FAST FOURIER TRANSFORM

Fast Fourier Transform, or FFT, is an algorithm for computing the N point DFT with a computational complexity of O(nlogn).

It is an efficient way of calculating the DFT of x(n)

•••

Fast Fourier Transform:

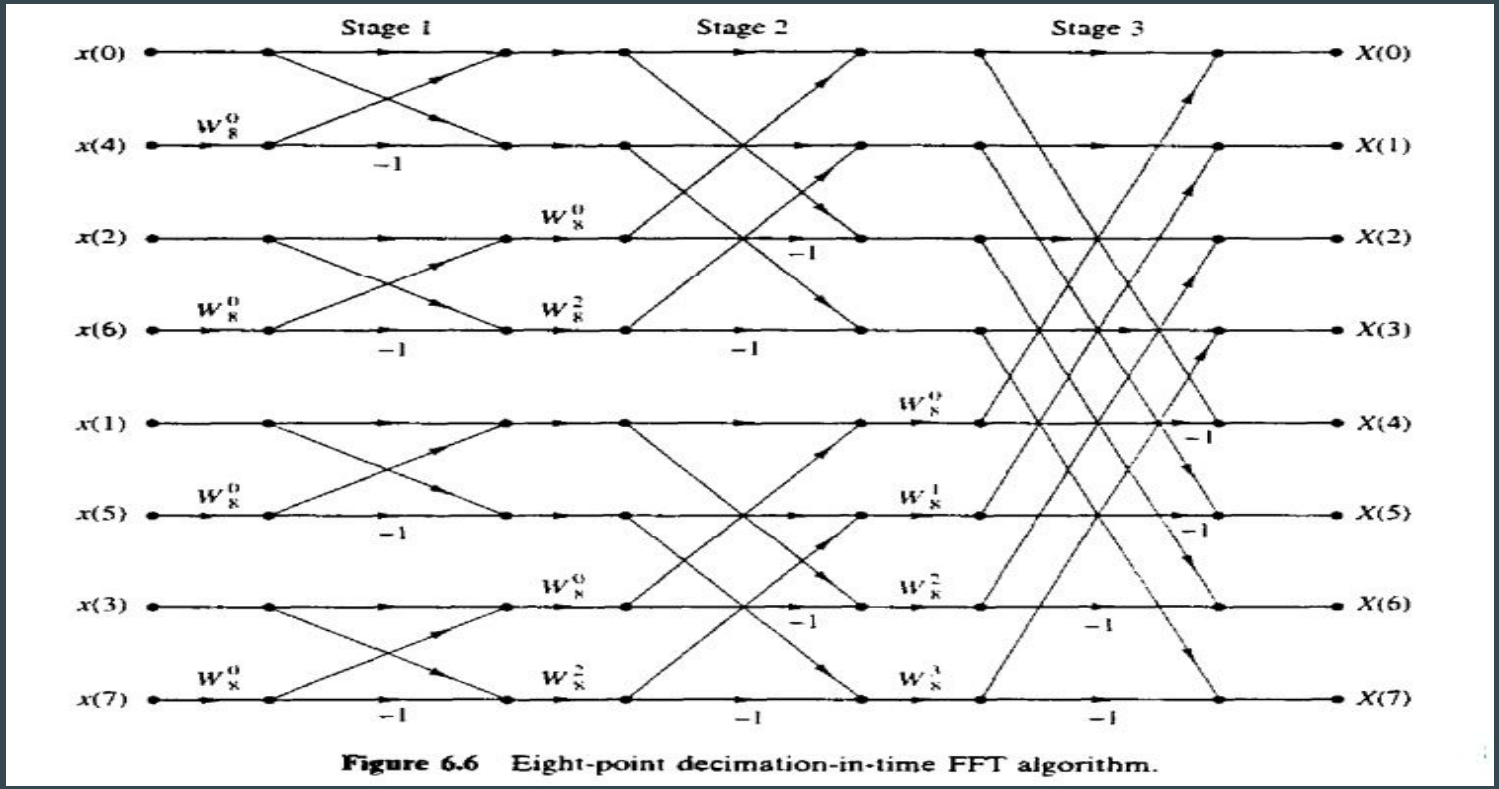


Figure 6.6 Eight-point decimation-in-time FFT algorithm.

Matlab FFT Code without pipelining:

In this code N point fft is implemented without pipelining i.e it is implemented without any synchronisation with the clock as soon as the input is given we get the output as that of combinational block.

Total elapsed time is 0.0026 sec for without pipelined processor for an 8 point fft.

MATLAB R2016b - academic use

HOME | PLOTS | APPS | SHORTCUTS | EDITOR | PUBLISH | VIEW

Search Documentation

/ ▸ home ▸ ayush ▸ Desktop ▸ 201530241_assign2_dip ▸ problem_2 ▸ problem_a

Command Window

```
>> fftsp([1 2 3 4 5 6 7 8])
Elapsed time is 0.000153 seconds.
Elapsed time is 0.000040 seconds.
Elapsed time is 0.001653 seconds.
Elapsed time is 0.000035 seconds.
Elapsed time is 0.000027 seconds.
Elapsed time is 0.000236 seconds.
Elapsed time is 0.004178 seconds.

ans =

  36.0000 + 0.0000i
  -4.0000 + 9.6569i
  -4.0000 + 4.0000i
  -4.0000 + 1.6569i
  -4.0000 + 0.0000i
  -4.0000 - 1.6569i
  -4.0000 - 4.0000i
  -4.0000 - 9.6569i

>> 0.000153+0.000040+0.001653+0.000035+0.00

ans =

    0.0026

fx >>
```

Editor - /home/ayush/Desktop/201530241_assign2_dip/problem_2/problem_a/fftsp.m

fft.m | fftsp.m

```
1  function [X] = fftsp(x)
2  tic;
3  N = size(x,2);
4  if N>=4
5      Even = x([1:2:end]);
6      Odd = x([2:2:end]);
7      Evenfft = fftsp(Even);
8      Oddfft = fftsp(Odd);
9      Multiply = exp(((-1i*2*pi)/N)*(0:N/2-1)');
10     X = [Evenfft + Multiply.*Oddfft;Evenfft - Multiply.*Oddfft];
11 else
12     X = [1 1;1 -1]*x';
13 end
14 toc;
15 end
```

4 usages of "N" found

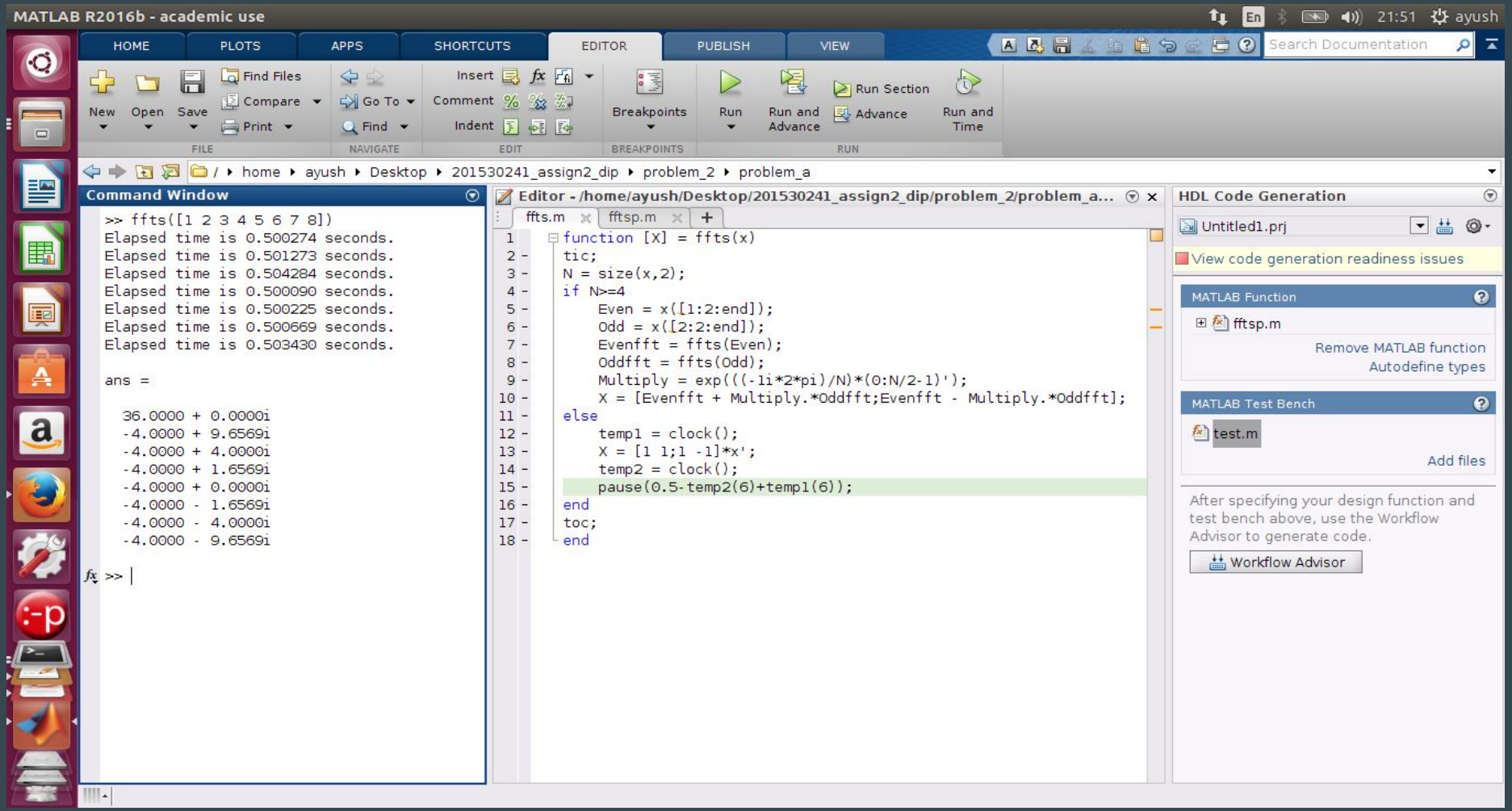
fftsp

Ln 3 Col 1

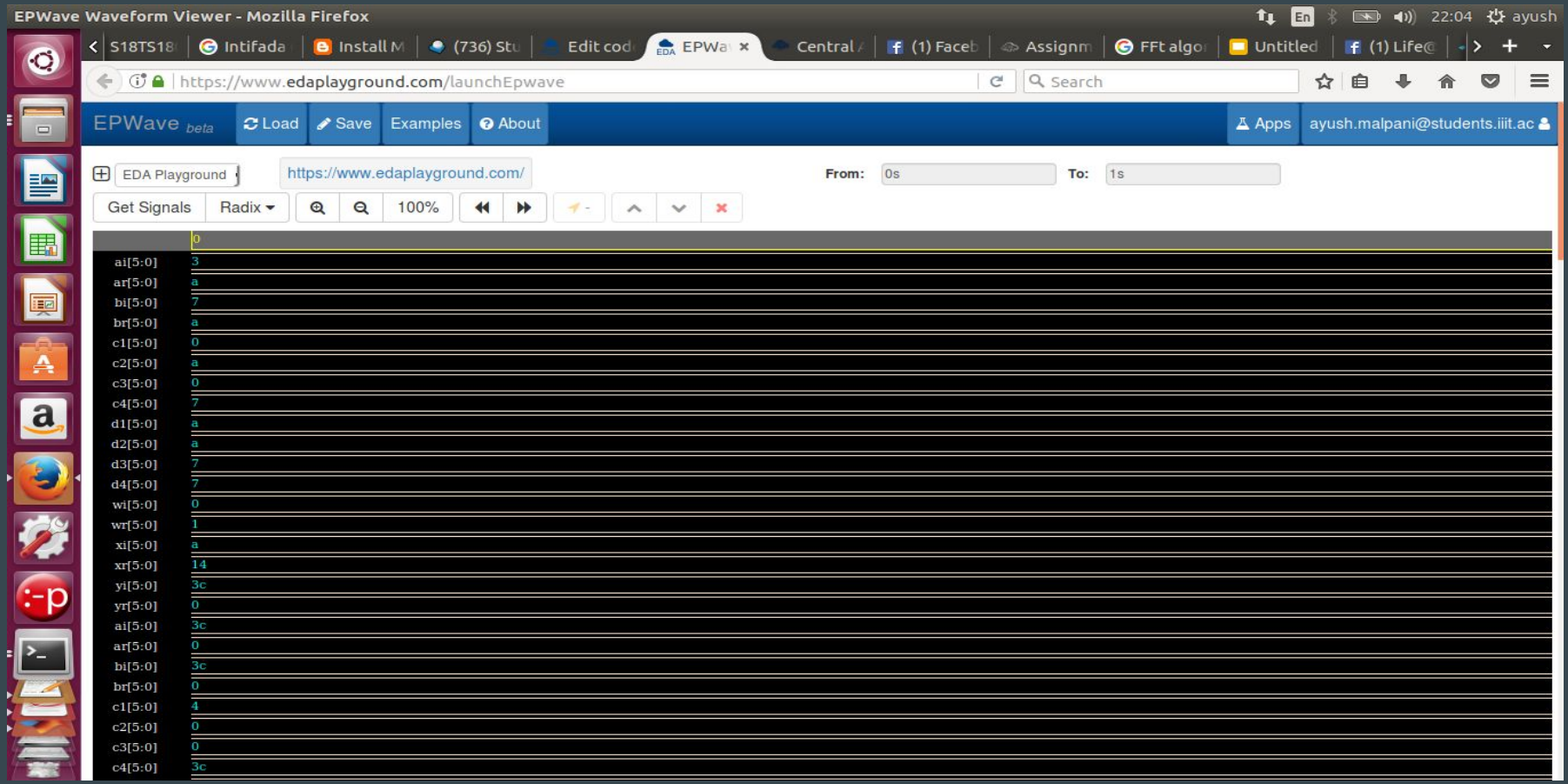
Matlab Code with pipelining:

In this code N point fft is calculated with pipelining i.e implemented each stage in synchronisation with a clock but as clock cannot be generated externally I have used pause function in matlab for clock synchronisation each stage is set to 0.5 sec as the clock rate. It calculates each stage value in some p sec and then pause for 0.5 - p sec and then gives its value to the next stage.

By using this latency increases little bit but throughput of the function increases hence the efficiency increases.


MATLAB R2016b - academic use
21:51
ayush
HOME
PLOTS
APPS
SHORTCUTS
EDITOR
PUBLISH
VIEW
Search Documentation
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
/ ▸ home ▸ ayush ▸ Desktop ▸ 201530241_assign2_dip ▸ problem_2 ▸ problem_a
Command Window
>> ffts([1 2 3 4 5 6 7 8])
Elapsed time is 0.500274 seconds.
Elapsed time is 0.501273 seconds.
Elapsed time is 0.504284 seconds.
Elapsed time is 0.500090 seconds.
Elapsed time is 0.500225 seconds.
Elapsed time is 0.500669 seconds.
Elapsed time is 0.503430 seconds.
ans =
36.0000 + 0.0000i
-4.0000 + 9.6569i
-4.0000 + 4.0000i
-4.0000 + 1.6569i
-4.0000 + 0.0000i
-4.0000 - 1.6569i
-4.0000 - 4.0000i
-4.0000 - 9.6569i
>>
Editor - /home/ayush/Desktop/201530241_assign2_dip/problem_2/problem_a...
ffts.m
fftsp.m
function [X] = ffts(x)
tic;
N = size(x,2);
if N>=4
Even = x([1:2:end]);
Odd = x([2:2:end]);
Evenfft = ffts(Even);
Oddfft = ffts(Odd);
Multiply = exp(((-1i*2*pi)/N)*(0:N/2-1)');
X = [Evenfft + Multiply.*Oddfft;Evenfft - Multiply.*Oddfft];
else
temp1 = clock();
X = [1 1;1 -1]*x';
temp2 = clock();
pause(0.5-temp2(6)+temp1(6));
end
toc;
end
HDL Code Generation
Untitled1.prj
View code generation readiness issues
MATLAB Function
fftsp.m
Remove MATLAB function
Autodefine types
MATLAB Test Bench
test.m
Add files
After specifying your design function and test bench above, use the Workflow Advisor to generate code.
Workflow Advisor

WAVEFORM:


EPWave Waveform Viewer - Mozilla Firefox
https://www.edaplayground.com/launchEpwave
EPWave beta
Load
Save
Examples
About
Apps
ayush.malpani@students.iiit.ac
EDA Playground
https://www.edaplayground.com/
From:
0s
To:
1s
Get Signals
Radix
100%
ai[5:0] 3
ar[5:0] a
bi[5:0] 7
br[5:0] a
c1[5:0] 0
c2[5:0] a
c3[5:0] 0
c4[5:0] 7
d1[5:0] a
d2[5:0] a
d3[5:0] 7
d4[5:0] 7
wi[5:0] 0
wr[5:0] 1
xi[5:0] a
xr[5:0] 14
yi[5:0] 3c
yr[5:0] 0
ai[5:0] 3c
ar[5:0] 0
bi[5:0] 3c
br[5:0] 0
c1[5:0] 4
c2[5:0] 0
c3[5:0] 0
c4[5:0] 3c

FFT Using Verilog Without pipelining:

In this 8 point fft is implemented using verilog and without pipelining i.e all blocks are considered as asynchronous combinational blocks as soon one block gets its output it sends its output to the next block.

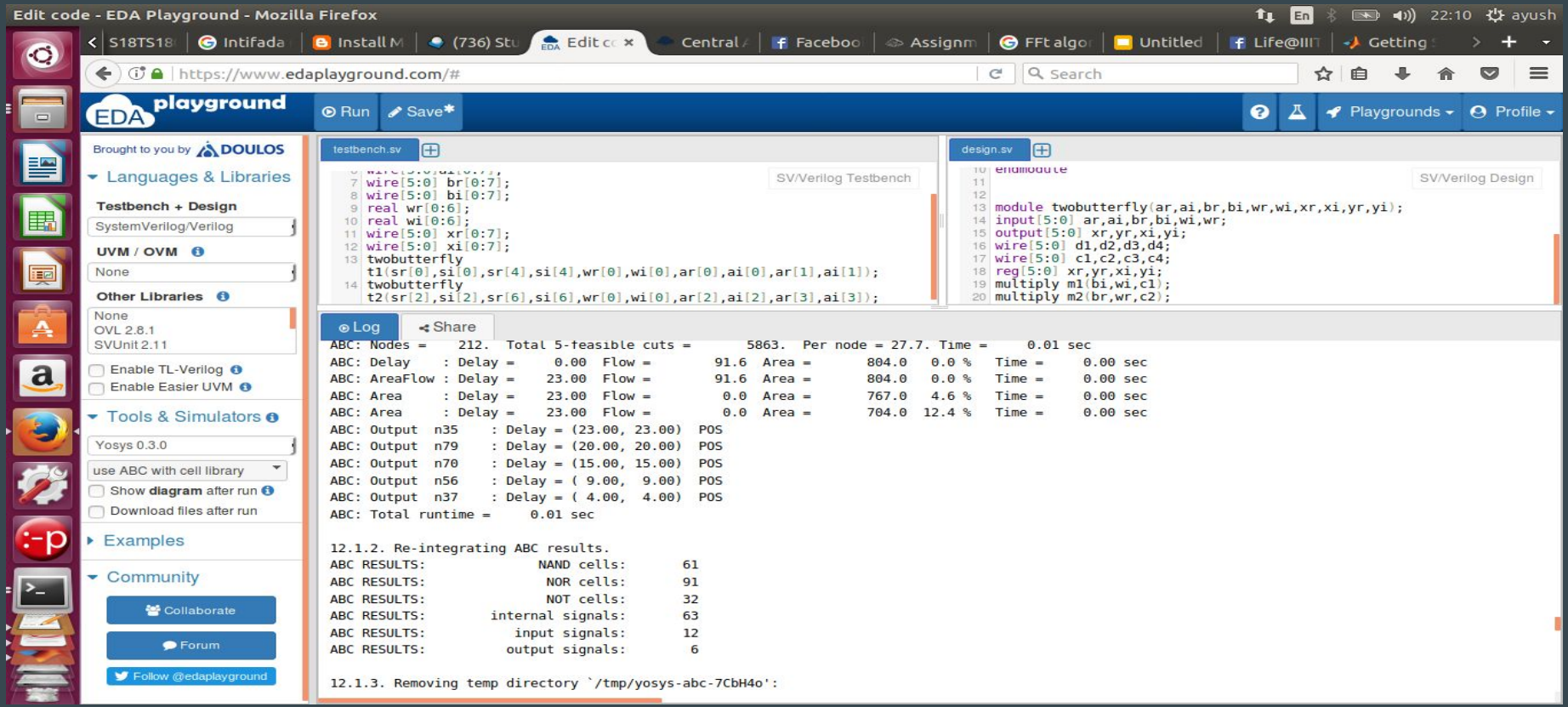
I have used online EDA - tool for synthesis and simulation of code.

FFT Using Verilog With pipelining:

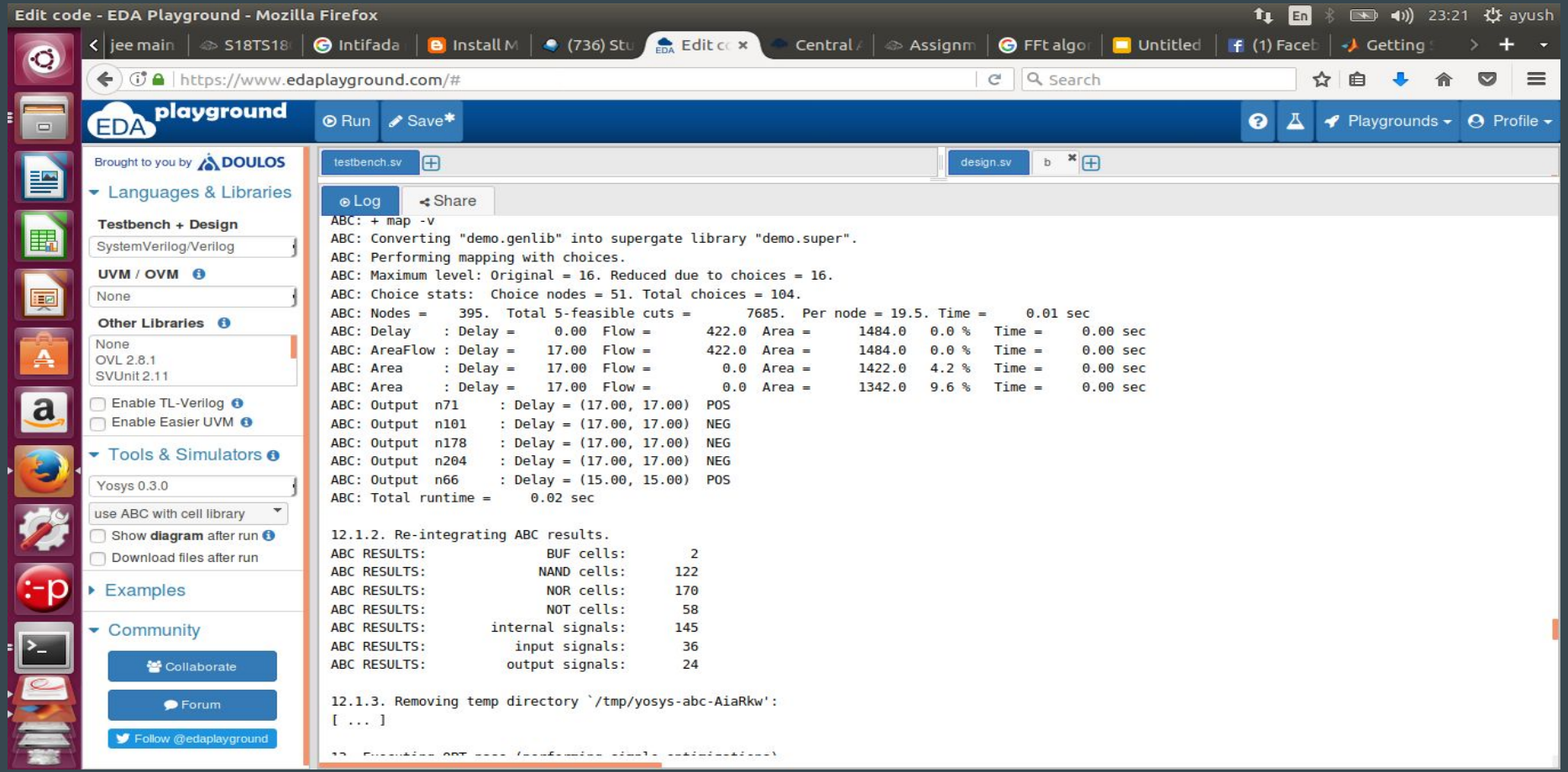
In this 8 point fft is implemented using verilog and with pipelining i.e all blocks are considered as synchronous blocks which gives its value to the next stage on rising edge of the clock.

However latency of pipelined processor increases slightly throughput increases.

WITHOUT PIPELINING SYNTHESIS REPORT:



WITH PIPELINING SYNTHESIS REPORT:



COMPARISON BETWEEN PIPELINED AND WITHOUT:

	PIPELINED	WITHOUT PIPELINE
Area	1400	890
Throughput	100 instructions per sec(0.01 sec for one ins)	300 instructions per second (0.01/3 sec for one ins)
Latency	Around 0.01 sec	Slightly greater than 0.01 sec
Cost	More for pipelined as number of gates and area is more	Less as number of gates is less and area reqd. Is less
Gates reqd	120 nand cells as d-flip flops are required more number of gates	70 nand cells