ACA 2021 Lab #B
Beamforming Acceleration

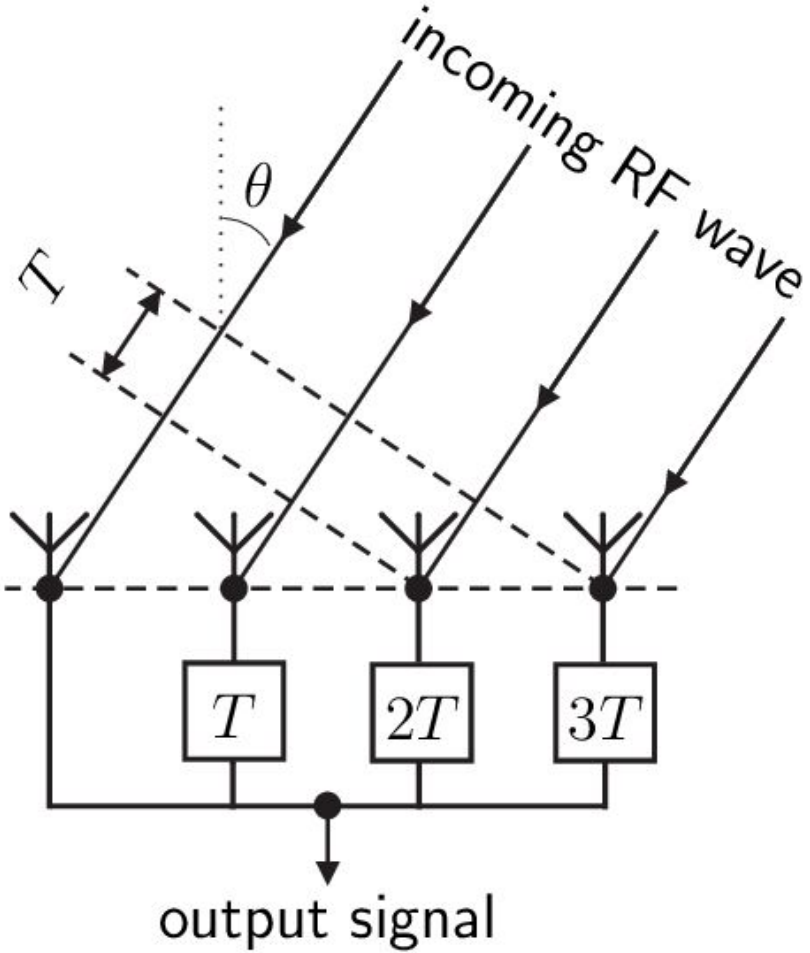
D09922024 游子緒
2021/5/11

GitHub: https://github.com/e841018/beamformer

Introduction

Beamforming

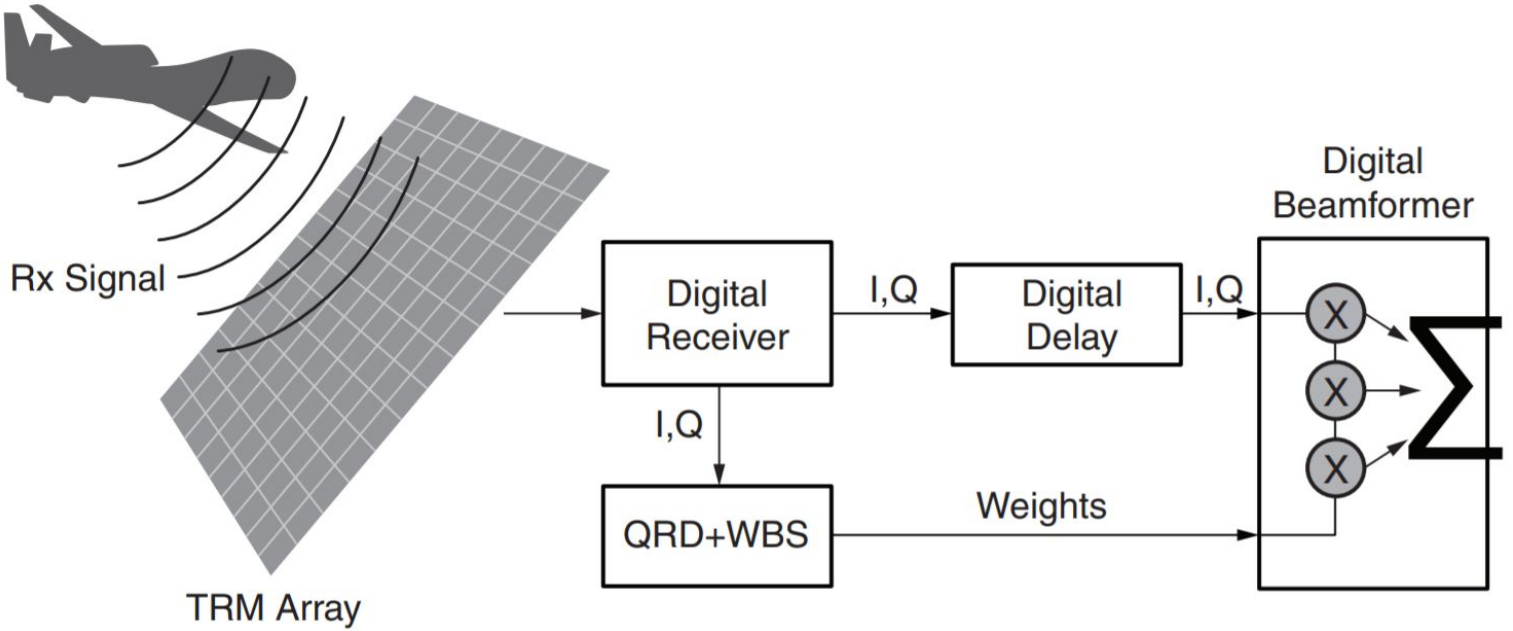
- Receiving or transmitting waves in a specific direction (a beam) in a radio system with multiple antennas (channels)
- Adaptive beamforming:
  Estimate the components from the multi-channel signals and steer the beams to correct directions



Tijmes, M.R. & Roeloffzen, Chris & Meijerink, Arjan & Bentum, M.J.. (2021).
Simulation of a ring-resonator based optical beamformer system for phased array receive antennas.

“Adaptive Beamforming for Radar: Floating-Point QRD+WBS in an FPGA”

- Whitepaper WP452 from Xilinx, written by Luke Miller in 2014



“Adaptive Beamforming for Radar: Floating-Point QRD+WBS in an FPGA”

The signals from a specific direction can be extracted by a set of weights:

$$\begin{pmatrix} ch1[t] & ch2[t] & \dots & chN[t] \\ ch1[t+1] & ch2[t+1] & \dots & chN[t+1] \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} w[1] \\ w[2] \\ \vdots \\ w[N] \end{pmatrix} = \begin{pmatrix} beam[t] \\ beam[t+1] \\ \vdots \end{pmatrix}$$

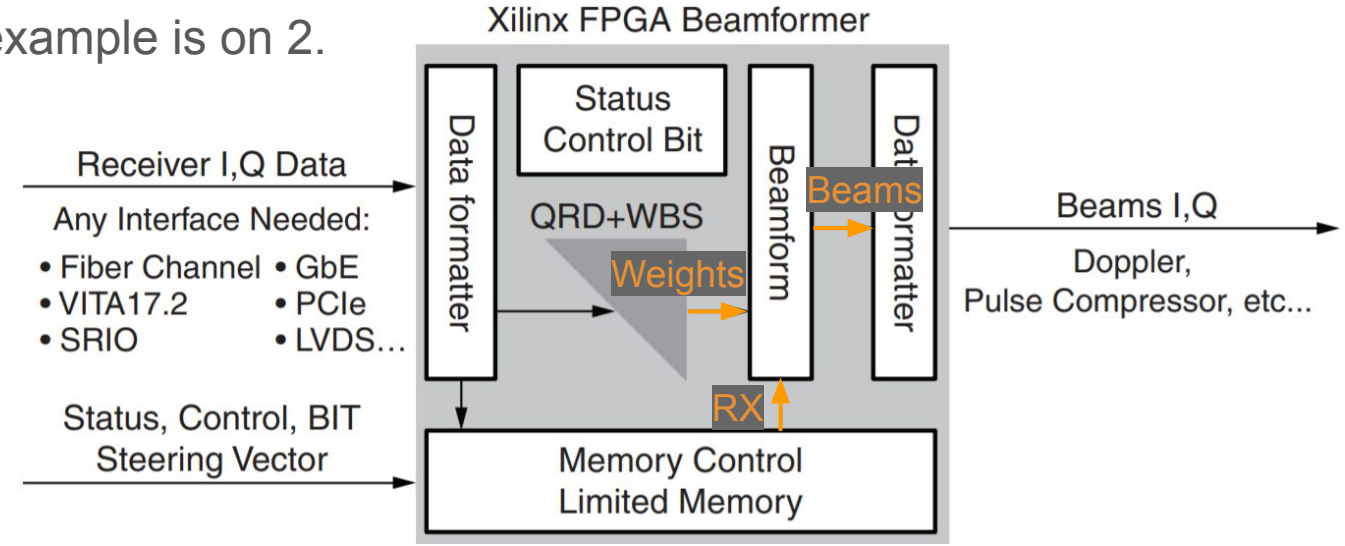
How to find the weights?

- By solving Ax=b, where A and b are known
- QRD+WBS: QR decomposition + weight back substitution (a matrix inversion algorithm)

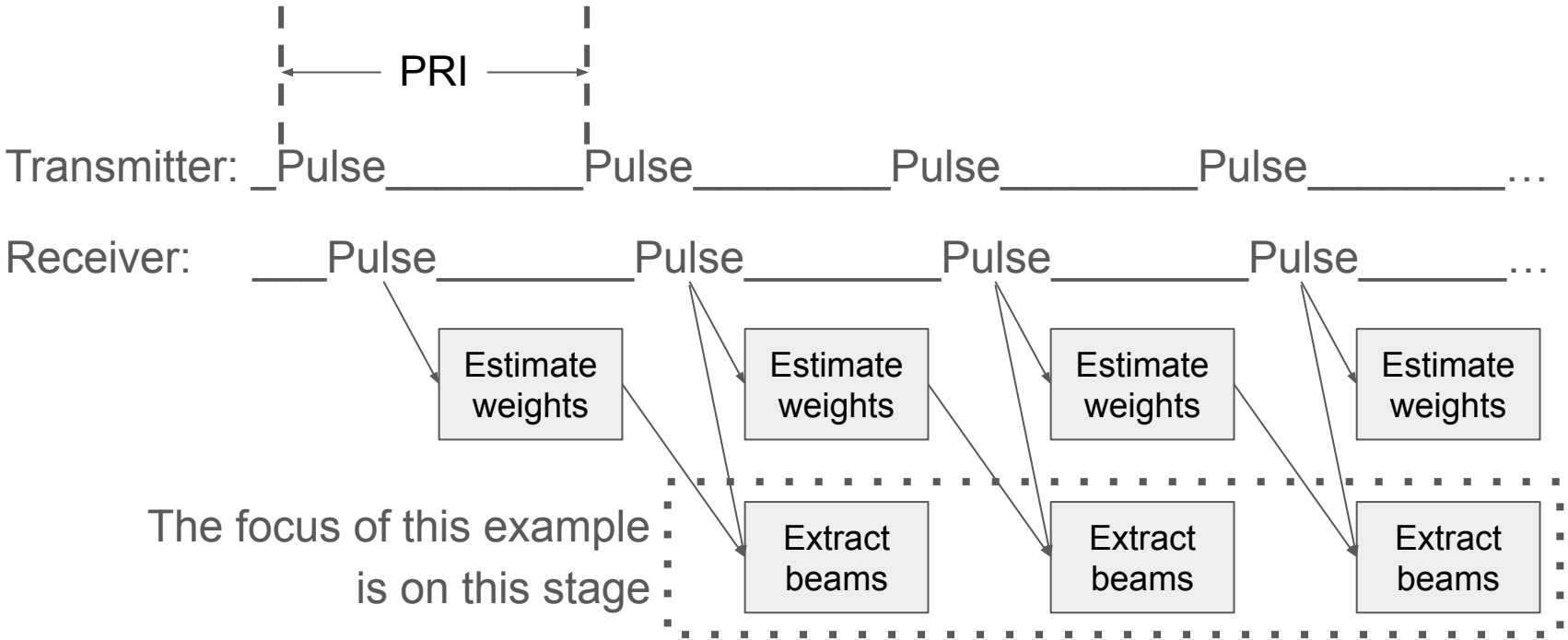
“Adaptive Beamforming for Radar: Floating-Point QRD+WBS in an FPGA”

1. Find the weights for each beam by QRD+WBS, using samples of last PRI
2. Use the weights to extract signals for each beam

The focus of this example is on 2.



Pulse repetition interval (PRI)



Implementation

Forming beams from channels

3 **complex** matrices involved.

- RX [SAMPLES] [CHANNELS]:
  received samples in each physical channel
- W [BEAMS] [CHANNELS]:
  estimated weights for each channel-beam pair
- B [SAMPLES] [BEAMS]:
  samples of each beam

For each beam, $B[:][b] = RX * W[b][:]^T$

Simplified into a single matrix multiplication: $B = RX * W^T$

Complex matrix multiplication

Rewrite $B = RX * W^T$ with 6 **real** matrices.

i: in-phase (real), q: quadrature (imaginary)

- RXi, RXq
- Wi, Wq
- Bi, Bq

$Bi = RXi * Wi^T - RXq * Wq^T$

$Bq = RXi * Wq^T + RXq * Wi^T$

Matrix multiplication: 3 levels of for loop

```
loop1: for (i = 0; i < SAMPLES; i++) {
    loop2: for (j = 0; j < BEAMS; j++) {
        int result_i = 0;
        int result_q = 0;
        loop3: for (k = 0; k < CHANNELS; k++) {
            result_i += RXi[i][k] * Wi[j][k] - RXq[i][k] * Wq[j][k];
            result_q += RXi[i][k] * Wq[j][k] + RXq[i][k] * Wi[j][k];
        }
        Bi[i][j] = result_i;
        Bq[i][j] = result_q;
    }
}
```

IO

```
void beamformer(const int *RXi_in1, const int *RXq_in1, // Read-Only Matrix 1
                const int *Wi_in2,  const int *Wq_in2,  // Read-Only Matrix 2
                int *Bi_out,        int *Bq_out) {      // Output Result
    // declare local arrays
    int RXi[SAMPLES][CHANNELS]; int RXq[SAMPLES][CHANNELS];
    int Wi[BEAMS][CHANNELS];    int Wq[BEAMS][CHANNELS];
    int Bi[SAMPLES][BEAMS];     int Bq[SAMPLES][BEAMS];
    readA: for (itr = 0; itr < SAMPLES * CHANNELS; itr++) {...} // Burst read RX
    readB: for (itr = 0; itr < BEAMS * CHANNELS; itr++) {...} // Burst read Wi
    loop1: ... // Matrix multiplication
    writeC: for (itr = 0; itr < BEAMS * CHANNELS; itr++) {...} // Burst write B
}
```

Optimizations

Experiment settings

- #channel = 16, #beam = 3, #sample = 2500 in a PRI
- FPGA: Alveo U200
- Clock: 100 MHz (10 ns)

Optimizations

Apply optimizations gradually:

1. Bundle IO into 2 groups
2. Pipeline matrix multiplication
3. Pipeline IO
4. Partition local arrays

		solution0_baseline	solution1_bundle	solution2_pipeline_L2	solution3_pipeline_IO	solution4_partition
Latency (cycles)	min	768032	362617	162618	107576	55076
	max	768032	362617	162618	107576	55076
Latency (absolute)	min	7.680 ms	3.626 ms	1.626 ms	1.076 ms	0.551 ms
	max	7.680 ms	3.626 ms	1.626 ms	1.076 ms	0.551 ms
Interval (cycles)	min	768032	362617	162618	107576	55076
	max	768032	362617	162618	107576	55076

Optimization 1: Bundle IO into 2 groups

Baseline (1 AXI Master ports)

```
set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmem "beamformer" RXi_in1

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmem "beamformer" RXq_in1

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmem "beamformer" Wi_in2

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmem "beamformer" Wq_in2

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmem "beamformer" Bi_out

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmem "beamformer" Bq_out
```

Bundled IO (2 AXI Master ports)

```
set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmemi "beamformer" RXi_in1

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmemq "beamformer" RXq_in1

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmemi "beamformer" Wi_in2

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmemq "beamformer" Wq_in2

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmemi "beamformer" Bi_out

set_directive_interface -mode m_axi -depth 128
-offset slave -bundle gmemq "beamformer" Bq_out
```

Latency: 7.6ms →3.6ms

Optimization 2: Pipeline matrix multiplication (II=1)

Which loop to pipeline?

- loop1: □
  unrolls loop2 and loop3
  uses lots of resource
- loop2: ✔
  unrolls loop3
  calculates 1 output per cycle
- loop3: □
  parallelism not fully exploited
  output is not pipelined

```
loop1: for (i = 0; i < SAMPLES; i++) {
    loop2: for (j = 0; j < BEAMS; j++) {
        int result_i = 0;
        int result_q = 0;
        loop3: for (k = 0; k < CHANNELS; k++) {
            result_i += RXi[i][k] * Wi[j][k] - RXq[i][k] * Wq[j][k];
            result_q += RXi[i][k] * Wq[j][k] + RXq[i][k] * Wi[j][k];
        }
        Bi[i][j] = result_i;
        Bq[i][j] = result_q;
    }
}
```

Latency: 3.6ms →1.6ms

Optimization 3: Pipeline IO (II=1)

Read / write 1 element from / to each array per cycle

```
for (itr = 0, i = 0, j = 0; itr < SAMPLES * CHANNELS; itr++, j++) {
    #pragma HLS PIPELINE II=1
    if (j == CHANNELS) {
        j = 0;
        i++;
    }
    RXi[i][j] = RXi_in1[itr];
    RXq[i][j] = RXq_in1[itr];
}
...
```

Latency: 1.6ms →1.1ms

Optimization 4: Partition local arrays

Which dimension to partition?

- Dim. 2 of RX and W are accessed in parallel
- Partition dim. 2 of RX and W with option “complete”

```
    loop1: for (i = 0; i < SAMPLES; i++) {
        loop2: for (j = 0; j < BEAMS; j++) {
            #pragma HLS PIPELINE II=1
            int result_i = 0;
            int result_q = 0;
            result_i += RXi[i][0] * Wi[j][0] - RXq[i][0] * Wq[j][0];
            result_q += RXi[i][0] * Wq[j][0] + RXq[i][0] * Wi[j][0];
            result_i += RXi[i][1] * Wi[j][1] - RXq[i][1] * Wq[j][1];
            result_q += RXi[i][1] * Wq[j][1] + RXq[i][1] * Wi[j][1];
            ...
            Bi[i][j] = result_i;
            Bq[i][j] = result_q;
        }
    }
```

loop3 unrolled

Latency: 1.1ms →0.55ms

Resource usage

1. Bundle: increased use of FF and LUT (one more AXI Master)
2. Pipeline loop2: much more DSP, FF and LUT
3. Pipeline IO: no significant change
4. Partition: 8 times DSP, much more BRAM and FF

	solution0_baseline	solution1_bundle	solution2_pipeline_L2	solution3_pipeline_IO	solution4_partition
BRAM_18K	180	182	184	184	258
DSP48E	12	12	24	24	192
FF	2004	2443	3033	2940	5747
LUT	2508	2934	5731	5824	6247
URAM	0	0	0	0	0

SLX FPGA

Recommended reading from Jiin: “Adaptive Beamformer: An HLS Optimization Case Study with SLX FPGA”

- Whitepaper from Silexica, written by Zubair Wadood in 2020
- Optimized the QRD+WBS part with SLX, giving slightly better results than domain experts

	Latency (ms)	Speed-up
Unoptimized	14.75	1
SLX 20.1	2.08	7.0x
SLX 20.1-sp1	**1.82**	**8.1x**
Hand optimized	1.94	7.6x

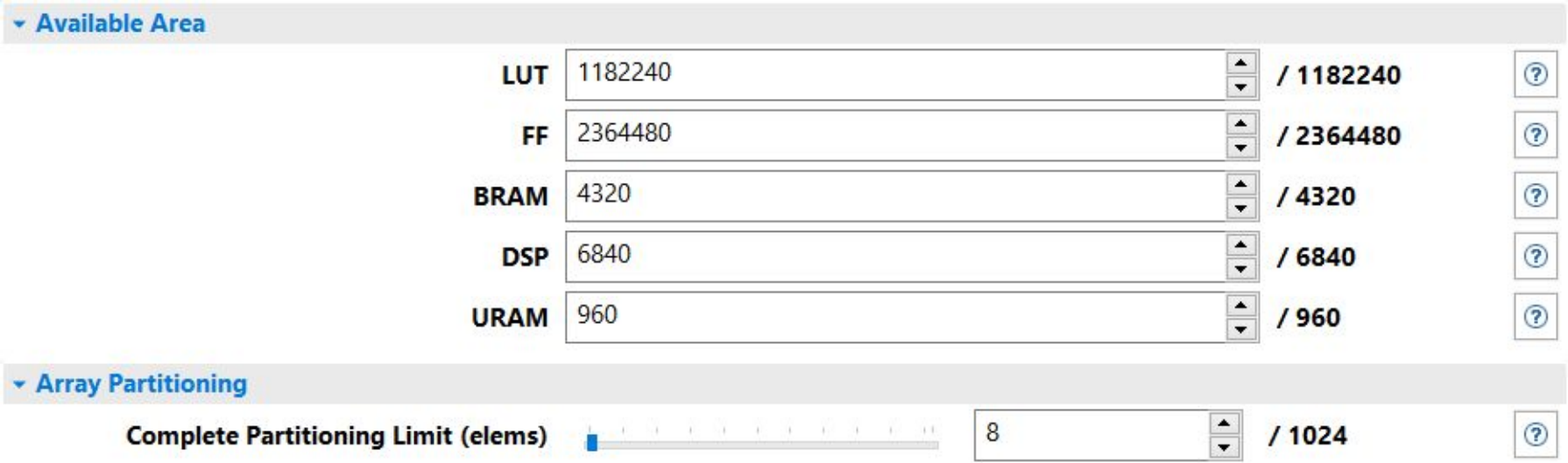
Apply SLX to the beamformer part

- SLX requires all arrays to be of known size at compile time
- `<type> *<var>` and `<type> <var>[<len>]` are the same to a C compiler, but SLX takes `<len>` as a hint

```
void beamformer(
    const int *RXi_in1,
    const int *RXq_in1,
    const int *Wi_in2,
    const int *Wq_in2,
    int *Bi_out,
    int *Bq_out
) {...}
```

```
void beamformer(
    const int RXi_in1[SAMPLES * CHANNELS],
    const int RXq_in1[SAMPLES * CHANNELS],
    const int Wi_in2[BEAMS * CHANNELS],
    const int Wq_in2[BEAMS * CHANNELS],
    int Bi_out[SAMPLES * BEAMS],
    int Bq_out[SAMPLES * BEAMS]
) {...}
```

Default constraints



Default optimizaitons

Hand-crafted (latency = 0.551 ms)

- pipeline readA, readB, writeC
- pipeline loop2 (unroll loop3)
- partition RX, W (**complete**)

SLX (latency = 0.625 ms)

- rechape IO arguments, (cyclic, factor=4), unroll readA (full)
- pipeline loop1 (unroll loop2, loop3)
- partition RX, W (**cyclic, factor=4, dim=2**)

Hand-crafted:

Loop Name	Latency (cycles) min	Latency (cycles) max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
readA	**40001**	**40001**	**3**	1	1	40000	**yes**
readB	48	48	2	1	1	48	yes
loop1_loop2	7501	7501	3	1	1	7500	yes
writeC	7501	7501	3	1	1	7500	yes

SLX:

Loop Name	Latency (cycles) min	Latency (cycles) max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
readA	**50000**	**50000**	**5**	-	-	10000	**no**
loop1	5002	5002	5	2	1	2500	yes
writeC	7500	7500	4	-	-	1875	no

Trial 1: Complete Partitioning Limit 8 -> 64

Branch-and-bound algorithm fails, resulting in unoptimized pragmas

```
Running branch & bound algorithm
⋮
Step 1021, 241301 configurations remaining. 4 alive branches, min/max latency: 192616/25194823
Step 1022, 236501 configurations remaining. 4 alive branches, min/max latency: 192616/25194823
Step 1023, 234341 configurations remaining. 4 alive branches, min/max latency: 192616/25194823
Step 1024, 234101 configurations remaining. 3 alive branches, min/max latency: 192616/25194823
Step 1152, 15601 configurations remaining. 3 alive branches, min/max latency: 192616/194823
spec:0:0: info: More parallelism is available but cannot be exploited because all the area is exhausted.
```

```
Running branch & bound algorithm
Creating multi-dim VSS for Variable 'RXi' ..\beamformer\beamformer.c:66:0 VarId: 20
Creating multi-dim VSS for Variable 'RXq' ..\beamformer\beamformer.c:67:0 VarId: 21
Creating multi-dim VSS for Variable 'Bi' ..\beamformer\beamformer.c:70:0 VarId: 24
Creating multi-dim VSS for Variable 'Bq' ..\beamformer\beamformer.c:71:0 VarId: 25
Size of relevant design space is 7.40301e+07 configurations.
warning: spec:0:0: No configuration of the initial design space fits in the device.
```

Trial 2: Manually pipelining readA

Latency: 0.625 ms -> 0.525 ms, slightly better than hand-crafted

Loop Name	Latency (cycles) min	Latency (cycles) max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
- readA	**50000**	**50000**	**5**	-	-	10000	**no**
- loop1	5002	5002	5	2	1	2500	yes
- writeC	7500	7500	4	-	-	1875	no

Loop Name	Latency (cycles) min	Latency (cycles) max	Iteration Latency	Initiation Interval achieved	Initiation Interval target	Trip Count	Pipelined
- readA	**40001**	**40001**	**6**	4	1	10000	**yes**
- loop1	5002	5002	5	2	1	2500	yes
- writeC	7500	7500	4	-	-	1875	no

Resource usage: hand-crafted v.s. SLX + pipeline_readA

- Pipelining loop1 requires much more expressions than pipelining loop2
- Partitioning with (cyclic, factor=4, dim=2) uses less memory than (complete)

Hand-crafted

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	**-**	**192**	**0**	**3249**	**-**
FIFO	-	-	-	-	-
Instance	4	-	1288	2000	-
Memory	**254**	**-**	**0**	**0**	**0**
Multiplexer	-	-	-	998	-
Register	-	-	4459	-	-
Total	258	192	5747	6247	0

SLX+ pipeline_readA

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	**-**	**576**	**0**	**9262**	**-**
FIFO	-	-	-	-	-
Instance	16	-	1484	1998	-
Memory	**182**	**-**	**0**	**0**	**0**
Multiplexer	-	-	-	1204	-
Register	-	-	5729	-	-
Total	198	576	7213	12464	0

Conclusion

- SLX helps in quickly exploring possibilities,
  but doesn’t guarantee to give optimal pragmas
- Some further modification in source code may be required to use SLX,
  since the analysis needs more hints than required by vivado HLS

Appendix: unsolved issue on Vitis

- Tried to validate the optimizations with Vitis
- SW emulation passed, but HW emulation failed
- A thread about this issue is found on Xilinx forum,
  but the solution doesn't work

```
[21:23:29] Run vpl: Step config_hw_emulation: RUNNING...
[21:24:45] Run vpl: Step config_hw_emulation: RUNNING...
[21:25:09] Run vpl: Step config_hw_emulation: Failed
[21:25:13] Run vpl: FINISHED. Run Status: config_hw_emulation ERROR
```

Takeaways

- In a nested loop shown below, what are the pros and cons of pipelining loop1 instead of loop2?

```
loop1: for (i = 0; i < SAMPLES; i++)
    loop2: for (j = 0; j < BEAMS; j++)
        loop3: for (k = 0; k < CHANNELS; k++)
            B[i][j] += RX[i][k] * W[j][k];
```

- If the loop1 is pipelined, which dimensions of the array `W` should be partitioned to allow II=1?