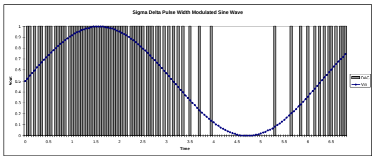
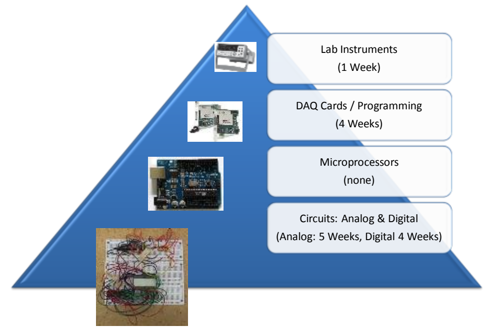
FPGA & Pulse Width Modulation

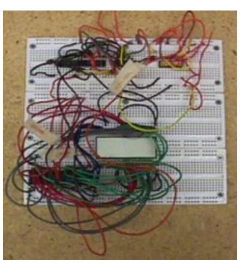


BFY
BFY Workshop 30 University of Maryland 2015
Kurt Wick (wick@umn.edu)
University of Minnesota

Time Allotment During the First 14 Weeks of Our Advanced Lab Course



Digital Logic



- Rules of Digital logic are relatively few but (interesting) designs often require:
  - **Large number of components**
  - **Fast clock speeds**
- Conclusion: today's circuit very rarely use the old ASSP (*A*pplication-*S*pecific *S*tandard *P*roducts) such as the74 series gates.
- They either use ASIC (*A*pplication *S*pecific *I*ntegrated *C*ircuits) or FPGAs or CPLDs.

Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs)

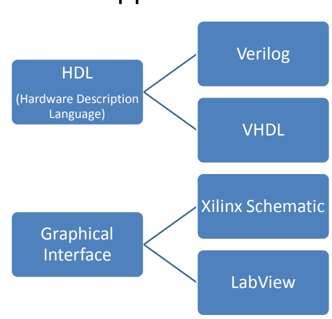
- Number of Generic Gates: 100 000 to a few millions.
- Number of Flip-Flops: 2000 and above
- Clock Speed: 25 MHz up to GHz range.
- Pins: 100 to 400.
- Cost: a few dollars

XILINX
ALL PROGRAMMABLE.

Programing the FPGA

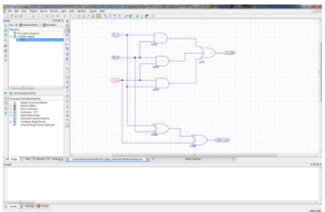
- Step1: Implement Your Design
- Step 2: Assign Pins to I/O Ports
- Step 3: Generate Bit File / Firmware
- Step 4: Transfer Bit File to FPGA

Step 1: Implementing Your Design Approaches

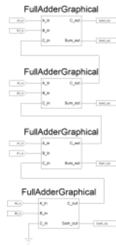


Implementing Your Design in a Graphical Environment: Creating a 4 Bit Full Adder

Simple 1 Bit Adder

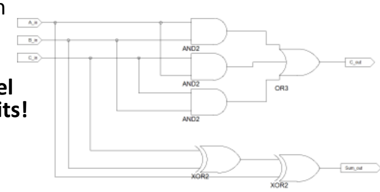


4-Bit Adder



Implementing Your Design in a Graphical Environment (continued)

- Eas(ier) to learn than HDL
- **Displays parallel nature of circuits!**
- No Standard
- LabView Version will only work with LabView boards.



Step 1: Implementing Your Design in HDL

Verilog

- Users: 50%
- C-Based: Case sensitive
- IEEE Standard 1364-2005

```
module MyAndGate(
      input a,
      input b,
      output q);

assign q = a&b;

endmodule
```

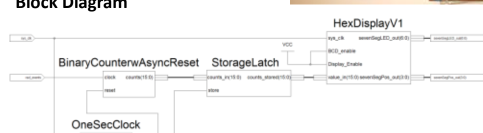
VHDL

- Users: 50%
- Based on ADA: not case sensitive
- IEEE Standard 1076-2008

```
-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity ANDGATE is
  port (
    I1 : in std_logic;
    I2 : in std_logic;
    O  : out std_logic);
end entity ANDGATE;

-- this is the architecture
architecture RTL of ANDGATE is
begin
  O <= I1 and I2;
end architecture RTL;
```

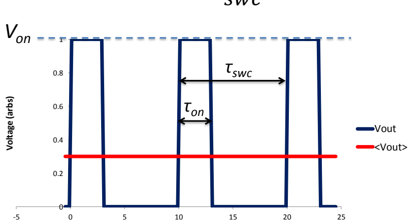
Student Lab Assignment: Radiation Counter

Block Diagram

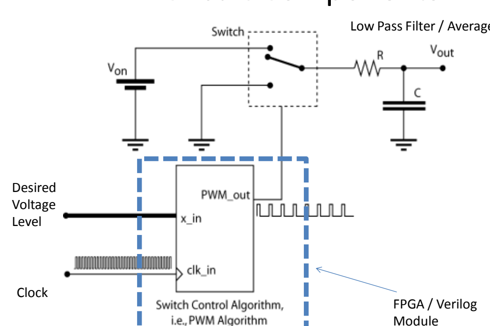


PWM Theory

$$\langle V_{out} \rangle = \frac{\tau_{on}}{\tau_{swc}} V_{on}$$



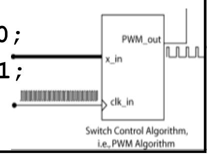
PWM Circuit Components



Simple PWM Control Algorithm

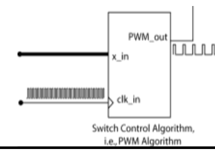
- *An n-bit counter continuously increments from 0 to its maximum value, i.e., 2^n-1 and then repeats the cycle.*
- *Range of input value x_in: $0 \le x\_in \le 2^n-1$*

```
if ( counter < x_in )
        PWM_out <= 1;
    else
        PWM_out <= 0;
counter <= counter+1;
```



Verilog Implementation of Simple 8 Bit PWM Algorithm

```
module SimplePWM(clk_in, x_in, PWM_out);

        input clk_in;   //clock for counter
        input [7:0] x_in;               //control value that
                                        //defines pulse width
        output reg PWM_out = 1;         //PWM signal out

        reg [7:0] counter = 0;

        always@ (posedge clk_in )begin
                if ( counter < x_in )
                        PWM_out <= 1;
                else
                        PWM_out <= 0;
                counter <= counter+1;
                end
endmodule
```



A2D Key Concepts: **Resolution / Sensitivity**

- Resolution of an n-bit PWM A2D is: $\mathbf{V_{on} / 2^n}$
- (Hypothetical) Resolution for our BASYS board PWM A2D with V_{on} = 5 Volts would be:

bits	Resolution (Volts)
8	1.9E-02
16	7.6E-05
32	1.2E-09
64	2.7E-19

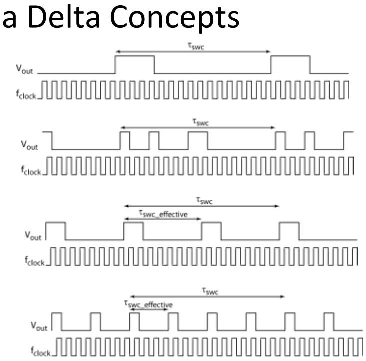
A2D Key Concepts: **Conversion Time**

- It takes at least one complete counter cycle to average V_{out}. For a counter running at f_o, this corresponds to: $\mathbf{2^n/f_o}$
- (Optimal) Conversion Time for our BASYS board PWM A2D with f_o = 25 MHz:

Bits	Resolution (Volts)	Conversion Time
8	1.9E-02	10.2 usec
16	7.6E-05	2.6 msec
32	1.2E-09	2.8 min
64	2.7E-19	23000 years

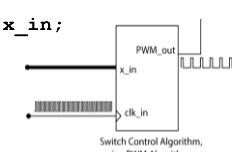
Sigma Delta Concepts

Though $V_{out}(t)$ is different in each of these timing diagrams, $\langle V_{out} \rangle$ remains identical.

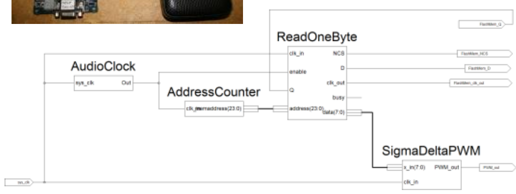


Sigma Delta PWM Algorithm

```
if ( Sigma >= Delta )
    {
    Sigma = (Sigma - Delta) + x_in;
    Out = 1;
    }
else
    {
    Sigma = Sigma + x_in;
    Out = 0;
    }
```



PWM Application: Music Player



Conclusions about PWM

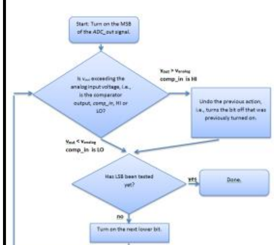
- Pulse Width Modulation can be used to create an analog signal from a digital signal.
- Allows the reduction of a DC signal while being much more energy efficient than, for example, a passive voltage divider.
- Sigma Delta algorithm can also be used for voltage to frequency conversion.

Educational Goals

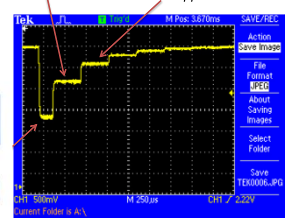
- The PWM exercises expose students to basic digital concepts such as clocks and counters.
- They are easily implemented with an FPGA and thereby exposes students with this ubiquitous electronic component.
- Familiarizes them with digital-to-analog converters and the basic concepts of resolution and conversion time.
- The exercise can be extended by turning it into an analog-to-digital converter using successive approximation and a state machine.

Successive Approximation A2D

Verilog State Machine



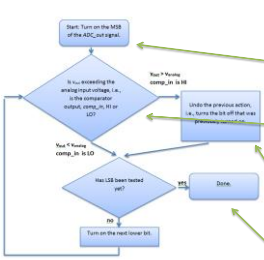
2nd approximation *3rd approximation*



Vout vs. t showing the different "guesses" and How the circuit approaches the final value.

Successive Approximation A2D

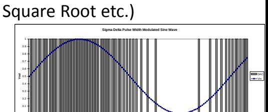
Verilog State Machine



```
reg [7:0] mask_now = 8'b1000_0000;
reg [7:0] mask_next = 8'b0100_0000;

parameter ADJ_MASK = 2'b00;
parameter INIT = 2'b01;
parameter COMP_ADJUST = 2'b10;
parameter DONE = 2'b11;

  (* FSM_ENCODING="SEQUENTIAL", SAFE_IMPLEMENTATION="YES",
SAFE_RECOVERY_STATE="<recovery_state_value>" *) reg [1:0] state = INIT;

  always@(posedge sys_clk)
     (* PARALLEL_CASE, FULL_CASE *) case (state)

     INIT : begin        //initalize all parameters
                state <= COMP_ADJUST;

                mask_now <= 8'b1000_0000;
                mask_next <= 8'b0100_0000;
                ADC_out <= 8'b1000_0000;
                busy <= 1;
      end

      COMP_ADJUST : begin //adjust DAC output based on comp_in value; also set next lower bit
                if (mask_now == 1)
                   state <= DONE;
                else
                   state <= ADJ_MASK;
                if( comp_in )
                     ADC_out <= (ADC_out ^ mask_now) | mask_next;
                else
                     ADC_out <= ADC_out | mask_next;
      end

     ADJ_MASK : begin      //adjust the masks which define the bits that are currently guessed
                state <= COMP_ADJUST;
                mask_now <= mask_now >> 1;
                mask_next <= mask_next >> 1;
     end

    DONE : begin        //send the conversion done message
                state <= INIT;
                busy <= 0;
      end

   default : begin // Fault Recovery
         state <= INIT;
      end
     endcase
```

IP (Intellectual Property) Cores

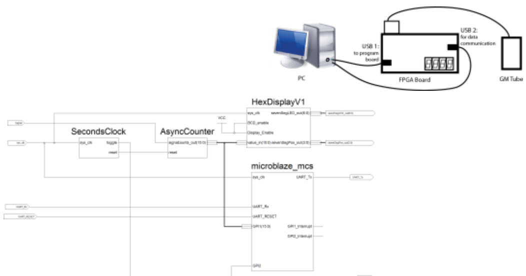
- Add New Source: in the Source Wizard select: IP (CORE Generator and Architecture Wizard.)
- Free IP Cores in the ISE Webpack:
  - Signal Processing: FIR Filters
  - Transforms: FFT
  - Math Functions (Trig, Divide, Square Root etc.)
  - Embedded Microprocessors



- Using a Trigonometric IP Core to Create an Arbitrary Waveform

Additional Projects: Using IP Cores for Embedded Designs

Using a Microprocessor to Communicate with the PC



Additional Resources

- Books
  - Advanced Digital Design with the Verilog HDL (2nd Edition) [Hardcover] Michael D. Ciletti
- Web Sites
  - Opencores is a website that lets users upload and download their VHDL and Verilog code. It contains a large repository for communications controllers (SPI, Serial etc) and microprocessors: http://opencores.org/
  - Our Wiki Page: https://wiki.umn.edu/MXP/AlphaWorkshop