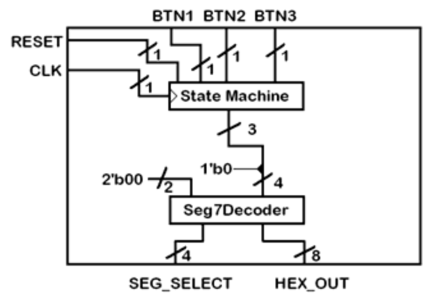
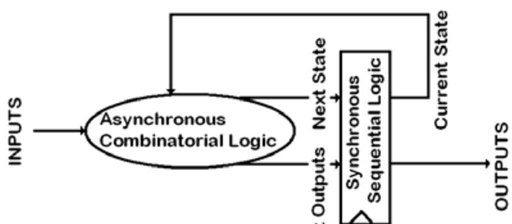
“Gateway” lab exercises

10. ColourTheWorld - *parameters* in generics; VGA display control to generate sync signals and RGB colors (see BASYS2 manual).
11. **WorldOfStateMachines - making *state machines* using sequential and combinational blocks (switch/case statements) and using ROM modules ($readmemb).**
12. WorldOfLinkedStateMachines - multiple state machines linked by a master state machine.
13. Snake - a complete snake game.

11. WorldOfStateMachines



11. WorldOfStateMachines




```
reg   [2:0]     CurrState;
reg   [2:0]     NextState;

assign OUT = CurrState[2:0];
```


11. WorldOfStateMachines

Two always@ statements

- Synchronous one

```
//Sequential
always@(posedge CLK) begin
   //This is a synchronous RESET
   if (RESET) begin
      //The state is reset to the rest state
      //usually this is 0
      CurrState    <= 3'b000;
      //The outputs are reset to their reset conditions
      CurrOutputs <= //Reset conditions for outputs
   end
   //For normal operations, on the rising edge
   //of the clock, the current State and Outputs
   //Get set to the next state and outputs.
   //They are then held at this value untill the
   //next rising edge of the clock.
   else begin
      CurrState    <= NextState;
      CurrOutputs <= NextOutputs;
   end
end
```

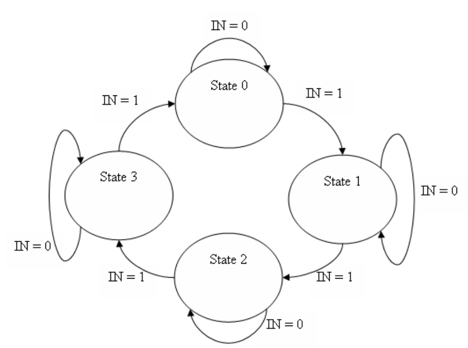
11. WorldOfStateMachines

Two always@ statements

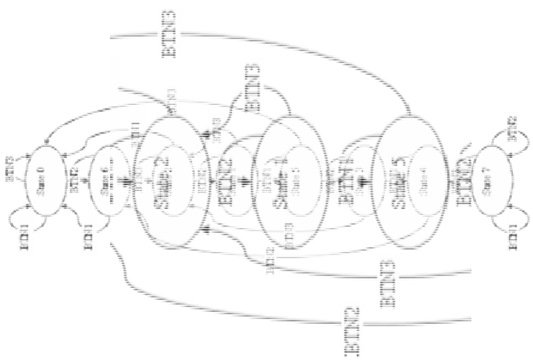
- Asynchronous one (combinational)

```
//Combinatorial
always@(CurrState or IN) begin
   case (CurrState)

      2'd0      : begin
         if (IN)
            NextState    <= 2'd1;
         else
            NextState    <= CurrState;
      end

      default   :
            NextState    <= 2'd0;
   endcase
end
```

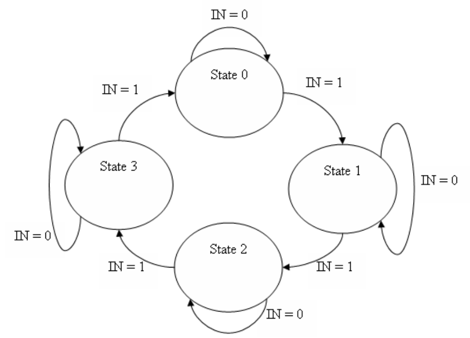
11. WorldOfStateMachines



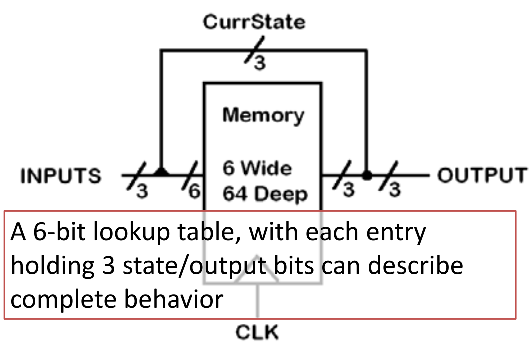
11. WorldOfStateMachines: State Maze



11. WorldOfStateMachines



11. WorldOfStateMachines
Using memory look-up table to compute next state



A 6-bit lookup table, with each entry holding 3 state/output bits can describe complete behavior

11. WorldOfStateMachines
Using memory look-up table to compute next state

```
   //Loads a ROM configuration from a file to
   //the two dimentional array
   initial
      $readmemb("SM_Mem.list", NextStateMem);

   //Simple synchronous system that sets the currstate
   //to state 0 given the reset command, and if this
   //is not the case, it fetches the nextstate from the
   //memory using the current state and the inputs
   //BTN1, BTN2, and BTN3 as the address.
   always@(posedge CLK) begin
      if (RESET)
         CurrState <= 3'h0;
      else
         CurrState <= NextStateMem[{CurrState,BTN3,BTN2,BTN1}];
   end

   assign OUT = CurrState;

endmodule
```


11. WorldOfStateMachines
Using memory look-up table to compute next state

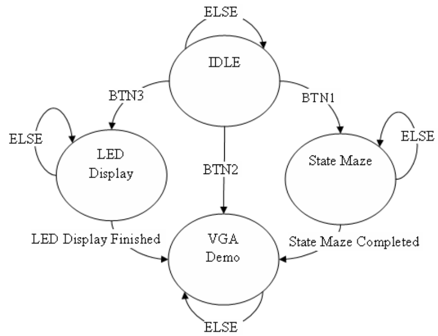
Pros
- Simple logic
- Uses the built-in FPGA bit memory (there's lots)
- Forces you to consider each state
- Lookup table can be computed e.g. in matlab then saved
- No need to figure out logic functions

Cons
- Only good for very small state machines

"Gateway" lab exercises

10. ColourTheWorld - *parameters* in generics; VGA display control to generate sync signals and RGB colors (see BASYS2 manual).
11. WorldOfStateMachines - making ***state machines*** using sequential and combinational blocks (switch/case statements) and using ROM modules (**$readmemb**).
12. **WorldOfLinkedStateMachines - multiple state machines linked by a master state machine.**
13. Snake - a complete snake game.

12. WorldOfLinkedStateMachines



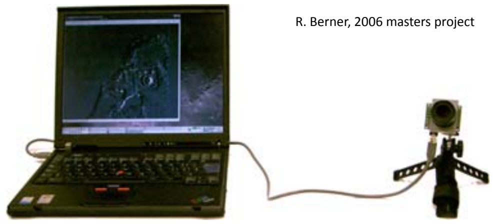
12. WorldOfLinkedStateMachines

Each of the three sub-modules (the State maze, VGA interface, and LED Display) all take the master state machine's state as an input, using it to determine when they must change their behavior (go from a blank to a colorful display) or move their own state machines from their idle position. Additionally, the two sub-modules that comprise a state machine, output their state so that it can be used by the master state machine in determining when each of the sub modules have completed their tasks. In this way, the master state machine will only progress from the "LED Display state" or the "State Maze state" to the "VGA Interface state" after the corresponding sub modules pass through their "Finished state".

USB silicon retina

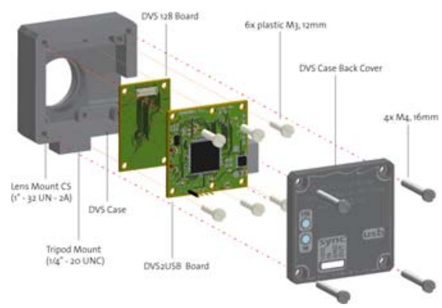
An example of a useful handshaking/data transfer state machine

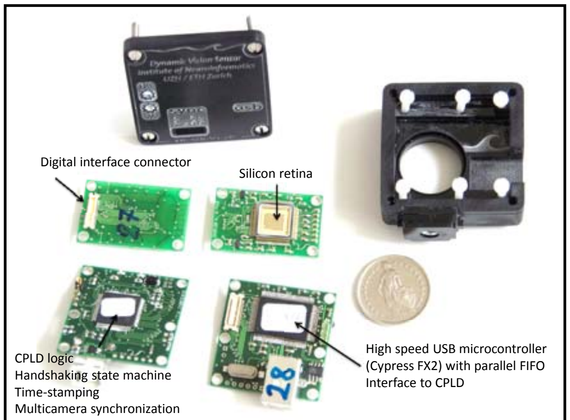
R. Berner, 2006 masters project



Camera PCB acquires asynchronous digital data from retina chip, time stamps it, sends data over high speed USB interface to PC. It also receives control commands for biasing, etc. from PC and sends to retina chip serially.

http://www.ini.uzh.ch/~tobi/studentProjectReports/bernerUSB2AER2006.pdf





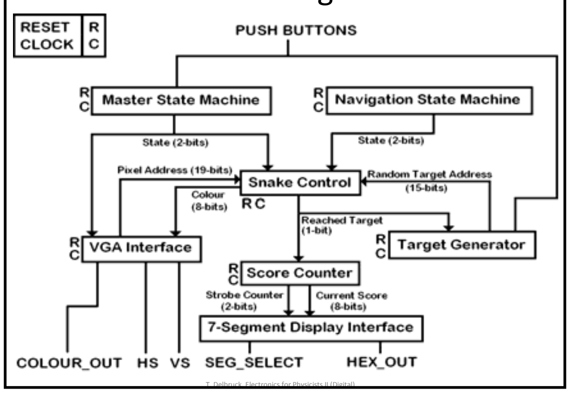
"Gateway" lab exercises

10. ColourTheWorld - *parameters* in generics; VGA display control to generate sync signals and RGB colors (see BASYS2 manual).
11. WorldOfStateMachines - making ***state machines*** using sequential and combinational blocks (switch/case statements) and using ROM modules (**$readmemb**).
12. WorldOfLinkedStateMachines - multiple state machines linked by a master state machine.
13. Snake - a complete snake game.

12. Snake game

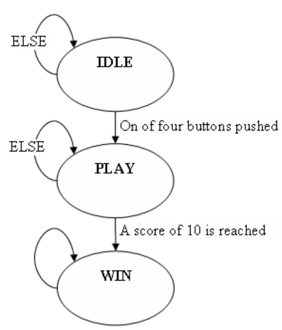
http://youtu.be/iB3tXDpL9hI

12. Snake game



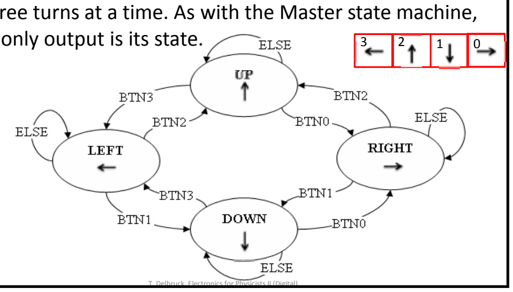
12. Snake Master state machine

The master state machine consists of three states, **IDLE**, **PLAY**, and **WIN**. The RESET signal resets the State Machine to IDLE. If any push button is pressed, the State Machine transitions into the PLAY state. When the Score, which is collected by the Score Counter module, reaches 10, the State Machine transitions from PLAY to WIN, where it stays until the system is reset. The only output of this module is its state which is used to control the other modules.



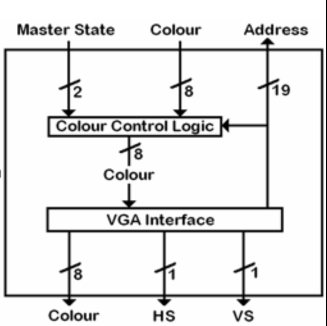
12. Snake Navigation machine

This state machine takes its input from the push buttons, and dictates the direction the snake should be travelling. The State Machine should be designed to only allow 90 degree turns at a time. As with the Master state machine, the only output is its state.



12. Snake VGA machine

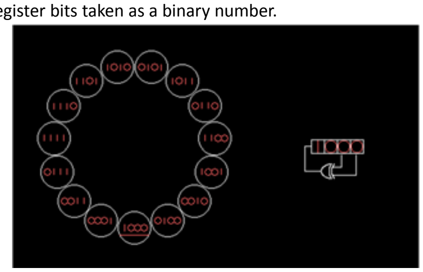
Passes to snake control machine address information *out*, and gets color information *in*. However, when the Master State Machine is not in the PLAY state, it also needs to overwrite the incoming color information to display either a blank blue screen when in the IDLE state, or the multi-color animated display from the previous exercise when in the WIN state. The address bus is combined X axis (horizontal [9:0]) and Y axis (vertical [8:0]).



12. Snake Target Generator

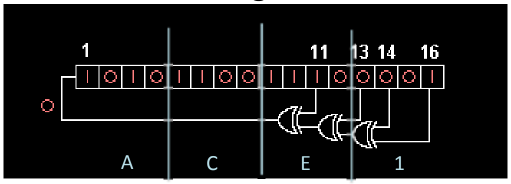
Pseudo-Random Number Generator (PRNG)

- Based on Linear Feedback Shift Register (LFSR) – **input** comes from XORed bits (actually XNORed). **Output** is the shift register bits taken as a binary number.



http://en.wikipedia.org/wiki/Linear_feedback_shift_register

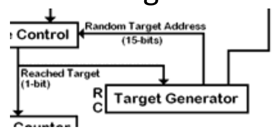
12. Snake Target Generator



A 16-bit Fibonacci LFSR. The feedback tap numbers in white correspond to a "primitive polynomial" so the register cycles through the maximum number of 65535 states excluding the all-zeroes state. The state shown, 0xACE1 (hexadecimal) will be followed by 0x5670.

http://en.wikipedia.org/wiki/Linear_feedback_shift_register

12. Snake Target Generator



This module randomly generates the position of the next target.

1. This is done using two pseudo random number generators.
2. The Snake Control module determines the color of a pixel based upon its address, ignoring the 2 lsbs, thereby reducing the resolution by a factor of four in each direction, from 640x480 to 160x120.
3. We therefore require two LFSRs, one 8-bits long, the other 7-bits, to be able to randomly generate numbers between 0-159 and 0-119.
4. A second process is required that records and holds the current value of the LFSR whenever the reached target signal is asserted, thereby allowing the LFSRs to constantly shift in the background
5. The PRNG will generate numbers outside these ranges, so the Target Generator detects this and instead generates a fixed location in these cases.

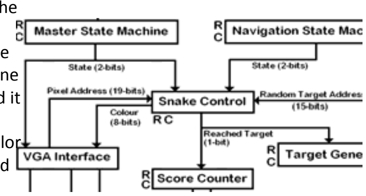
Download snake-target-generator for verilog PRNG for target generation

12. Snake ScoreCounter

Like CountingInDecimel but triggered by external enable. Instantiates Seg7Decoder.

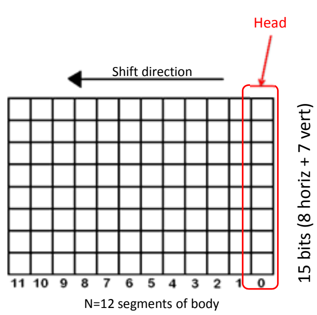
12. Snake controller SnakeControl

This module has two functions: it stores the current position of all the elements of the snake, using the input from the Navigation State Machine to alter its position; and it then translates this position into a serial color stream that is presented to the VGA Display module, which passes it to the monitor. This is by far the most complex module that you will create.

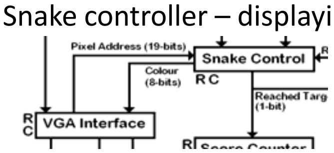


12. Snake controller SnakeControl

The snake is represented as a two dimensional shift register, with a size of 15 by N, where N is the length of the snake. Each element (15-bits) corresponds to the (X,Y) coordinates of that part of the snake. At each clock cycle, all the bits shift in the N direction, and a new 15-bit element is added to the end of the shift register. This element is constructed based upon the previous first element, incrementing or decrementing either the X or Y address depending on the direction the snake is moving.



12. Snake controller – displaying snake

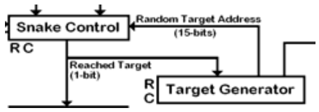


Two address buses come from the VGA Display module to present the address of the pixel that is about to be displayed.

A synchronous process can then decide what color that pixel should be, by seeing if its address corresponds to any of the Snake sections, or to the target.

If the pixel belongs to the snake, it should be colored yellow, if it belongs to the target, it should be colored red, if it does not correspond to either then it should be colored blue.

12. Snake controller – determining target eaten



A synchronous process is required to determine if the head of the snake (the first element of the two dimensional array) has the same position (address) as the target provided by the linear feedback shift registers.

If this is the case then the output REACHED_TARGET is asserted.

If this is not the case, then REACHED_TARGET is de-asserted.

Asserting REACHED_TARGET will cause a new target to be selected and the score to be incremented.

12. Snake game – use parameters

```
reg [7:0] SnakeState_X [0:SnakeLength-1];
reg [6:0] SnakeState_Y [0:SnakeLength-1];

//Changing the position of the snake registers
//Shift the snakestate x and y
genvar PixNo;
generate
   for (PixNo=0; PixNo < SnakeLength-1; PixNo=PixNo+1)
   begin: PixShift
      always@(posedge CLK) begin
         if(RESET) begin
            SnakeState_X[PixNo+1] <= 80;
            SnakeState_Y[PixNo+1] <= 100;
         end
         else if (Counter == 0) begin
            SnakeState_X[PixNo+1] <= SnakeState_X[PixNo];
            SnakeState_Y[PixNo+1] <= SnakeState_Y[PixNo];
         end
      end
   end
endgenerate

// replace top snake state with new one based on direction
always@(posedge CLK) begin
   if(RESET) begin
      //Set the initial state of the snake
      SnakeState_X[0] <= 80;
      SnakeState_Y[0] <= 100;
   end
   else if (Counter == 0) begin
      case (DIRECTION)
         2'b00       : begin
            if( SnakeState_Y[0] == 0)  SnakeState_Y[0] <= MaxY;
            else                       SnakeState_Y[0] <= SnakeState_Y[0] - 1;
         end
```

This code is repeated by the generate statement a number of times equal to the value of the parameter "SnakeLength".

At each repetition, the value of the generate variable PixNo increments, thereby changing which bits of the two dimensional array are targeted.

12. Snake game – use parameters

After all the sub-modules have been created, it is a relatively simple task to tie them all together. There will probably be many errors in your code, and you will not notice them until you implement your final design and load it on to your FPGA.

Priority should be given to making sure that the VGA Display and the user interface works properly. Do not be afraid to simulate each of your sub modules to make sure they work as you intend. Often this will save you a lot of time in the long run.

Once you have got your snake working, you can try to implement:

- Determine failure when the snake hits itself
- Making snake grow with each target that is acquired
- Making snake move faster with each target that is acquired
- Making new end of game graphical display
- Adding sound effects using external speaker and PWN sound generation

Options for future development

For high-speed USB

- Cypress FX2/FX3

- uC+FTDI (e.g. FT232H).

For FPGA

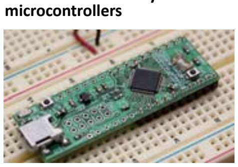
- Lattice
  - simpler environment, flexible voltage options
- Altera
  - embedded NIOS processor
- Xilinx
  - industry leader, Zynq processor

For embedded system

- Your AVR32 bronze board. You know it
- Arduino Atmel uC, easy to learn, lots of users, lots of boards
- Rasberry Pi ARM11, embedded linux, very active lately

What you accomplished

1st half: Embedded systems with microcontrollers

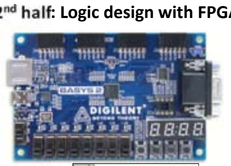


PCB SMD assembly

PCB design and layout

+ embedded systems programming, ADC, PWM, DSP

2nd half: Logic design with FPGA



+ synchronous logic, verilog, state machines, counters, VGA control etc