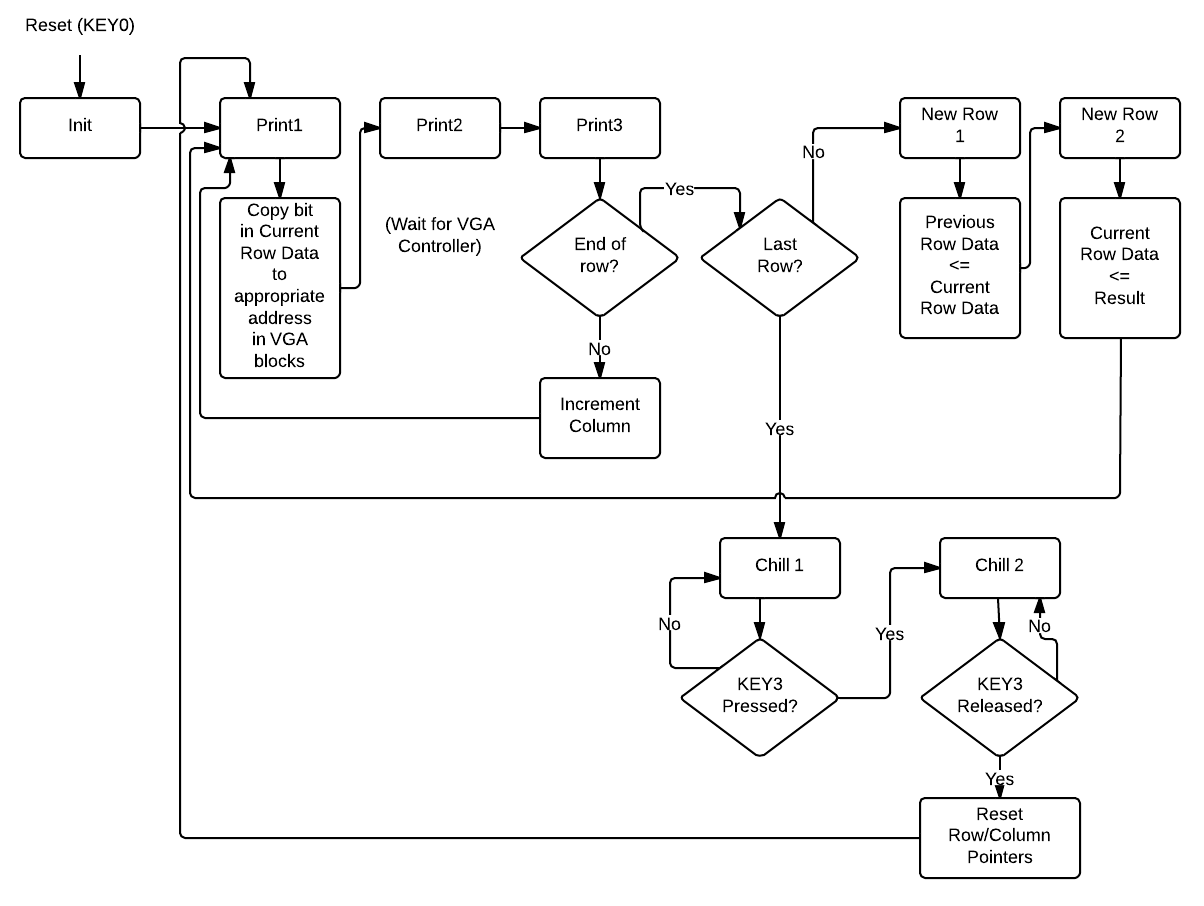


Figure 3: State Machine Diagram

## Automaton Hardware

Figure 4Figure 4 shows the process for computing an entire row of automaton values in parallel.

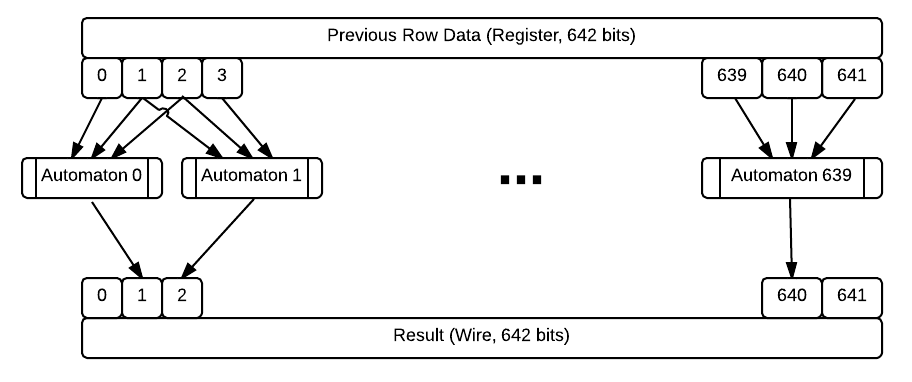


Figure 4: Automaton Row-by-Row Calculation

Each row of pixel values is stored in a large 642-bit array. The additional two bits are used as zero padding for the pixels that lie at the edges of the screen. For initial testing, the switches 17-15 were used to enter three values for the cellular automaton rule to operate on and the output of a single computation was displayed on the VGA in one pixel. By comparing the correct outcome of the cellular automaton with the output displayed allowed us to verify the computation.

## VGA

The VGA controller onboard the DE2 board is connected to a pre-allocated 640x480 block within M4K, accessed via addr\_reg and data\_reg pointers. Computed data is pushed into the VGA registers from registers on the Cyclone II chip, and the VGA controller accesses the M4K memory at 60 Hz to refresh the screen.

## Random Seed generation

20 32-bit linear feedback shift registers were used to generate a total of 640 pixels for the randomized first row when the appropriate selector switch was activated. A conglomeration of 32-bit registers were used instead of one large 640 bit register because there is no known pseudo-random-generating 640-bit-wide linear feedback shift register. Figure 5 below shows how one of these 32-bit linear feedback shift registers works:

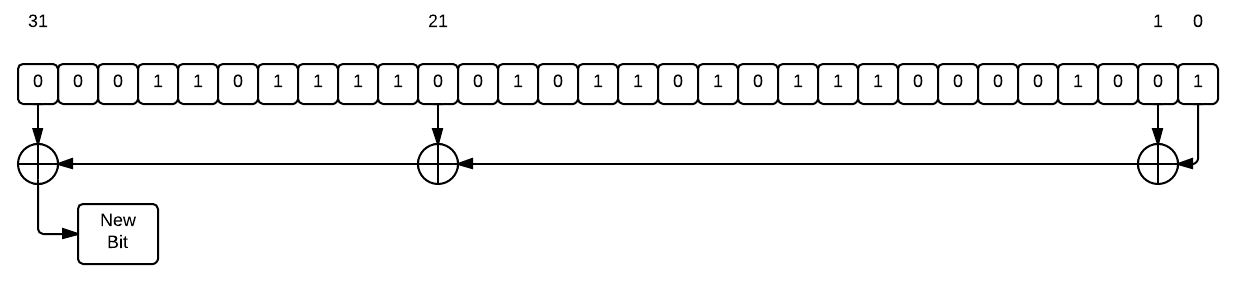


Figure 5: 32-Bit LFSR[[1]](#footnote-1)

In Figure 6 below, you can see how the values from 20 LFSR’s were chained together to generate 640 random bits of data for the first seed row.

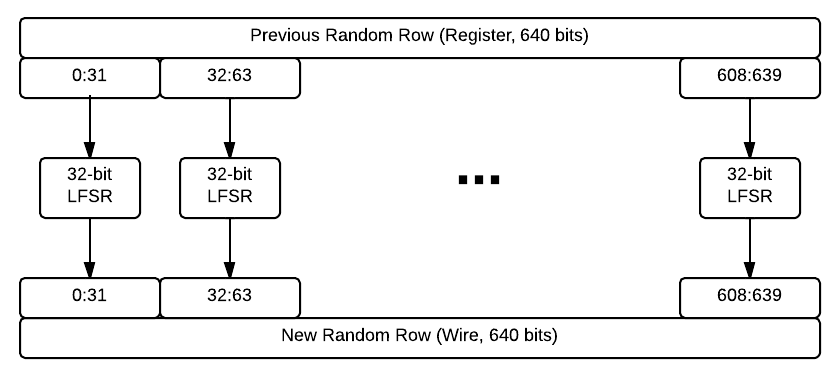
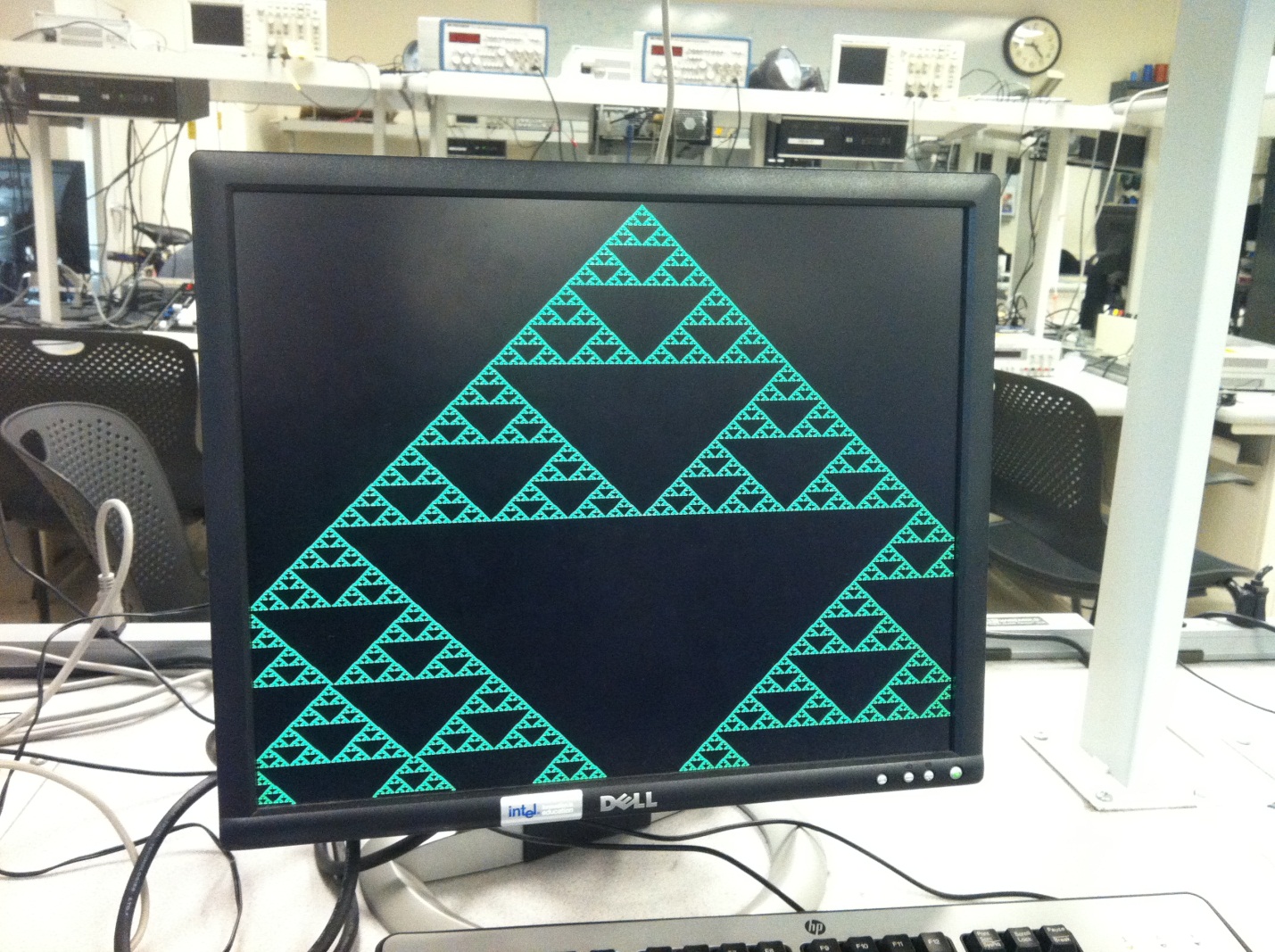


Figure 6: Chaining 20 LFSRs together

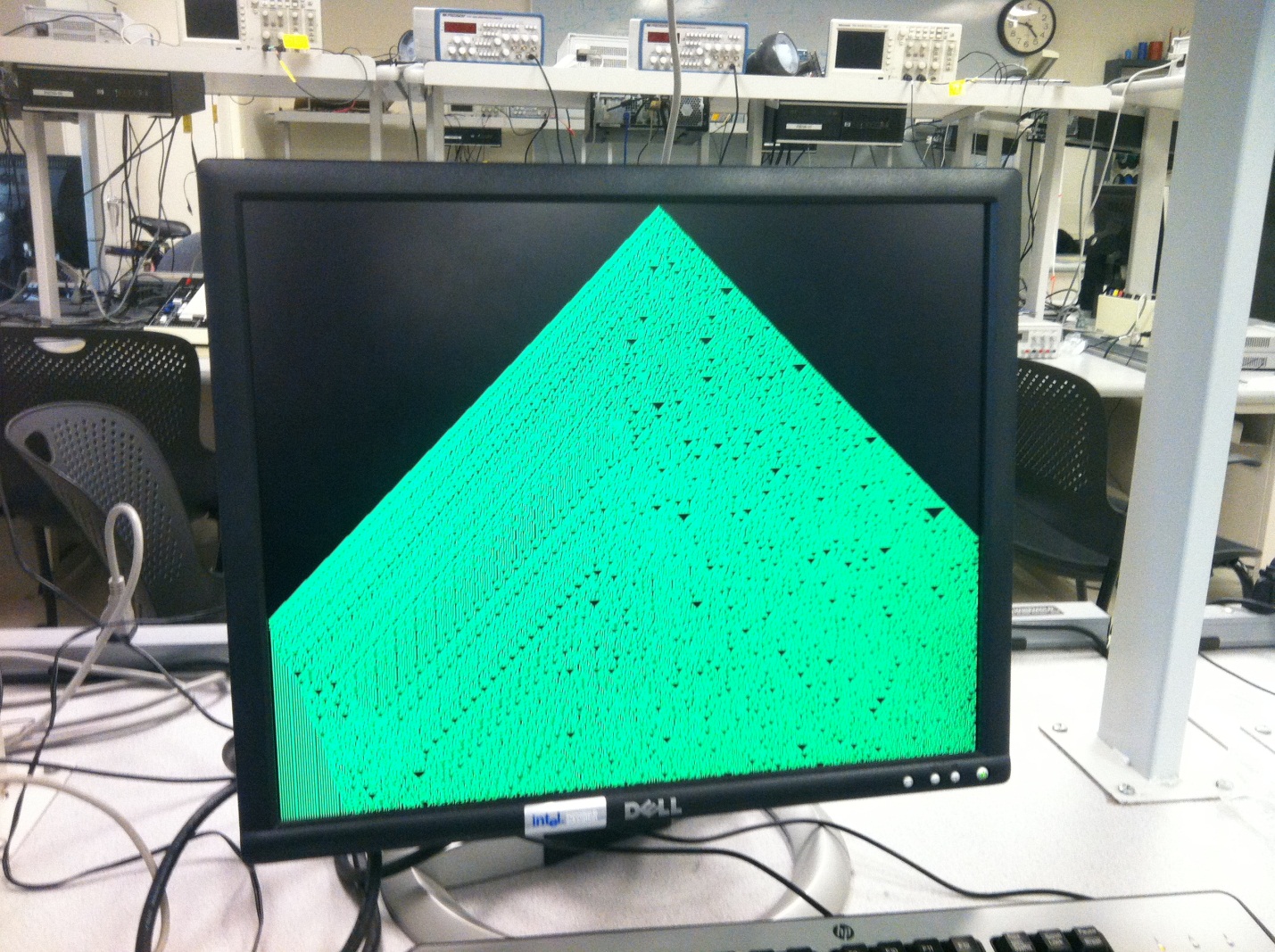
# Documentation

## Example Images

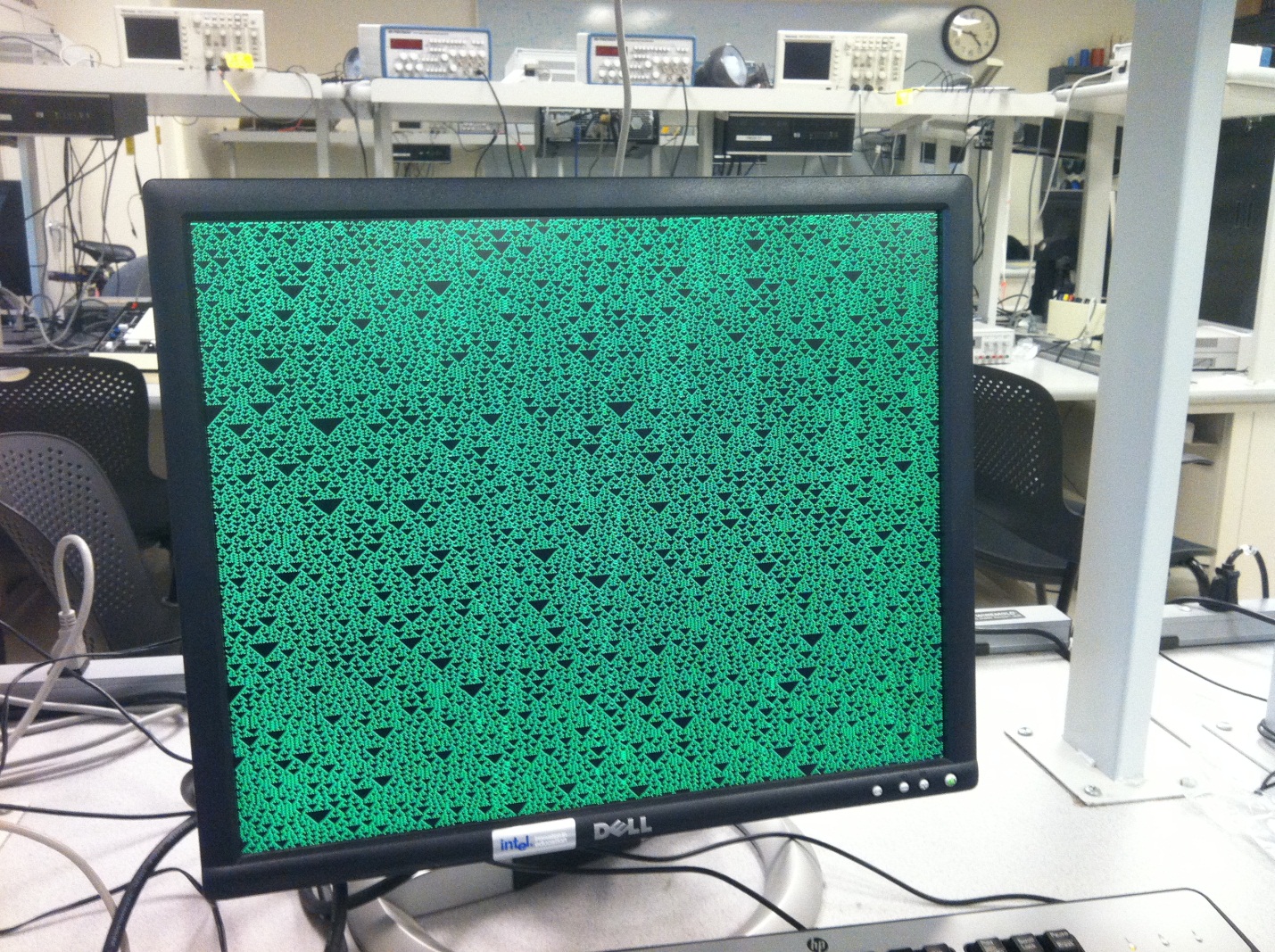
Below we show an example of rule 26 with a single seed pixel, which makes a repeating triangle tessellation.



The next figure shows rule 30 with a single seed pixel.



The next figure shows the same rule applied to a randomized seed row.



# Comparison of 1-D Cellular Automaton with Natural Phenomenon

## Seashells

The pigmentation of certain seashells is governed by cellular automaton-like rules: The shell of the *Conus Textile* is not unlike rule 30:



Figure 7: *Conus Textile* Seashell (Source: Wikipedia)

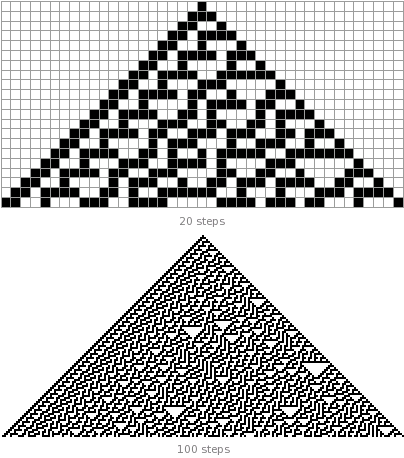


Figure 8: Rule 30 (Source: Wolfram Alpha)

## Caterpillars

For caterpillars, rule 105 is similar to the pattern on the head of the caterpillar, but the pattern on the body does not appear to be governed by a 1-dimentional cellular automaton rule.



Figure 9: A caterpillar with a cellular automaton like pattern. (Source: http://www.kathryncramer.com/kathryn\_cramer/2005/09/cellular\_automa.html)

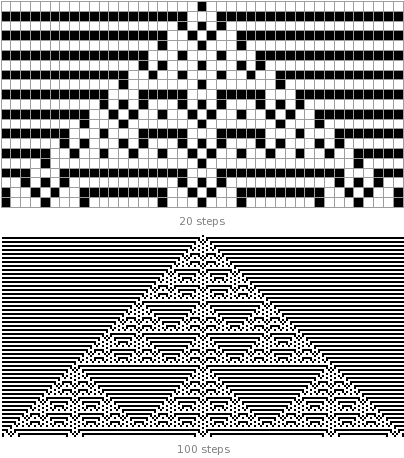


Figure 10: Rule 105 (Source: Wolfram Alpha)

## Other 1-D Growth Systems

Rule 184 is used to model surface deposition in certain crystalline structures (D. de Cogan and L.R. Martin, "Simple models for crystallisation processes," Intl. Conference on Solid State Crystals, 2000.). Cellular automatons are diffusion-limited aggregation are both examples of one-dimensional growth systems.

# Program Listing

// --------------------------------------------------------------------

//

// Major Functions: Cellular Automaton

// state is in m4k blocks

// --------------------------------------------------------------------

**module** DE2\_TOP **(**

// Clock Input

**input** CLOCK\_27**,** // 27 MHz

**input** CLOCK\_50**,** // 50 MHz

**input** EXT\_CLOCK**,** // External Clock

// Push Button

**input** **[**3**:**0**]** KEY**,** // Pushbutton[3:0]

// DPDT Switch

**input** **[**17**:**0**]** SW**,** // Toggle Switch[17:0]

// 7-SEG Display

**output** **[**6**:**0**]** HEX0**,** // Seven Segment Digit 0

**output** **[**6**:**0**]** HEX1**,** // Seven Segment Digit 1

**output** **[**6**:**0**]** HEX2**,** // Seven Segment Digit 2

**output** **[**6**:**0**]** HEX3**,** // Seven Segment Digit 3

**output** **[**6**:**0**]** HEX4**,** // Seven Segment Digit 4

**output** **[**6**:**0**]** HEX5**,** // Seven Segment Digit 5

**output** **[**6**:**0**]** HEX6**,** // Seven Segment Digit 6

**output** **[**6**:**0**]** HEX7**,** // Seven Segment Digit 7

// LED

**output** **[**8**:**0**]** LEDG**,** // LED Green[8:0]

**output** **[**17**:**0**]** LEDR**,** // LED Red[17:0]

// UART

**output** UART\_TXD**,** // UART Transmitter

**input** UART\_RXD**,** // UART Receiver

// IRDA

**output** IRDA\_TXD**,** // IRDA Transmitter

**input** IRDA\_RXD**,** // IRDA Receiver

// SDRAM Interface

**inout** **[**15**:**0**]** DRAM\_DQ**,** // SDRAM Data bus 16 Bits

**output** **[**11**:**0**]** DRAM\_ADDR**,** // SDRAM Address bus 12 Bits

**output** DRAM\_LDQM**,** // SDRAM Low-byte Data Mask

**output** DRAM\_UDQM**,** // SDRAM High-byte Data Mask

**output** DRAM\_WE\_N**,** // SDRAM Write Enable

**output** DRAM\_CAS\_N**,** // SDRAM Column Address Strobe

**output** DRAM\_RAS\_N**,** // SDRAM Row Address Strobe

**output** DRAM\_CS\_N**,** // SDRAM Chip Select

**output** DRAM\_BA\_0**,** // SDRAM Bank Address 0

**output** DRAM\_BA\_1**,** // SDRAM Bank Address 0

**output** DRAM\_CLK**,** // SDRAM Clock

**output** DRAM\_CKE**,** // SDRAM Clock Enable

// Flash Interface

**inout** **[**7**:**0**]** FL\_DQ**,** // FLASH Data bus 8 Bits

**output** **[**21**:**0**]** FL\_ADDR**,** // FLASH Address bus 22 Bits

**output** FL\_WE\_N**,** // FLASH Write Enable

**output** FL\_RST\_N**,** // FLASH Reset

**output** FL\_OE\_N**,** // FLASH Output Enable

**output** FL\_CE\_N**,** // FLASH Chip Enable

// SRAM Interface

**inout** **[**15**:**0**]** SRAM\_DQ**,** // SRAM Data bus 16 Bits

**output** **[**17**:**0**]** SRAM\_ADDR**,** // SRAM Address bus 18 Bits

**output** SRAM\_UB\_N**,** // SRAM High-byte Data Mask

**output** SRAM\_LB\_N**,** // SRAM Low-byte Data Mask

**output** SRAM\_WE\_N**,** // SRAM Write Enable

**output** SRAM\_CE\_N**,** // SRAM Chip Enable

**output** SRAM\_OE\_N**,** // SRAM Output Enable

// ISP1362 Interface

**inout** **[**15**:**0**]** OTG\_DATA**,** // ISP1362 Data bus 16 Bits

**output** **[**1**:**0**]** OTG\_ADDR**,** // ISP1362 Address 2 Bits

**output** OTG\_CS\_N**,** // ISP1362 Chip Select

**output** OTG\_RD\_N**,** // ISP1362 Write

**output** OTG\_WR\_N**,** // ISP1362 Read

**output** OTG\_RST\_N**,** // ISP1362 Reset

**output** OTG\_FSPEED**,** // USB Full Speed, 0 = Enable, Z = Disable

**output** OTG\_LSPEED**,** // USB Low Speed, 0 = Enable, Z = Disable

**input** OTG\_INT0**,** // ISP1362 Interrupt 0

**input** OTG\_INT1**,** // ISP1362 Interrupt 1

**input** OTG\_DREQ0**,** // ISP1362 DMA Request 0

**input** OTG\_DREQ1**,** // ISP1362 DMA Request 1

**output** OTG\_DACK0\_N**,** // ISP1362 DMA Acknowledge 0

**output** OTG\_DACK1\_N**,** // ISP1362 DMA Acknowledge 1

// LCD Module 16X2

**inout** **[**7**:**0**]** LCD\_DATA**,** // LCD Data bus 8 bits

**output** LCD\_ON**,** // LCD Power ON/OFF

**output** LCD\_BLON**,** // LCD Back Light ON/OFF

**output** LCD\_RW**,** // LCD Read/Write Select, 0 = Write, 1 = Read

**output** LCD\_EN**,** // LCD Enable

**output** LCD\_RS**,** // LCD Command/Data Select, 0 = Command, 1 = Data

// SD Card Interface

**inout** SD\_DAT**,** // SD Card Data

**inout** SD\_DAT3**,** // SD Card Data 3

**inout** SD\_CMD**,** // SD Card Command Signal

**output** SD\_CLK**,** // SD Card Clock

// I2C

**inout** I2C\_SDAT**,** // I2C Data

**output** I2C\_SCLK**,** // I2C Clock

// PS2

**input** PS2\_DAT**,** // PS2 Data

**input** PS2\_CLK**,** // PS2 Clock

// USB JTAG link

**input** TDI**,** // CPLD -> FPGA (data in)

**input** TCK**,** // CPLD -> FPGA (clk)

**input** TCS**,** // CPLD -> FPGA (CS)

**output** TDO**,** // FPGA -> CPLD (data out)

// VGA

**output** VGA\_CLK**,** // VGA Clock

**output** VGA\_HS**,** // VGA H\_SYNC

**output** VGA\_VS**,** // VGA V\_SYNC

**output** VGA\_BLANK**,** // VGA BLANK

**output** VGA\_SYNC**,** // VGA SYNC

**output** **[**9**:**0**]** VGA\_R**,** // VGA Red[9:0]

**output** **[**9**:**0**]** VGA\_G**,** // VGA Green[9:0]

**output** **[**9**:**0**]** VGA\_B**,** // VGA Blue[9:0]

// Ethernet Interface

**inout** **[**15**:**0**]** ENET\_DATA**,** // DM9000A DATA bus 16Bits

**output** ENET\_CMD**,** // DM9000A Command/Data Select, 0 = Command, 1 = Data

**output** ENET\_CS\_N**,** // DM9000A Chip Select

**output** ENET\_WR\_N**,** // DM9000A Write

**output** ENET\_RD\_N**,** // DM9000A Read

**output** ENET\_RST\_N**,** // DM9000A Reset

**input** ENET\_INT**,** // DM9000A Interrupt

**output** ENET\_CLK**,** // DM9000A Clock 25 MHz

// Audio CODEC

**inout** AUD\_ADCLRCK**,** // Audio CODEC ADC LR Clock

**input** AUD\_ADCDAT**,** // Audio CODEC ADC Data

**inout** AUD\_DACLRCK**,** // Audio CODEC DAC LR Clock

**output** AUD\_DACDAT**,** // Audio CODEC DAC Data

**inout** AUD\_BCLK**,** // Audio CODEC Bit-Stream Clock

**output** AUD\_XCK**,** // Audio CODEC Chip Clock

// TV Decoder

**input** **[**7**:**0**]** TD\_DATA**,** // TV Decoder Data bus 8 bits

**input** TD\_HS**,** // TV Decoder H\_SYNC

**input** TD\_VS**,** // TV Decoder V\_SYNC

**output** TD\_RESET**,** // TV Decoder Reset

// GPIO

**inout** **[**35**:**0**]** GPIO\_0**,** // GPIO Connection 0

**inout** **[**35**:**0**]** GPIO\_1 // GPIO Connection 1

**);**

//Turn off all displays.

//assign HEX0 = 7'h7F;

//assign HEX1 = 7'h7F;

//assign HEX2 = 7'h7F;

//assign HEX3 = 7'h7F;

//assign HEX4 = 7'h7F;

//assign HEX5 = 7'h7F;

//assign HEX6 = 7'h7F;

//assign HEX7 = 7'h7F;

//assign LEDR = 18'h0;

//assign LEDG = 9'h0;

//Set all GPIO to tri-state.

**assign** GPIO\_0 **=** 36'hzzzzzzzzz**;**

**assign** GPIO\_1 **=** 36'hzzzzzzzzz**;**

//Disable audio codec.

//assign AUD\_DACDAT = 1'b0;

//assign AUD\_XCK = 1'b0;

//Disable DRAM.

**assign** DRAM\_ADDR **=** 12'h0**;**

**assign** DRAM\_BA\_0 **=** 1'b0**;**

**assign** DRAM\_BA\_1 **=** 1'b0**;**

**assign** DRAM\_CAS\_N **=** 1'b1**;**

**assign** DRAM\_CKE **=** 1'b0**;**

**assign** DRAM\_CLK **=** 1'b0**;**

**assign** DRAM\_CS\_N **=** 1'b1**;**

**assign** DRAM\_DQ **=** 16'hzzzz**;**

**assign** DRAM\_LDQM **=** 1'b0**;**

**assign** DRAM\_RAS\_N **=** 1'b1**;**

**assign** DRAM\_UDQM **=** 1'b0**;**

**assign** DRAM\_WE\_N **=** 1'b1**;**

//Disable Ethernet.

**assign** ENET\_CLK **=** 1'b0**;**

**assign** ENET\_CS\_N **=** 1'b1**;**

**assign** ENET\_CMD **=** 1'b0**;**

**assign** ENET\_DATA **=** 16'hzzzz**;**

**assign** ENET\_RD\_N **=** 1'b1**;**

**assign** ENET\_RST\_N **=** 1'b1**;**

**assign** ENET\_WR\_N **=** 1'b1**;**

//Disable flash.

**assign** FL\_ADDR **=** 22'h0**;**

**assign** FL\_CE\_N **=** 1'b1**;**

**assign** FL\_DQ **=** 8'hzz**;**

**assign** FL\_OE\_N **=** 1'b1**;**

**assign** FL\_RST\_N **=** 1'b1**;**

**assign** FL\_WE\_N **=** 1'b1**;**

//Disable LCD.

**assign** LCD\_BLON **=** 1'b0**;**

**assign** LCD\_DATA **=** 8'hzz**;**

**assign** LCD\_EN **=** 1'b0**;**

**assign** LCD\_ON **=** 1'b0**;**

**assign** LCD\_RS **=** 1'b0**;**

**assign** LCD\_RW **=** 1'b0**;**

//Disable OTG.

**assign** OTG\_ADDR **=** 2'h0**;**

**assign** OTG\_CS\_N **=** 1'b1**;**

**assign** OTG\_DACK0\_N **=** 1'b1**;**

**assign** OTG\_DACK1\_N **=** 1'b1**;**

**assign** OTG\_FSPEED **=** 1'b1**;**

**assign** OTG\_DATA **=** 16'hzzzz**;**

**assign** OTG\_LSPEED **=** 1'b1**;**

**assign** OTG\_RD\_N **=** 1'b1**;**

**assign** OTG\_RST\_N **=** 1'b1**;**

**assign** OTG\_WR\_N **=** 1'b1**;**

//Disable SDRAM.

**assign** SD\_DAT **=** 1'bz**;**

**assign** SD\_CLK **=** 1'b0**;**

//Disable SRAM.

**assign** SRAM\_ADDR **=** 18'h0**;**

**assign** SRAM\_CE\_N **=** 1'b1**;**

**assign** SRAM\_DQ **=** 16'hzzzz**;**

**assign** SRAM\_LB\_N **=** 1'b1**;**

**assign** SRAM\_OE\_N **=** 1'b1**;**

**assign** SRAM\_UB\_N **=** 1'b1**;**

**assign** SRAM\_WE\_N **=** 1'b1**;**

//Disable VGA.

/\*

assign VGA\_CLK = 1'b0;

assign VGA\_BLANK = 1'b0;

assign VGA\_SYNC = 1'b0;

assign VGA\_HS = 1'b0;

assign VGA\_VS = 1'b0;

assign VGA\_R = 10'h0;

assign VGA\_G = 10'h0;

assign VGA\_B = 10'h0;

\*/

//Disable all other peripherals.

//assign I2C\_SCLK = 1'b0;

**assign** IRDA\_TXD **=** 1'b0**;**

//assign TD\_RESET = 1'b0;

**assign** TDO **=** 1'b0**;**

**assign** UART\_TXD **=** 1'b0**;**

**wire** VGA\_CTRL\_CLK**;**

**wire** AUD\_CTRL\_CLK**;**

**wire** DLY\_RST**;**

**assign** TD\_RESET **=** 1'b1**;** // Allow 27 MHz

**assign** AUD\_ADCLRCK **=** AUD\_DACLRCK**;**

**assign** AUD\_XCK **=** AUD\_CTRL\_CLK**;**

Reset\_Delay r0 **(** **.**iCLK**(**CLOCK\_50**),.**oRESET**(**DLY\_RST**)** **);**

VGA\_Audio\_PLL p1 **(** **.**areset**(~**DLY\_RST**),.**inclk0**(**CLOCK\_27**),.**c0**(**VGA\_CTRL\_CLK**),.**c1**(**AUD\_CTRL\_CLK**),.**c2**(**VGA\_CLK**)** **);**

VGA\_Controller u1 **(** // Host Side

**.**iCursor\_RGB\_EN**(**4'b0111**),**

**.**oAddress**(**mVGA\_ADDR**),**

**.**oCoord\_X**(**Coord\_X**),**

**.**oCoord\_Y**(**Coord\_Y**),**

**.**iRed**(**mVGA\_R**),**

**.**iGreen**(**mVGA\_G**),**

**.**iBlue**(**mVGA\_B**),**

// VGA Side

**.**oVGA\_R**(**VGA\_R**),**

**.**oVGA\_G**(**VGA\_G**),**

**.**oVGA\_B**(**VGA\_B**),**

**.**oVGA\_H\_SYNC**(**VGA\_HS**),**

**.**oVGA\_V\_SYNC**(**VGA\_VS**),**

**.**oVGA\_SYNC**(**VGA\_SYNC**),**

**.**oVGA\_BLANK**(**VGA\_BLANK**),**

// Control Signal

**.**iCLK**(**VGA\_CTRL\_CLK**),**

**.**iRST\_N**(**DLY\_RST**)** **);**

**wire** **[**9**:**0**]** mVGA\_R**;** //memory output to VGA

**wire** **[**9**:**0**]** mVGA\_G**;**

**wire** **[**9**:**0**]** mVGA\_B**;**

**wire** **[**18**:**0**]** mVGA\_ADDR**;** //video memory address

**wire** **[**9**:**0**]** Coord\_X**,** Coord\_Y**;** //display coods

////////////////////////////////////

//DLA state machine variables

**wire** reset**;**

**reg** **[**3**:**0**]** state**;** //state machine

**reg** **[**30**:**0**]** x\_rand**;** //shift registers for random number gen

**reg** **[**28**:**0**]** y\_rand**;**

**wire** seed\_low\_bit**,** x\_low\_bit**,** y\_low\_bit**;** //rand low bits for SR

**reg** **[**9**:**0**]** x\_walker**;** //particle coords of random walker

**reg** **[**8**:**0**]** y\_walker**;**

**reg** **[**7**:**0**]** rule**;**

**reg** **[**3**:**0**]** sum**;** //neighbor sum

////////////////////////////////////

/\*From megaWizard:

module vga\_buffer (

address\_a, // use a for state machine

address\_b, // use b for VGA refresh

clock\_a,

clock\_b,

data\_a,

data\_b,

wren\_a,

wren\_b,

q\_a,

q\_b);\*/

// Show m4k on the VGA

// -- use m4k a for state machine

// -- use m4k b for VGA refresh

**wire** mem\_bit **;** //current data from m4k to VGA

**reg** disp\_bit **;** // registered data from m4k to VGA

**wire** state\_bit **;** // current data from m4k to state machine

**reg** we **;** // write enable for a

**reg** **[**18**:**0**]** addr\_reg **;** // for a

**reg** data\_reg **;** // for a

vga\_buffer display**(**

**.**address\_a **(**addr\_reg**)** **,**

**.**address\_b **({**Coord\_X**[**9**:**0**],**Coord\_Y**[**8**:**0**]}),** // vga current address

**.**clock\_a **(**VGA\_CTRL\_CLK**),**

**.**clock\_b **(**VGA\_CTRL\_CLK**),**

**.**data\_a **(**data\_reg**),**

**.**data\_b **(**1'b0**),** // never write on port b

**.**wren\_a **(**we**),**

**.**wren\_b **(**1'b0**),** // never write on port b

**.**q\_a **(**state\_bit**),**

**.**q\_b **(**mem\_bit**)** **);** // data used to update VGA

// make the color white

//reg [9:0] color\_r = {8{disp\_bit}} ;

//reg [9:0] color\_g = {10{disp\_bit}} ;

//reg [9:0] color\_b = {2{disp\_bit}} ;

**assign** mVGA\_R **=** **{**2**{**disp\_bit**}};**

**assign** mVGA\_G **=** **{**10**{**disp\_bit**}};**

**assign** mVGA\_B **=** **{**2**{**disp\_bit**}};**

// DLA state machine

**assign** reset **=** **~**KEY**[**0**];**

HexDigit H0**(**HEX0**,** rule**[**0**]);**

HexDigit H1**(**HEX1**,** rule**[**1**]);**

HexDigit H2**(**HEX2**,** rule**[**2**]);**

HexDigit H3**(**HEX3**,** rule**[**3**]);**

HexDigit H4**(**HEX4**,** rule**[**4**]);**

HexDigit H5**(**HEX5**,** rule**[**5**]);**

HexDigit H6**(**HEX6**,** rule**[**6**]);**

HexDigit H7**(**HEX7**,** rule**[**7**]);**

**assign** LEDR**[**7**:**0**]** **=** SW**[**7**:**0**];**

**assign** LEDG**[**0**]** **=** **~**KEY**[**0**];**

**assign** LEDG**[**1**]** **=** **~**KEY**[**0**];**

**assign** LEDG**[**2**]** **=** **~**KEY**[**1**];**

**assign** LEDG**[**3**]** **=** **~**KEY**[**1**];**

**assign** LEDG**[**4**]** **=** **~**KEY**[**2**];**

**assign** LEDG**[**5**]** **=** **~**KEY**[**2**];**

**assign** LEDG**[**6**]** **=** **~**KEY**[**3**];**

**assign** LEDG**[**7**]** **=** **~**KEY**[**3**];**

**reg** **[**7**:**0**]** current\_state **=** 8'b11111111**;**

**assign** LEDR**[**17**:**10**]** **=** current\_state**[**7**:**0**];**

**reg** reset\_val**;**

**assign** LEDG**[**8**]** **=** reset\_val**;**

**reg** **[**641**:**0**]** row\_data **=** 642'd0**;** //Extra 2 are for zero-padding

**reg** **[**641**:**0**]** prow\_data **=** 642'd0**;** //Extra 2 are for zero-padding

**reg** **[**8**:**0**]** current\_row **=** 9'd0**;**

**reg** **[**9**:**0**]** current\_col **=** 10'd1**;** //Start row pointer at the 1, due to zero padding.

**wire** **[**641**:**0**]** result**;**

**reg** print **=** 1'b1**;**

genvar index**;**

generate

**for** **(**index**=**1**;** index **<** 641**;** index**=**index**+**1**)**

**begin:** gen\_code\_label

Automaton A\_inst**(**result**[**index**],** prow\_data**[**index**+**1**:**index**-**1**],** rule**);**

**end**

endgenerate

//Randomly generated Seed for LFSRs

**reg** **[**639**:**0**]** prev\_random\_row **=** 640'hf9232a25a8c301011f64ff197a07037e15f2b9498b9dec83273c0dae73b1adbf1fa46164a15fa1057184b67f0c22d55ae3bf4f0c5a213cb2da6226b3520b3de83dea4fc9080c12544c4e2afb78206410**;**

**wire** **[**639**:**0**]** random\_row**;**

generate

**for** **(**index**=**0**;** index **<** 20**;** index**=**index**+**1**)**

**begin:** gen\_code\_label2

Random32 R\_inst**(**random\_row**[(**32**\***index **+** 31**):(**32**\***index**)],** prev\_random\_row**[(**32**\***index **+** 31**):(**32**\***index**)]);**

**end**

endgenerate

//Automaton Tester

//state names

**parameter** init**=**4'd0**,** test1**=**4'd1**,** test2**=**4'd2**,** test3**=**4'd3**,** test4**=**4'd4**,** test5**=**4'd5**,** test6**=**4'd6**,**

new\_row**=**4'd7**,** new\_row2**=**4'd8**,** chill**=**4'd9**,**

chill2**=**4'd10**,** init2**=**4'd11**,** draw\_walker1**=**4'd12**,** draw\_walker2**=**4'd13 **;**

**always** **@** **(negedge** VGA\_CTRL\_CLK**)**

**begin**

// register the m4k output for better timing on VGA

// negedge seems to work better than posedge

disp\_bit **<=** mem\_bit**;**

**end**

**always** **@** **(posedge** VGA\_CTRL\_CLK**)** //VGA\_CTRL\_CLK

**begin**

// register the m4k output for better timing on VGA

//disp\_bit <= mem\_bit;

**if** **(**reset**)** //synch reset assumes KEY0 is held down 1/60 second

**begin**

reset\_val **<=** 1'b1**;**

//clear the screen

addr\_reg **<=** **{**Coord\_X**[**9**:**0**],**Coord\_Y**[**8**:**0**]}** **;** // [17:0]

we **<=** 1'b1**;** //write some memory

data\_reg **<=** 1'b0**;** //write all zeros (black)

//read SW0:7

rule **<=** SW**[**7**:**0**];**

row\_data **<=** 642'd0**;**

prow\_data **<=** 642'd0**;**

**if** **(**SW**[**17**])** **begin**

row\_data **<=** **{**1'b0**,** prev\_random\_row**,** 1'b0**};** //The random rows are 640 bits and each row needs to be 642

**end**

**else** **begin**

row\_data**[**320**]** **<=** 1'b1**;**

**end**

state **<=** init**;** //first state in regular state machine

**end**

//begin state machine to modify display

**else**

**begin**

reset\_val **<=** 1'b0**;**

**case(**state**)**

// next three states write the inital dot

init**:** //write a single dot in the middle of the screen

**begin**

we **<=** 1'b1 **;**

addr\_reg **<=** **{**10'd320**,**9'd0**}** **;** //(x,y)

//write a white dot in the middle of the screen

data\_reg **<=** 1'b1 **;**

print **=** 1'b1**;**

current\_col **<=** 10'd1**;**

current\_row **<=** 9'd0**;**

current\_state **<=** 8'b10000000**;**

state **<=** test1 **;**

**end**

//

test1**:**

**begin**

we **<=** 1'b0**;**

//addr\_reg <= {10'd250,9'd0};

//data\_reg <= compute;

**if** **(**print **==** 1'b1**)** **begin**

addr\_reg **<=** **{**current\_col**,**current\_row**};**

data\_reg **<=** row\_data**[**642**-**current\_col**];**

//color\_r <= {prow\_data[current\_col+1:current\_col-1]{disp\_bit}};

//color\_g <= {prow\_data[current\_col+1:current\_col-1]{disp\_bit}};

//color\_b <= {prow\_data[current\_col+1:current\_col-1]{disp\_bit}};

**end**

current\_state **<=** 8'b01000000**;**

state **<=** test2**;**

**end**

//wait for write

test2**:**

**begin**

we **<=** 1'b1**;**

current\_state **<=** 8'b00100000**;**

state **<=** test3**;**

**end**

//updating pixels

test3**:**

**begin**

we **<=** 1'b0**;**

**if** **(**current\_col **>=** 10'd640**)** **begin**

current\_col **<=** 10'd1**;**

**if** **(**current\_row **>=** 9'd480**)** **begin**

//current\_row <= 9'd0;

print **<=** 1'b0**;**

state **<=** chill**;**

**end**

**else** **begin**

//if ( ~KEY[2] )

//begin

current\_row **<=** current\_row **+** 9'd1**;**

state **<=** new\_row**;**

//end

//else begin

//state <= test1;

//end

**end**

**end**

**else** **begin**

current\_col **<=** current\_col **+** 10'd1**;**

state **<=** test1**;**

**end**

current\_state **<=** 8'b00010000**;**

//addr\_reg <= {10'd250,9'd100};

//data\_reg <= compute;

**end**

//starting a new row of pixels

new\_row**:**

**begin**

**if** **(**print **==** 1'b1**)** **begin**

prow\_data **<=** row\_data**;**

**end**

current\_state **<=** 8'b00001000**;**

state **<=** new\_row2**;**

**end**

new\_row2**:**

**begin**

**if** **(**print **==** 1'b1**)** **begin**

row\_data **<=** result**;**

**end**

current\_state **<=** 8'b00000100**;**

state **<=** test1**;**

**end**

chill**:**

**begin**

//just chill

**if** **(~**KEY**[**3**])**

**begin**

state **<=** chill2**;**

**end**

prev\_random\_row **<=** random\_row**;**

current\_state **<=** 8'b00000010**;**

//if (state != test1) begin

//state <= chill;

//end

**end**

chill2**:**

**begin**

**if** **(**KEY**[**3**])**

**begin**

print **<=** 1'b1**;**

current\_row **<=** 9'd0**;**

current\_col **<=** 10'd1**;**

state **<=** test1**;**

**end**

**else** **begin**

state **<=** chill2**;**

**end**

current\_state **<=** 8'b00000001**;**

**end**

**default:**

**begin**

state **<=** chill**;**

**end**

**endcase**

**end** // else if ( KEY[3])

**end** // always @ (posedge VGA\_CTRL\_CLK)

**endmodule** //top module

//////////////////////////////////////////////

// Larger Automaton

**module** Automaton**(**outpixel**,** inpixels**,** rule**);**

**input** **[**2**:**0**]** inpixels**;**

**output** outpixel**;**

**reg** outpixel**;**

**input** **[**7**:**0**]** rule**;**

**always** **@** **(**inpixels**,** rule**)**

**begin**

**case** **(**inpixels**)**

3'd0**:** outpixel **<=** rule**[**0**];**

3'd1**:** outpixel **<=** rule**[**1**];**

3'd2**:** outpixel **<=** rule**[**2**];**

3'd3**:** outpixel **<=** rule**[**3**];**

3'd4**:** outpixel **<=** rule**[**4**];**

3'd5**:** outpixel **<=** rule**[**5**];**

3'd6**:** outpixel **<=** rule**[**6**];**

3'd7**:** outpixel **<=** rule**[**7**];**

**default:** outpixel **<=** 1'b0**;**

**endcase**

**end**

**endmodule**

//////////////////////////////////////////////

// Random Number Generator

**module** Random32**(**out**,** in**);**

**input** **[**31**:**0**]** in**;**

**output** **[**31**:**0**]** out**;**

**reg** **[**31**:**0**]** out**;**

**reg** newbit**;**

**always** **@** **(**in**)**

**begin**

newbit **=** in**[**31**]** **^** in**[**21**]** **^** in**[**1**]** **^** in**[**0**];**

out **=** **{**in**[**30**:**0**],** newbit**};**

**end**

**endmodule**

//////////////////////////////////////////////

// Decode one hex digit for LED 7-seg display

**module** HexDigit**(**segs**,** num**);**

**input** **[**3**:**0**]** num **;** //the hex digit to be displayed

**output** **[**6**:**0**]** segs **;** //actual LED segments

**reg** **[**6**:**0**]** segs **;**

**always** **@** **(**num**)**

**begin**

**case** **(**num**)**

4'h0**:** segs **=** 7'b1000000**;**

4'h1**:** segs **=** 7'b1111001**;**

4'h2**:** segs **=** 7'b0100100**;**

4'h3**:** segs **=** 7'b0110000**;**

4'h4**:** segs **=** 7'b0011001**;**

4'h5**:** segs **=** 7'b0010010**;**

4'h6**:** segs **=** 7'b0000010**;**

4'h7**:** segs **=** 7'b1111000**;**

4'h8**:** segs **=** 7'b0000000**;**

4'h9**:** segs **=** 7'b0010000**;**

4'ha**:** segs **=** 7'b0001000**;**

4'hb**:** segs **=** 7'b0000011**;**

4'hc**:** segs **=** 7'b1000110**;**

4'hd**:** segs **=** 7'b0100001**;**

4'he**:** segs **=** 7'b0000110**;**

4'hf**:** segs **=** 7'b0001110**;**

**default** segs **=** 7'b1111111**;**

**endcase**

**end**

**endmodule**

///////////////////////////////////////////////

////////// end of file //////////////////////////

1. c "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators", Xilinx, XAPP 052 July 7,1996, http://www.xilinx.com/support/documentation/application\_notes/xapp052.pdf [↑](#footnote-ref-1)