FPGA Design Challenge :Techkriti'14

Digital Design using Verilog – Part 2

Anurag Dwivedi

Recap

- Verilog- Hardware Description Language
- Modules
- Combinational circuits
- assign statement
- Control statements
- Sequential circuits
- always@ block
- Blocking and non-blocking statements

Modular Circuits

- Time to construct larger modules from smaller modules.
- Various modules are interconnected to make a larger circuit (or module).
- Each sub-module has a separate Verilog file.
- A sub-module may have another sub-module in its circuit.
- One needs to indicate the top level module before synthesis.

Example

- Sub-module 1

  module Sub1 (input wire [7:0] a1, output wire [7:0] b1);

- Sub-module 2

  module Sub2 (input wire [7:0] a2, output wire[7:0] b2);

- Top Module

  module Top (input wire [7:0] a, output wire[7:0] b);

Instantiation

- Used to create instances of the module and create connections among different modules.
- In the above example, we need to instantiate the two sub-level modules in the top module.
- This is done as follows:

```
wire [7:0] c;
Sub1 Encoder (.a1(a), .b1(c));
Sub2 Decoder (.a2(c), .b2(b));
```

Example : Full Adder

```
module FAdder(
  input wire [3:0] A, B,
  output wire cout,
  output wire [3:0] S  );

wire c0, c1, c2;

  FA fa0( .a(A[0]), .b(B[0]), .cin(0),.cout(c0), .sum(S[0]));
  FA fa1( .a(A[1]), .b(B[1]), .cin(c0), .cout(c1), .sum(S[1]));
  FA fa2( .a(A[2]), .b(B[2]), .cin(c1), .cout(c2), .sum(S[2]));
  FA fa3( .a(A[3]), .b(B[3]), .cin(c2), .cout(cout), .sum(S[3]));

endmodule
```

```
module FA(
        input wire a, b, cin,
        output wire sum,
        output wire cout
);
    assign { carry, sum } = a+b +cin;
endmodule
```

Points to note

- All output ports of instantiated sub-module should be of wire data-type.
- Note in previous example, c0,c1,c2 and S are wires.
- Inputs may be reg or wire.
- Suppose in above, [3:0] S was of reg type.
  - Declare a dummy wire variable [3:0] add
  - Pass add[0], add[1] ... to the instantiations
  - Finally put:
    - always@(*)
    - S <= add;

Parameterized Modules

- A generalized type of module.
- Can be instantiated to any value of parameter.
- Parameterization is a good practice for reusable modules.
- Useful in large circuits.

N-Bit adder

```
module AdderN #(parameter N = 4)(
  input wire [N-1:0] IN1,
  input wire [N-1:0] IN2,
  output reg [N-1:0] OUT  );

always @(*)
  OUT <= IN1 + IN2;

endmodule
```

Instantiation of Parameterized Modules

<Module Name> #(.<Parameter Name>(value)) <Instance name> (.IN1(...) , .IN2(...), .OUT1(...), .OUT2(...));

Example :

AdderN # (.N(16)) Add16 (.IN1(in1), .IN2(in2), .OUT(out));

Test Bench

- Used to test the functionality of design by simulation.
- Instantiate our top most module and give varying inputs & verify if the outputs match expected results.
- Added functionalities in Test Bench:
  - Delays
  - $display(), $monitor()

Delays

- Not synthesized
- Can be used to model delays in actual circuit during simulation

- Used mostly in Test Benches to provide inputs at particular instants.

- Syntax: #<time steps>
  - #10 q = x + y; // inter assignment delay
  - q = #10 x + y;   // intra assignment delay
  - Most common:

always
#5 clk = ~clk;

More features

- $display()
  - used for printing text or variables to screen
  - syntax is the same as for printf in C
  - $display("time, \tclk, \tenable, \tcount");

- $monitor()
  - keeps track of changes to the variables in the list
  - whenever any of them changes, it prints all the values
  - only written once in initial block.
  - $monitor("%d,\t%b,\t%b,\t%b,\t%d",$time, clk, enable,count);

- $finish
  - terminating simulation

Test Bench Counter

```
module counter_tb;
 reg clk, reset, enable;
 wire [3:0] count;

 counter C0( .clk (clk), .reset (reset), .enable (enable), .count (count) );
 initial begin
 clk = 0;
    reset = 0;
    enable = 0;
  end
 always
 #5 clk = !clk;
 initial begin
 $display("time,\tclk,\tenable,\tcount");
 $monitor("%d,\t%b,\t%b,\t%d",$time, clk,enable,count);
  end
  initial
 #100 $finish;
endmodule
```

Demonstration

- 4029 Counter
- 4-bit Full Adder

FPGA Design Challenge Techkriti'14

Problem Statement :

The challenge in FPGA is to design and efficiently implement the Hilbert transformation of any given function.

It maps a time domain function to another time domain function by a convolution of the input signal with the function H(t) whose representation in frequency domain is:

$$\sigma_H(\omega) = \begin{cases} i = e^{+\frac{i\pi}{2}}, & \text{for } \omega < 0 \\ 0, & \text{for } \omega = 0 \\ -i = e^{-\frac{i\pi}{2}}, & \text{for } \omega > 0 \end{cases}$$

Frequency domain representation of Hilbert Transform

Fourier Transform

- FT is a mathematical transformation to transform a signal from time domain to frequency domain.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i x \xi}\, dx, \quad \text{for any real number } \xi.$$

Basic Theory

- If x(t) is our signal and h(t) is the signal corresponding to Hilbert Transform, then the Hilbert transform of x(t), hx(t) = x(t)*h(t)
- FT(hx(t)) = FT(x(t)*h(t)) = X(f) x H(f)
- Here, X(f) is the Fourier transform of x(t) and H(f) is the Fourier Transform of h(t) given by

$$\sigma_H(\omega) = \begin{cases} i = e^{+\frac{i\pi}{2}}, & \text{for } \omega < 0 \\ 0, & \text{for } \omega = 0 \\ -i = e^{-\frac{i\pi}{2}}, & \text{for } \omega > 0 \end{cases}$$

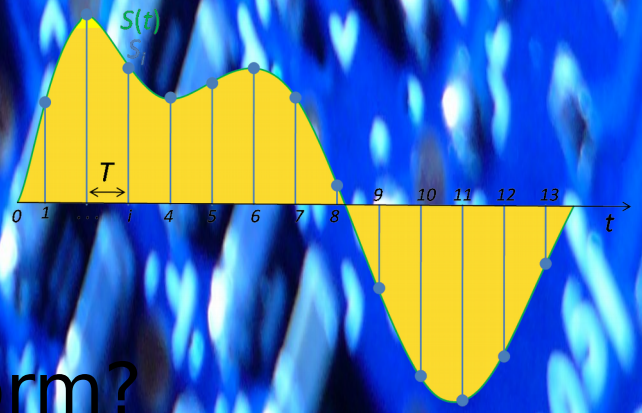
- hx(t) = IFT(X(f) x H(f))

Basic Theory

- Find Fourier Transform of the function.
- Multiply the FT by H(f)
- Apply inverse Fourier Transform of the function obtained.

Discrete Fourier Transform

- How to model continuous time signals in our digital hardware?
  - Work with samples of signal
  - xi[n] = x (nT) ; n = 0,1 ... N-1
  - T is the sampling period
- How to take Fourier transform?
  - Take Discrete Fourier Transform



$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}.$$

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \omega^{kn}.$$

$$\omega = e^{-2\pi i/N}$$

DFT Matrix

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \omega^{kn}.$$

$$\begin{pmatrix} F[0] \\ F[1] \\ F[2] \\ \vdots \\ F[N-1] \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{N-2} \\ 1 & W^3 & W^6 & W^9 & \dots & W^{N-3} \\ \vdots & & & & & \\ 1 & W^{N-1} & W^{N-2} & W^{N-3} & \dots & W \end{pmatrix} \begin{pmatrix} f[0] \\ f[1] \\ f[2] \\ \vdots \\ f[N-1] \end{pmatrix}$$

where $W = \exp(-j2\pi/N)$ and $W = W^{2N}$ etc. $= 1$.

- Total number of operations : O(N^2)

Fast Fourier Transform

- Computation of DFT involves a lot of redundant operations
- Time complexity can be reduced drastically by using this information
- A class of algorithms known as Fast Fourier Transform (FFT) is developed for the same

Cooley–Tukey algorithm : Radix 2 case

- When N is a factor of 2
- We have

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk},$$

- Writing the odd and even numbered terms separately

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2} mk}}_{\text{DFT of even-indexed part of } x_m} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2} mk}}_{\text{DFT of odd-indexed part of } x_m} = E_k + e^{-\frac{2\pi i}{N}k} O_k.$$

- Taking out the common factor

Cooley–Tukey algorithm : Radix 2 case

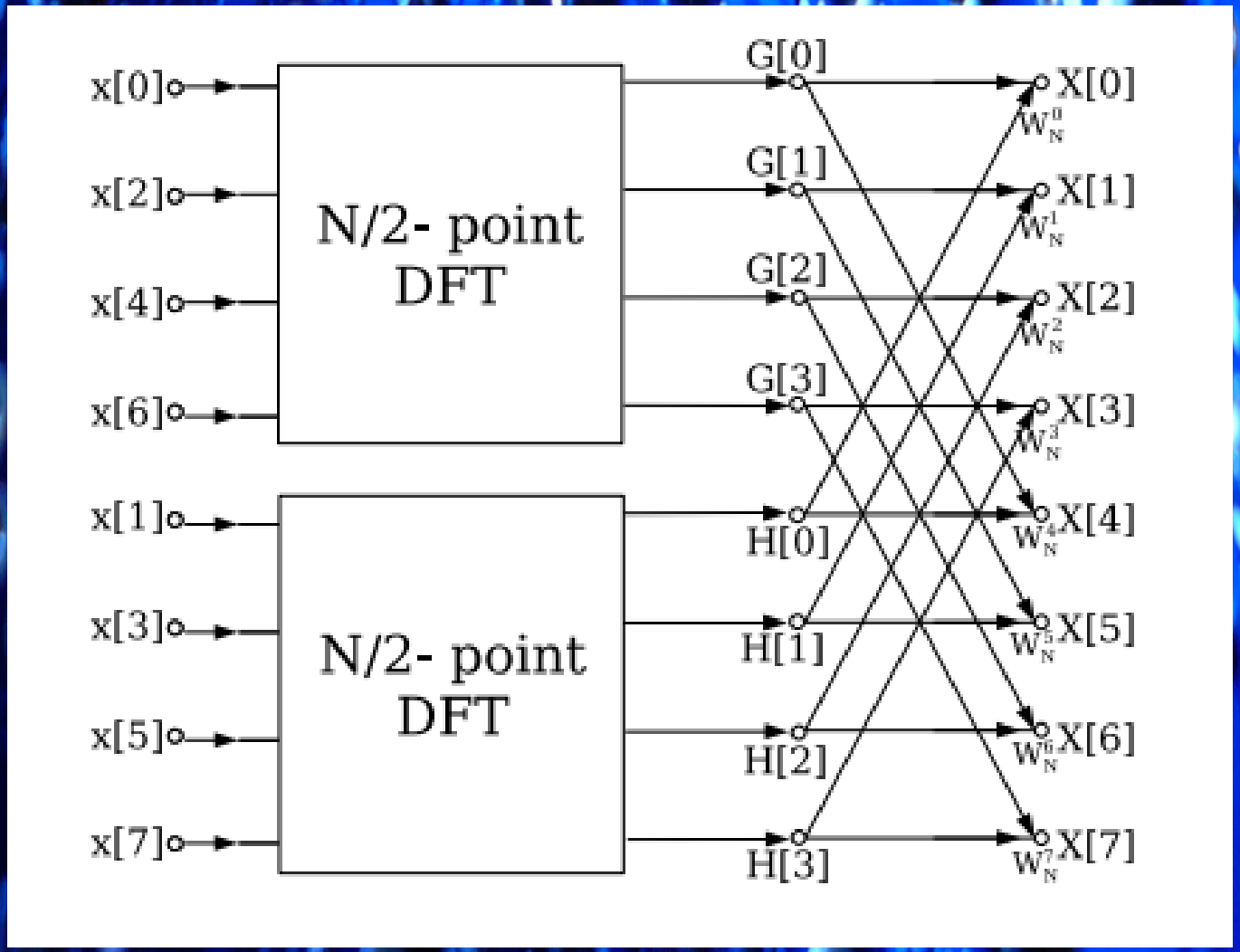
- Because of Periodicity of DFT we have

$$E_{k+\frac{N}{2}} = E_k \text{ and } O_{k+\frac{N}{2}} = O_k.$$

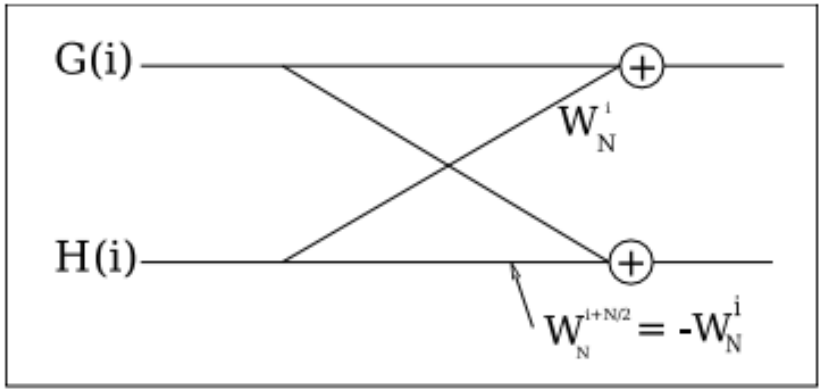
Finally,

$$X_k = E_k + e^{-\frac{2\pi i}{N}k} O_k$$

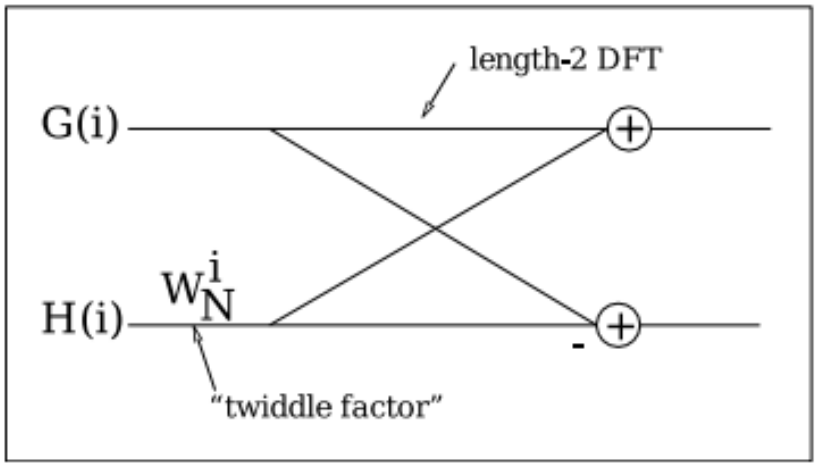
$$X_{k+\frac{N}{2}} = E_k - e^{-\frac{2\pi i}{N}k} O_k$$


x[0]
x[2]
x[4]
x[6]
N/2- point DFT
x[1]
x[3]
x[5]
x[7]
N/2- point DFT
G[0]
G[1]
G[2]
G[3]
H[0]
H[1]
H[2]
H[3]
X[0]
X[1]
X[2]
X[3]
X[4]
X[5]
X[6]
X[7]

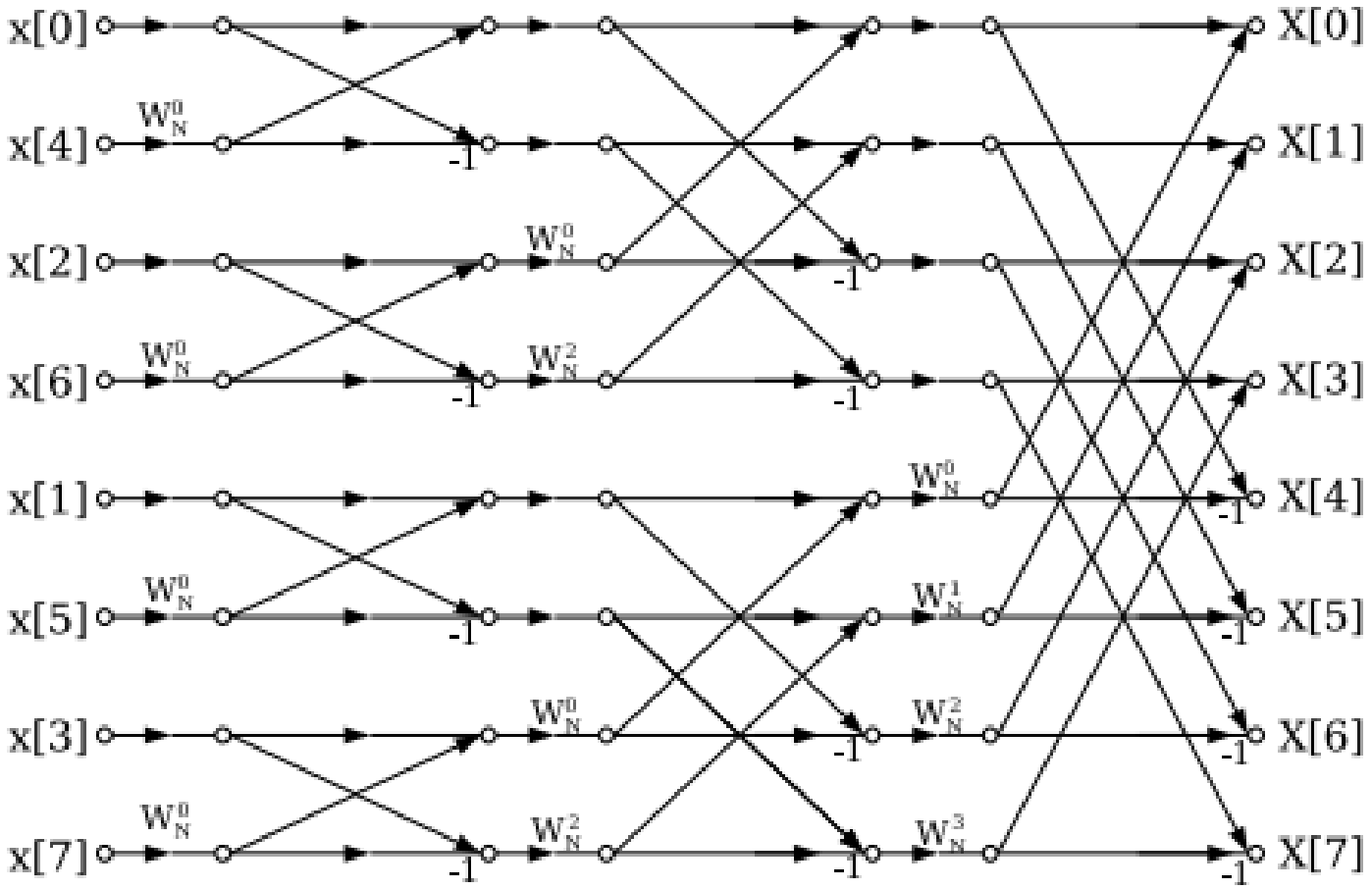
Further reduction in computation



(a)



Complete Structure



O(N log(N))

Basic Theory

- To implement Hilbert transform in Verilog
  - Find FFT of sequence
  - Multiply it by –j sgn(k)
  - Take IFFT of the resulting sequence

$$\hat{x}(n) = \mathrm{IFFT}(-j\,\mathrm{sgn}(k)X(k))$$

Where $X(k) = \mathrm{FFT}(x(n))$ and

$$-j\,\mathrm{sgn}(k) = \begin{cases} -j & k = 1,2,\ldots,N/2-1 \\ 0 & k = 0,\ N/2 \\ +j & k = N/2+1,\ldots,N-1 \end{cases}$$

Note

- The approach mentioned above is one of the many possible solutions to the problem statement.
- You should search more and look for better solutions.

Judging criteria

- Area (#LUTs, #FFs, #BRAMs, #DSP Elements etc.)
- Latency (No. of cycles Reqd).
- Need for external memory (eg. DRAM Controller – DDR/DDR2 and size of memory required).
- Maximum frequency achieved.
- Power consumption as reported by ISE tool.
- Extra Feature : Implementation of a feature where Hilbert transform is being used