Parallel Programming for FPGAs (part 2) Practice

Lab1: Discrete Fourier Transform (*DFT*)

- The Discrete Fourier Transform (*DFT*) is a numerical variant of the Fourier Transform.
- The *DFT* transforms a sequence of N complex numbers $\{g_k\} \coloneqq g_0,\ g_1,\ \ldots,\ g_{N-1}$ into another sequence of complex numbers $\{G_n\} \coloneqq G_0, G_1, \ldots, G_{N-1}$, which defined by

$$G_n = \sum_{k=0}^{N-1} g_k \cdot e^{-\frac{i2\pi}{N}kn} = \sum_{k=0}^{N-1} g_k \cdot s^{kn}$$

, where $s = \exp(-\frac{i2\pi}{N})$

Discrete Fourier Transform (2)

$$G_n = \sum_{k=0}^{N-1} g_k \cdot s^{kn}$$

$G = Sg$, where

$$S = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & s & s^2 & \cdots & s^{(N-1)} \\ 1 & s^2 & s^4 & \cdots & s^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & s^{(N-1)} & s^{2(N-1)} & \cdots & s^{(N-1)(N-1)} \end{bmatrix}$$

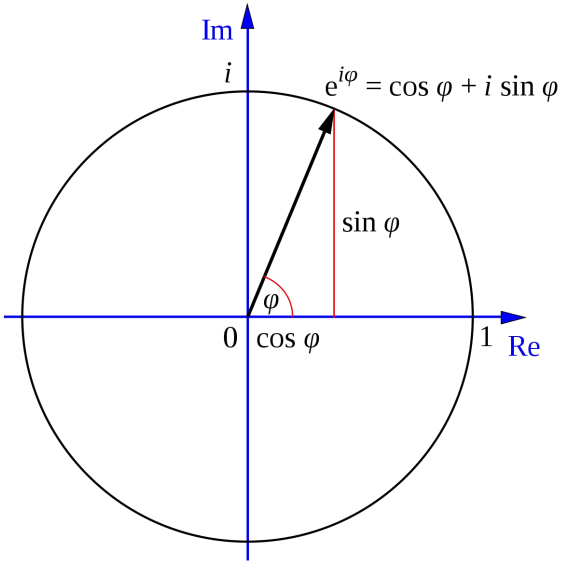
Matrix vector multiplication

How to calculate s^k ($s = \exp(-\frac{i2\pi}{N})$)?

Discrete Fourier Transform (3)

- Euler's Formula: $e^{ix} = \cos x + i \sin x$

$$s^k = \exp\left(-\frac{i2\pi}{N} \cdot k\right)$$
$$= \cos\left(-\frac{2\pi}{N} \cdot k\right) + i\sin\left(-\frac{2\pi}{N} \cdot k\right)$$



$$s^k = s^{k+jN},\ j \in N$$
$$s^k = s^{k \bmod N}$$

Summary

Matrix-vector multiplication: $G = Sg$, where

$$S = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & s & s^2 & \cdots & s^{(N-1)} \\ 1 & s^2 & s^4 & \cdots & s^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & s^{(N-1)} & s^{2(N-1)} & \cdots & s^{(N-1)(N-1)} \end{bmatrix}$$

$$s^k = \exp\left(-\frac{i2\pi}{N} \cdot k\right)$$

$$= \cos\left(-\frac{2\pi}{N} \cdot k\right) + i\sin\left(-\frac{2\pi}{N} \cdot k\right)$$

$$(a + bi) * (c + di) = (ac - bd) + (bc + ad)i$$

Source files

- *dft.h*: typedef, macro, kernel declaration
- *dft.cpp*: HLS kernel definition
- *dft_tb.cpp*: test bench for C-Simulation
- *out.gold.dat*: standard results for input data
- *coefficients1024.h*: $N = 1024$
  - cos_coefficients_table[k]= $\cos\left(-\frac{2\pi}{N}\cdot k\right)$
  - sin_coefficients_table[k]= $\sin\left(-\frac{2\pi}{N}\cdot k\right)$

$$
\begin{aligned}
s^k &= \exp\left(-\frac{2\pi k i}{N}\right) \\
&= \cos\left(-\frac{2\pi}{N}\cdot k\right) + i\sin\left(-\frac{2\pi}{N}\cdot k\right)
\end{aligned}
$$

Create baseline

- Create and configure the *HLS* project
- Create a baseline design
  - You may follow the template below.

```
void dft(DTYPE real_in[N], DTYPE imag_in[N],
         DTYPE real_out[N],DTYPE imag_out[N])
{
// 1. interface pragma, use m_axi protocol
// 2. define local buffers, and load inputs to them.

// 3. matrix-vector multiplication loop
     for (int i=0; i<N; i++) {
     // ...
          for (int j=0; j<N; j++) {
          // 3.1. load s[kn] from [cos/sin]_coefficients_table
          // 3.2. calculate complex multiply-accumulate
          }
     }
// 4. store answers through output interfaces

}
```

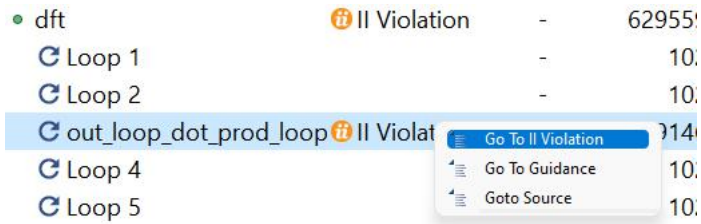
run C-Simulation
run C-Synthesis

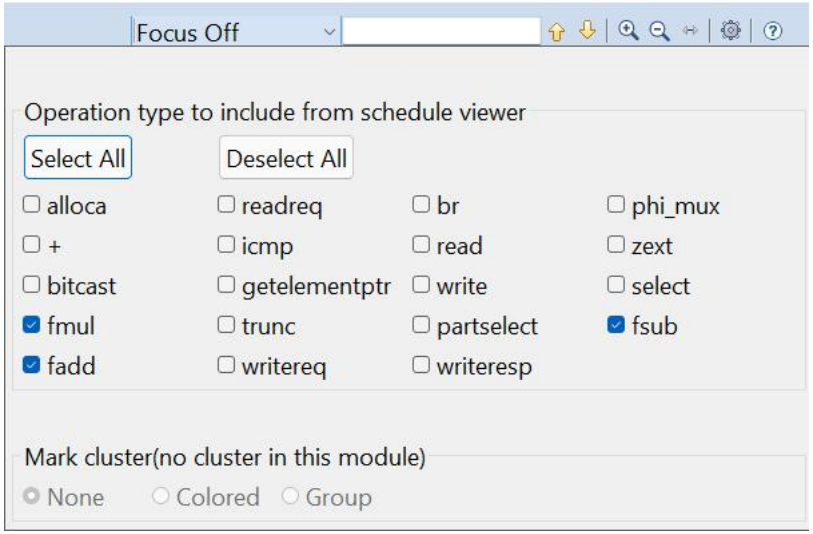
Create baseline (2)

```
DTYPE real_buf[N], imag_buf[N];
DTYPE real_ans[N], imag_ans[N];
memcpy(real_buf, real_in, N * sizeof(DTYPE));
memcpy(imag_buf, imag_in, N * sizeof(DTYPE));

for (int i=0; i<N; i++) {
  real_ans[i] = imag_ans[i] = 0.;
  int exp = 0;
  for (int j=0; j<N; j++) {
    float real = cos_coefficients_table[exp];
    float imag = sin_coefficients_table[exp];
    real_ans[i] += real * real_buf[j] - imag * imag_buf[j];
    imag_ans[i] += real * imag_buf[j] + imag * real_buf[j];
    exp += i;
    if (exp >= N) exp -= N;
  }
}
memcpy(real_out, real_ans, N * sizeof(DTYPE));
memcpy(imag_out, imag_ans, N * sizeof(DTYPE));
```

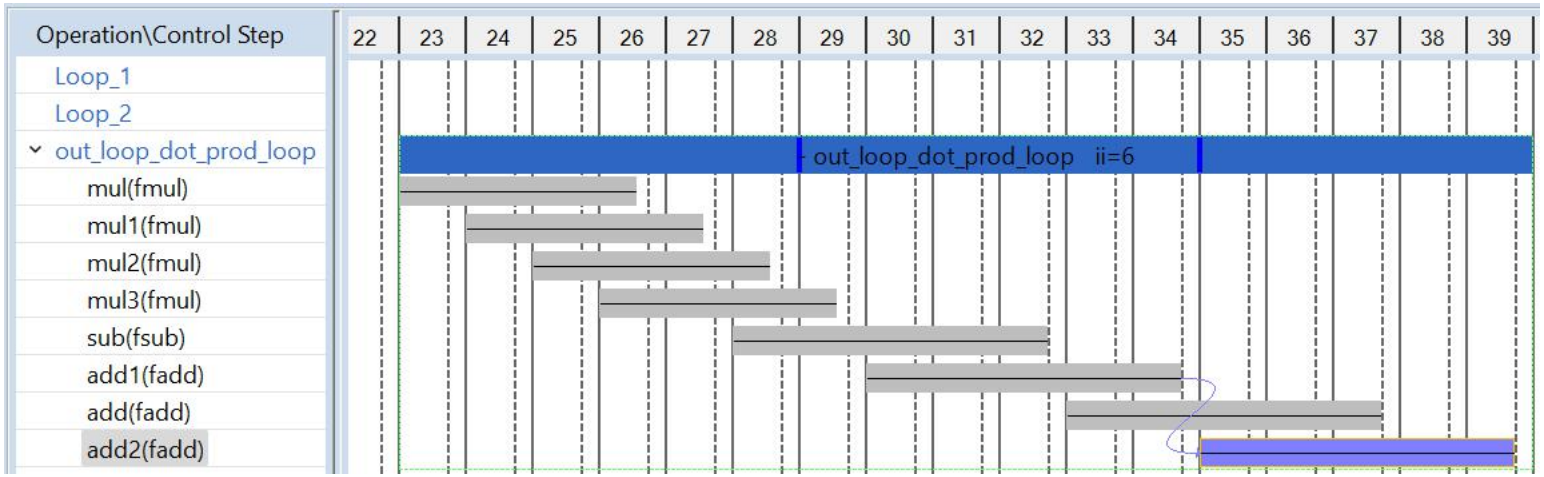
Baseline analysis

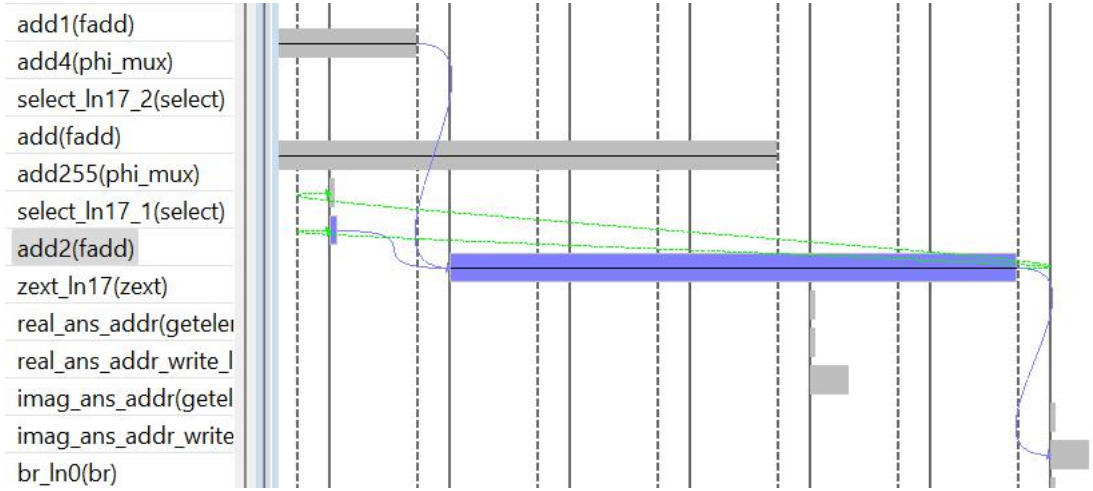
Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft	II Violation	-	6295599	6.296E7	-	6295600	-	no	20	5	3605	4243	0
Loop 1		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
Loop 2		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
out_loop_dot_prod_loop	II Violation	-	6291469	6.292E7	20	6	1048576	yes	-	-	-	-	-
Loop 4		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
Loop 5		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-





Baseline analysis (2)

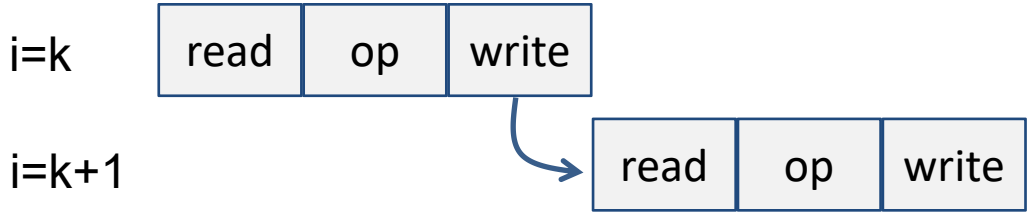




Baseline analysis (3)

```
for (int j=0; j<N; j++) {
  float real = cos_coefficients_table[exp];
  float imag = sin_coefficients_table[exp];
  real_ans[i] += real * real_buf[j] - imag * imag_buf[j];
  imag_ans[i] += real * imag_buf[j] + imag * real_buf[j];
  exp += i;
  if (exp >= N) exp -= N;
}
```

```
for (i) {
  x = x + a;
}
```

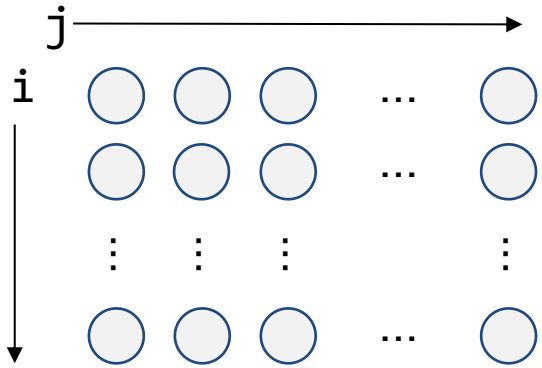


RAW (read-after-write)

$II \geq$ duration(read + op),
where duration(write) can be omitted by forwarding

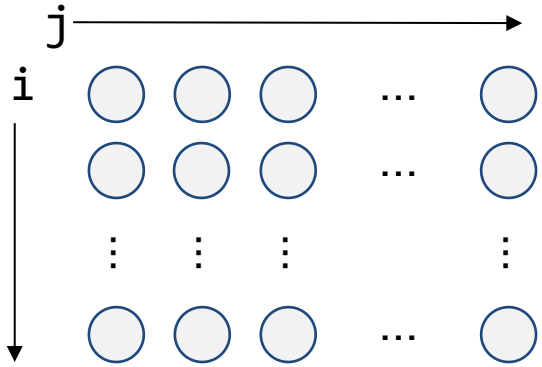
Remove *RAW*

```
for (int i=0; i<N; i++) {
..
    for (int j=0; j<N; j++) {
        ..
        real_ans[i] += ..;
        imag_ans[i] += ..;
        ..
    }
}
```




```
for (int j=0; i<N; i++) {
..
    for (int i=0; j<N; j++) {
        ..
        real_ans[i] += ..;
        imag_ans[i] += ..;
        ..
    }
}
```

loop exchange



Apply the optimization. Get design ***dft-opt***.

dft-opt analysis

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
dft		-	1054774	1.055E7	-	1054775	-	no	20	17	4355	5983	0
Loop 1		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
Loop 2		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
Loop 3		-	1024	1.024E4	1	1	1024	yes	-	-	-	-	-
Loop 4		-	1024	1.024E4	1	1	1024	yes	-	-	-	-	-
VITIS_LOOP_33_1_VITIS_LOOP_37_2		-	1048592	1.049E7	18	1	1048576	yes	-	-	-	-	-
Loop 6		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
Loop 7		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-

Fully pipelining

Lab2: Fast Fourier Transform(FFT)

- Fast Fourier Transform (*FFT*)
  - DFT implemented by matrix-vector multiplication requires $O(N^2)$ multiplications
  - FFT exploits the <u>structure of the matrix D</u> to reduce the complexity

$$s^{k+\frac{N}{2}} = -s^k$$

$$\begin{aligned} s^k &= \exp\left(-\frac{i2\pi}{N}\cdot k\right) \\ &= \cos\left(-\frac{2\pi}{N}\cdot k\right) + i\sin\left(-\frac{2\pi}{N}\cdot k\right) \end{aligned}$$

- Cooley-Tukey algorithm
  - A divide-and-conquer approach

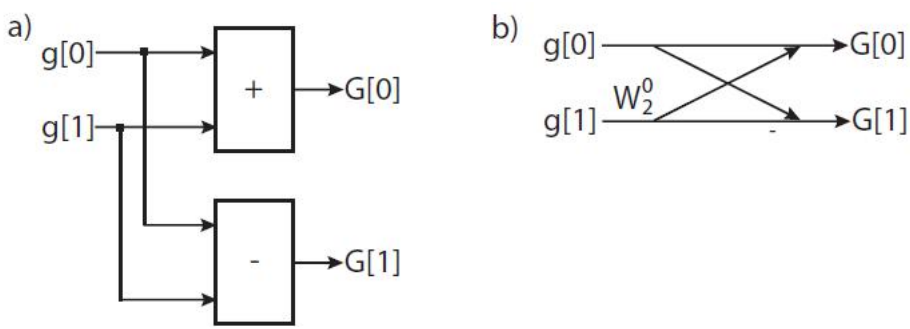
Fast Fourier Transform (2)

- Recall DFT: $X = Dx$
  - D is the DFT coefficients
- For a 2-point DFT

$$S = \begin{bmatrix} W_2^{00} & W_2^{01} \\ W_2^{10} & W_2^{11} \end{bmatrix}, \text{ where } W = e^{-j2\pi},\ \ W_4^{23} = e^{\frac{-j2\pi\cdot 2\cdot 3}{4}}$$

$$G[0] = g[0]\cdot e^{\frac{-j2\pi\cdot 0\cdot 0}{2}} + g[1]\cdot e^{\frac{-j2\pi\cdot 0\cdot 1}{2}} = g[0] + g[1]$$

$$G[1] = g[0]\cdot e^{\frac{-j2\pi\cdot 1\cdot 0}{2}} + g[1]\cdot e^{\frac{-j2\pi\cdot 1\cdot 1}{2}} = g[0] - g[1]$$

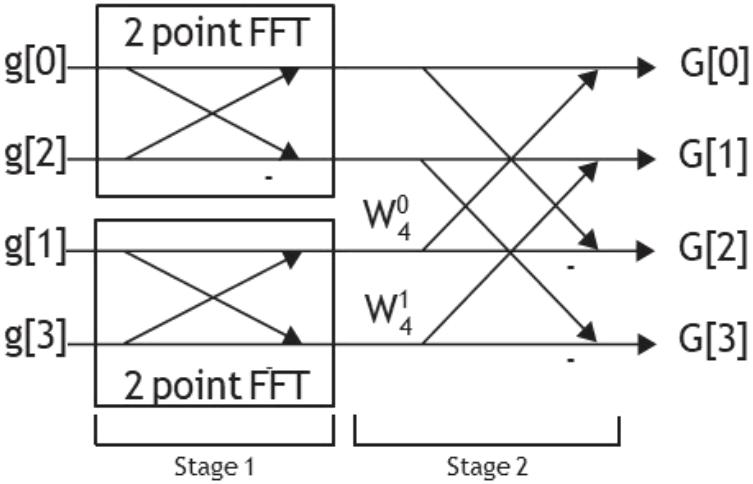


Fast Fourier Transform (3)

- For a 4-point DFT

$$G[0] = (g[0] + g[2]) + e^{\frac{-j2\pi\cdot 0}{4}}(g[1] + g[3])$$
$$G[1] = (g[0] - g[2]) + e^{\frac{-j2\pi\cdot 1}{4}}(g[1] - g[3])$$
$$G[2] = (g[0] + g[2]) + e^{\frac{-j2\pi\cdot 2}{4}}(g[1] + g[3])$$
$$G[3] = (g[0] - g[2]) + e^{\frac{-j2\pi\cdot 3}{4}}(g[1] - g[3])$$

$$G[0] = (g[0] + g[2]) + e^{\frac{-j2\pi\cdot 0}{4}}(g[1] + g[3])$$
$$G[1] = (g[0] - g[2]) + e^{\frac{-j2\pi\cdot 1}{4}}(g[1] - g[3])$$
$$G[2] = (g[0] + g[2]) - e^{\frac{-j2\pi\cdot 0}{4}}(g[1] + g[3])$$
$$G[3] = (g[0] - g[2]) - e^{\frac{-j2\pi\cdot 1}{4}}(g[1] - g[3])$$



Fast Fourier Transform (4)

$$G[k] = \sum_{n=0}^{N-1} g[n] \cdot e^{\frac{-j2\pi kn}{N}}, \text{ for } k = 0, \ \ldots, \ N-1$$

$$G[k] = \sum_{n=0}^{\frac{N}{2}-1} g[2n] \cdot e^{\frac{-j2\pi k(2n)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} g[2n+1] \cdot e^{\frac{-j2\pi k(2n+1)}{N}}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} g[2n] \cdot e^{\frac{-j2\pi kn}{N/2}} + e^{\frac{-j2\pi k}{N}} \sum_{n=0}^{\frac{N}{2}-1} g[2n+1] \cdot e^{\frac{-j2\pi kn}{N/2}}$$

a) $\mathrm{G}[k] = A_k + W_N^k B_k$

where A is the DFT result of $\{g_{2n}\} \coloneqq g_0, \ g_2, \ \ldots, \ g_{N-2}$,
and B is the DFT result of $\{g_{2n+1}\} \coloneqq g_1, \ g_3, \ \ldots, \ g_{N-1}$

Fast Fourier Transform (5)

What is $G[k+N/2]$ for $k=0,\ \ldots, N/2-1$?

$$G[k+N/2] = \sum_{n=0}^{\frac{N}{2}-1} g[2n]\cdot e^{\frac{-j2\pi\left(k+\frac{N}{2}\right)(2n)}{N}} + \sum_{n=0}^{\frac{N}{2}-1} g[2n+1]\cdot e^{\frac{-j2\pi\left(k+\frac{N}{2}\right)(2n+1)}{N}}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} g[2n]\cdot e^{\frac{-j2\pi k(2n)}{N}}\cdot e^{-j2n\pi} + \sum_{n=0}^{\frac{N}{2}-1} g[2n+1]\cdot e^{\frac{-j2\pi k(2n+1)}{N}}\cdot e^{-j(2n+1)\pi}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} g[2n]\cdot e^{\frac{-j2\pi kn}{N/2}} - e^{\frac{-j2\pi k}{N}} \sum_{n=0}^{\frac{N}{2}-1} g[2n+1]\cdot e^{\frac{-j2\pi kn}{N/2}}$$

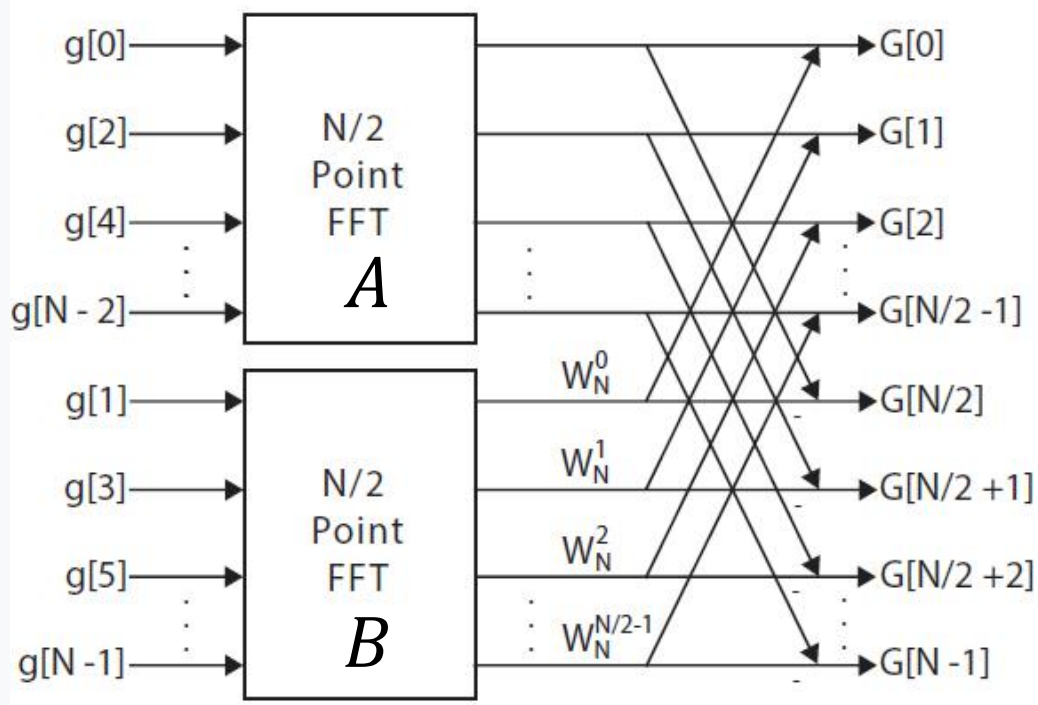
b) $\mathrm{G}[k+N/2] = A_k - W_n^k\, B_k$

Fast Fourier Transform (6)

a) $\mathrm{G}[k] = A_k + W_N^k B_k$

b) $\mathrm{G}[k + N/2] = A_k - W_n^k B_k$

```
G_{0,...,N−1} ← fft_recur(g, N, s):
// DFT of (g_0, g_s, g_2s, ... g_(N-1)s)
 if N = 1 then G_0 ← g_0
 else
  G_{0,...,N/2−1} ← fft_recur(g, N/2, 2s)
  G_{N/2,...,N−1} ← fft_recur(g+s, N/2, 2s)
  for k = 0 to N/2−1 do
   p ← G_k
   q ← exp(−2πik/N) G_{k+N/2}
   G_k ← p + q
   G_{k+N/2} ← p − q
  end for
 end if
```



Fast Fourier Transform (7)

- Bit reverse: prepare input data layout for *divide-and-conquer*

Index	Binary	Reversed Binary	Reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

$(g_0, g_1, \ldots, g_{N-1})$ are divided into: $(g_0, g_2, \ldots, g_{N-2})$ and $(g_1, g_3, \ldots, g_{N-1})$.

If the lowest bit of g_n's original index ($n\&1$) is 0, g_k will be divided to the first sequence, otherwise the second sequence.

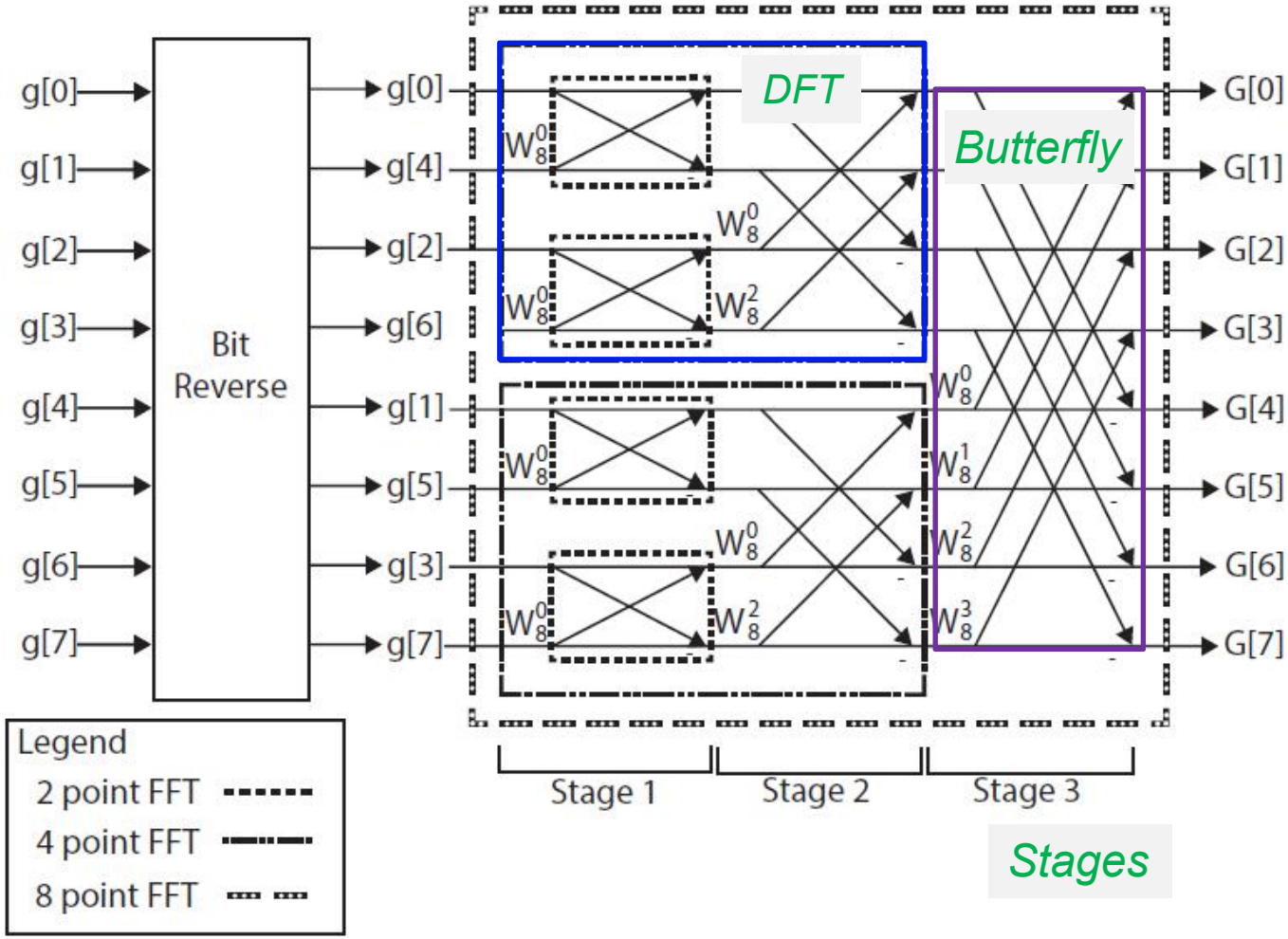
So, g_k's new index should have ($n\&1$) as its highest bit.

Repeat the procedure, **g_n's new index will be the bit-reverse (*rev*) of its original index.**

```
rev_index ← rev (index):
  k = 1
  rev_index = 0
  while index > 0 do:
    rev_index |= (index&1) << k
    index = index >> 1
    k +=1
  end while
```

Fast Fourier Transform (8)

- For $N = 8$,



Fast Fourier Transform (9)

- Iterative implementation

```
algorithm bit-reverse-copy(a,A) is
input: Array a of n=2^log2(n) complex values.
output: Array A of size n.
  n ← a.length
  for k = 0 to n – 1 do
    A[rev(k)] := a[k]
```

```
algorithm iterative-fft is
  input: Array a of n=2^log2(n) complex values.
  output: Array A the DFT of a.
  bit-reverse-copy(a, A)
  n ← a.length
  for s = 1 to log2(n) do
    m ← 2^s
    ω_m ← exp(−2πi/m)
    for k = 0 to n-1 by m do
      ω ← 1
      for j = 0 to m/2 – 1 do
        t ← ω A[k + j + m/2]
        u ← A[k + j]
        A[k + j] ← u + t
        A[k + j + m/2] ← u – t
        ω ← ω · ω_m
  return A
```

stage_loop

dft_loop

butterfly_ loop

Summary

- Iterative implementation of the *divide-and-conquer* algorithm

```
algorithm bit-reverse-copy(a,A) is
input: Array a of n=2^log2(n) complex values.
output: Array A of size n.
  n ← a.length
  for k = 0 to n – 1 do
    A[rev(k)] := a[k]
```

```
algorithm iterative-fft is
  input: Array a of n=2^log2(n) complex values.
  output: Array A the DFT of a.
  bit-reverse-copy(a, A)
  n ← a.length
  for s = 1 to log2(n) do
    m ← 2^s
    ω_m ← exp(−2πi/m)
    for k = 0 to n-1 by m do
      ω ← 1
      for j = 0 to m/2 – 1 do
        t ← ω A[k + j + m/2]
        u ← A[k + j]
        A[k + j] ← u + t
        A[k + j + m/2] ← u – t
        ω ← ω · ω_m
  return A
```

stage_loop

dft_loop

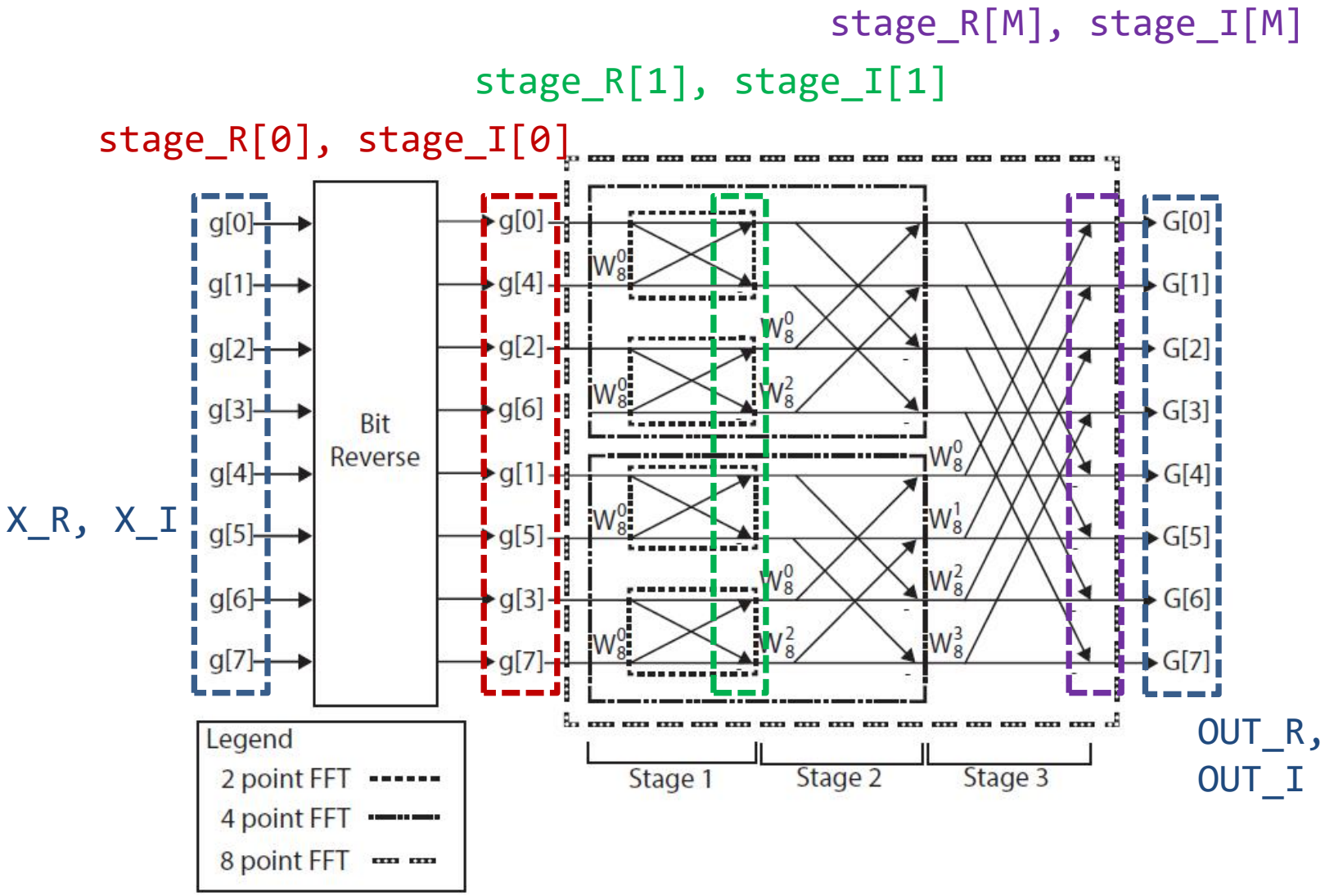
butterfly_loop

Create baseline

- Create and configure the *HLS* project
- Create a baseline design

```
void fft(DTYPE X_R[N], DTYPE X_I[N], DTYPE OUT_R[N], DTYPE OUT_I[N])
{

// interface pragmas, use m_axi protocol

  DTYPE stage_R[M+1][N], stage_I[M+1][N];

  bit_reverse(X_R, X_I, stage_R[0], stage_I[0]);

  for (int stage=0; stage<M; stage++) {
    fft_stage(stage,        stage_R[stage], stage_I[stage],
                            stage_R[stage+1], stage_I[stage+1]);
  }
  store_results(stage_R[M], OUT_R);
  store_results(stage_I[M], OUT_I);
}
```

Create baseline (2)



Create baseline (3)

```
void bit_reverse(DTYPE X_R[N], DTYPE X_I[N],
        DTYPE OUT_R[N], DTYPE OUT_I[N]) {
// Insert your code here
}
```

algorithm bit-reverse-copy(*a*,*A*) **is**
input: Array *a* of $N=2^{log2(N)}$ complex values.
output: Array *A* of size *N*.
  N ← *a*.length
  for *i* = 0 *to* *N* – 1 **do**
    A[*rev*(i)] := *a*[i]
  end for

how to implement *rev*(k)?

algorithm rev(*i*) **is**
  ret := 0
  for k = 0 to log2(N) **do**
    rev := (rev << 1) | ((i>>k) & 1)
  end for
return ret

```
void bit_reverse(DTYPE X_R[N], DTYPE X_I[N], DTYPE OUT_R[N], DTYPE OUT_I[N]) {
    for (int i=0; i<N; i++) {
        int rev = 0;
        for (int k=0; k<M; k++) {
            rev = (rev << 1) | ((i >> k) & 1);
        }
        OUT_R[rev] = X_R[i];
        OUT_I[rev] = X_I[i];
    }
}
```

Create baseline (4)

```
void fft_stage(int stage, DTYPE X_R[N], DTYPE X_I[N], DTYPE OUT_R[N],
               DTYPE OUT_I[N]) {
// Insert your code here
}
```

$m \leftarrow 2^s$
$\omega_m \leftarrow \exp(-2\pi i/m)$
for $k = 0$ **to** n-1 **by** m **do**
  $\omega \leftarrow 1$
  for $j = 0$ **to** $m/2 - 1$ **do**
    $t \leftarrow \omega\, A[k + j + m/2]$
    $u \leftarrow A[k + j]$
    $A[k + j] \leftarrow u + t$
    $A[k + j + m/2] \leftarrow u - t$
    $\omega \leftarrow \omega \cdot \omega_m$

$\omega_m \leftarrow \exp(-2\pi i/m)$
$= \exp(-2\pi i/N * (N/m))$
=W_real[N/m] + i W_imag[N/m]

```
void fft_stage(int stage, DTYPE X_R[N], DTYPE
               X_I[N], DTYPE OUT_R[N], DTYPE OUT_I[N]) {
  int m = 1 << stage, mh = 1 << (stage-1);
  int offset = N >> stage;
  float w_r, w_i, u_r, u_i, a_r, a_i, t_r, t_i;
  for (int k = 0; k < N; k += m) {
    int exp = 0;
    for (int j = 0; j < mh; j++) {
      w_r = W_real[exp], w_i = W_imag[exp];
      u_r = X_R[k+j], u_i = X_I[k+j];
      a_r = X_R[k+j+mh], a_i = X_I[k+j+mh];
      t_r = (w_r * a_r - w_i * a_i);
      t_i = (w_r * a_i + w_i * a_r);
      OUT_R[k+j] = u_r + t_r;
      OUT_I[k+j] = u_i + t_i;
      OUT_R[k+j+mh] = u_r - t_r;
      OUT_I[k+j+mh] = u_i - t_i;
      exp += offset;
    }
  }
}
```

Create baseline (5)

- Run C-Simulation and C-Synthesis

```
void store_results(DTYPE X[N], DTYPE OUT[N]) {
  memcpy(OUT, X, N * sizeof(DTYPE));
}
```

Baseline analysis

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
fft	II Violation	-	-	-	-	-	-	no	76	12	4247	5408	0
VITIS_LOOP_46_1		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
VITIS_LOOP_36_1		-	-	-	-	-	10	no	-	-	-	-	-
VITIS_LOOP_61_1		-	-	-	1044	-	-	no	-	-	-	-	-
VITIS_LOOP_63_2	II Violation	-	1041	1.041E4	20	2	512	yes	-	-	-	-	-
Loop 3		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-
Loop 4		-	1025	1.025E4	3	1	1024	yes	-	-	-	-	-

undetermined trip count and latency

FFT Optimization

- Inner loops have various behavior

```
stage_loop:
for (s=1; s<=log2(N); s++) {
  m = 1 << s;
  ..
dft_loop:
  for (k=0; k<N; k+=m) {
    ..
butterfly_loop:
    for (j=0; j<m/2; j++) {
      ..
    }
  }
}
```

Iterations of `butterfly_loop` and `dft_loop` depend on induction variable "*s*" of `stage_loop`.

Thus, the `stage_loop` can't be simply *statically* pipelined.

Apply *dataflow* optimization among stages: *dynamic* pipelining

How much stages? $\log_2 N$

Add *dataflow*

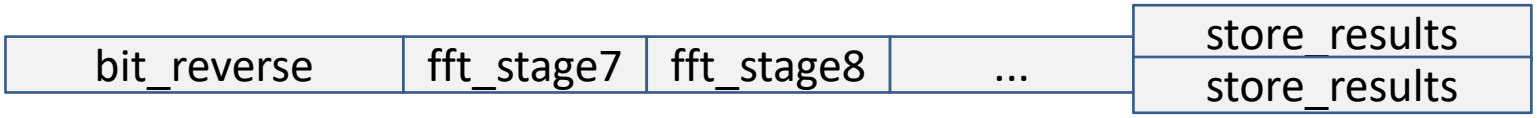
```
void fft(DTYPE X_R[N], DTYPE X_I[N], DTYPE OUT_R[N], DTYPE OUT_I[N])
{

#pragma HLS interface m_axi port=X_R
#pragma HLS interface m_axi port=X_I
#pragma HLS interface m_axi port=OUT_R
#pragma HLS interface m_axi port=OUT_I

#pragma HLS dataflow

  DTYPE stage_R[M+1][N], stage_I[M+1][N];
#pragma HLS array_partition variable=stage_R dim=1 complete
#pragma HLS array_partition variable=stage_I dim=1 complete

  bit_reverse(X_R, X_I, stage_R[0], stage_I[0]);

  for (int stage=0; stage<M; stage++) {
    #pragma HLS unroll
    fft_stage(stage+1, stage_R[stage], stage_I[stage], stage_R[stage+1],
stage_I[stage+1]);
  }

  store_results(stage_R[M], OUT_R);
  store_results(stage_I[M], OUT_I);
}
```

Dataflow analysis

Vitis HLS Console

```
INFO: [XFORM 203-712] Applying dataflow to function 'fft'
        'bit_reverse6'
        'fft_stage7'
        'fft_stage8'
        'fft_stage9'
        'fft_stage10'
        'fft_stage11'
        'fft_stage12'
        'fft_stage13'
        'fft_stage14'
        'fft_stage15'
        'fft_stage16'
        'store_results17'
        'store_results18'.
```

Modules & Loops	Issue Type	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval
fft		-	7374	7.374E4	-	1034
fft_stage15		-	530	5.300E3	-	530
fft_stage14		-	530	5.300E3	-	530
fft_stage13		-	530	5.300E3	-	530
fft_stage12		-	530	5.300E3	-	530
fft_stage11		-	530	5.300E3	-	530
fft_stage10		-	530	5.300E3	-	530
fft_stage9		-	530	5.300E3	-	530
fft_stage8		-	530	5.300E3	-	530
fft_stage7		-	530	5.300E3	-	530
fft_stage16		-	529	5.290E3	-	529
bit_reverse6		-	1033	1.033E4	-	1033
store_results17		-	1031	1.031E4	-	1031
store_results18		-	1031	1.031E4	-	1031

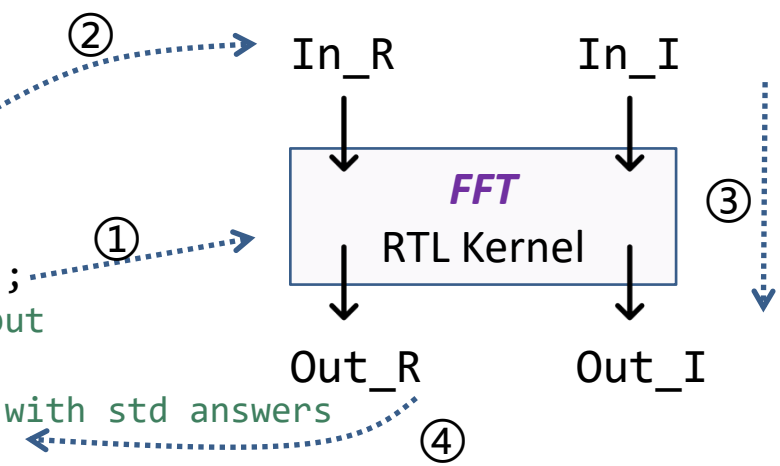


Name	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
fft_stage7	-	530	-	530	-
VITIS_LOOP_61_1_VITIS_LOOP_63_2	yes	528	18	1	512

C/RTL Cosimulation

- Test register-transfer level (RTL) design synthesized by the HLS tool
  1. Synthesize HLS kernel into RTL
  2. Converts inputs in testbench into input signals
  3. Feed input signals to RTL-level simulation, capture output signals
  4. Continue testbench program with the kernel function replaced by the captured signals.

```
//Generate input data
for(int i=0; i<N; i++){
  In_R[i] = i; In_I[i] = 0.0;
}
//Perform FFT
fft(In_R, In_I, Out_R, Out_I);
// comparing with golden output
for(int i=0; i<N; i++) {
  // .. compare Out_R, Out_I with std answers
}
```



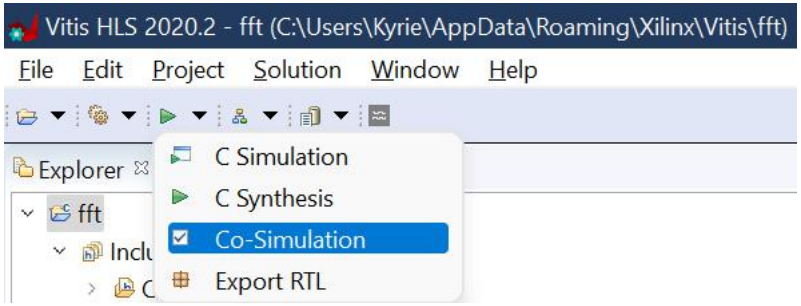
Why Co-Simulation?

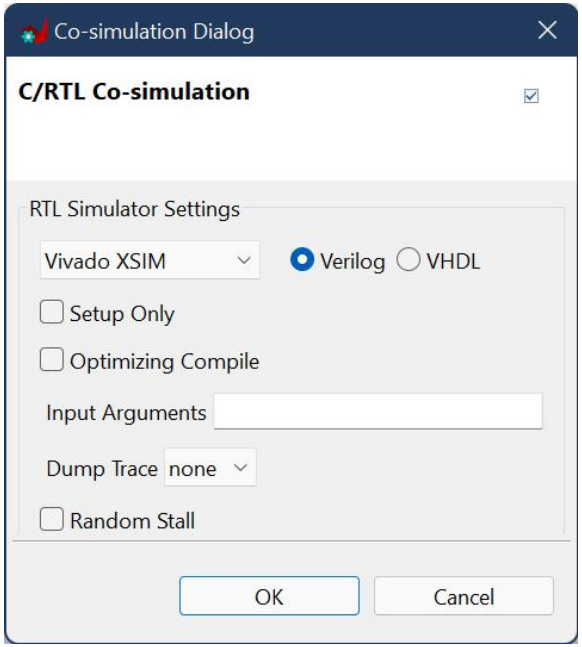
- Advantages
  - Reflect any optimizations
    - including architectual opts, such as pipelining
  - Accurate <u>cycle information</u>
    - especially ***stalling*** cycles in *dataflow*

- Disadvantages
  - cycle-by-cycle simulation is much slower
  - limited error messages

Co-Simulation

- Run Co-Simulation
  - this is SLOW!





Co-Simulation (2)

Cosimulation Report for 'fft'

General Information

Date: Sat Mar 19 13:02:25 CST 2022
Version: 2020.2 (Build 3064766 on Wed Nov 18 09:12:45 MST 2020)
Project: fft
Status: Pass

Solution: solution1 (Vivado IP Flow Target)
Product family: zynq
Target device: xc7z010-clg400-1

Cosim Options

Tool: Vivado XSIM

RTL: Verilog

Performance Estimates

Modules	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
fft				7386	7386	7386
fft_stage15				530	530	530
fft_stage14				530	530	530
fft_stage13				530	530	530
fft_stage12				530	530	530
fft_stage11				530	530	530
fft_stage10				530	530	530
fft_stage9				530	530	530
fft_stage8				530	530	530
fft_stage7				530	530	530
fft_stage16				529	529	529
bit_reverse6				1038	1038	1038
store_results17				1038	1038	1038
store_results18				1038	1038	1038

Co-Simulation (3)

Performance Estimates

Modules	Avg II	Max II	Min II	Avg Latency	Max Latency	Min Latency
					7386	7386
					530	530
					530	530
					530	530

- Open Dataflow Viewer
- Goto Source
- Open Old Cosimulation Report

Properties | Warnings | Guidance | C Source | Dataflow

Process | Channel

Name	Cosim Category	Cosim Stalling Time	Cosim Read Block Time	Cosim Write Block Time	Cosim Stall No Start	Cosim Stall No Continue	Cosim AVG II	Cosim Max II	Cosim Min II	Cosim AVG Latency	Cosim Max Latency	Cosim Min La
bit_reverse6_U0	none	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	1038	1038	1038
fft_stage7_U0	none	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	530	530	530
fft_stage8_U0	none	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	530	530	530
fft_stage9_U0	none	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	N/A	N/A	530	530	530

Better dataflow

Name	BRAM
Channels(22)	**88**
pingpong	**88**
stage_I_0_U	4
stage_I_1_U	4
stage_I_10_U	4
stage_I_2_U	4
stage_I_3_U	4
stage_I_4_U	4
stage_I_5_U	4
stage_I_6_U	4
stage_I_7_U	4
stage_I_8_U	4
stage_I_9_U	4
stage_R_0_U	4
stage_R_1_U	4
stage_R_10_U	4
stage_R_2_U	4
stage_R_3_U	4

use *FIFO*?

```
for (int k = 0; k < N; k += m) {
..
for (int j = 0; j < mh; j++) {
    ..
    u_r = X_R[k+j], u_i = X_I[k+j];
    a_r = X_R[k+j+mh], a_i = X_I[k+j+mh];
    ..
    OUT_R[k+j] = u_r + t_r;
    OUT_I[k+j] = u_i + t_i;
    OUT_R[k+j+mh] = u_r - t_r;
    OUT_I[k+j+mh] = u_i - t_i;
    ..
}
}
```

Not strictly ***In-Order***