MaxCompiler Optimization Cheat Sheet

v2 Interface

MAXELER Technologies

Kernel Optimization

Global Optimizations

Accessed via: KernelConfiguration.optimization

How to apply:

```
Manager m = new Manager("Simple", BOARDMODEL);
KernelConfiguration currKConf = m.getCurrentKernelConfig();
currKConf.optimization.setCEPipelining(2);
Kernel k = new MyKernel( m.makeKernelParameters() );
```

setCEPipelining(int pipelining)

Default: 2

Arguments: $2 \leq$ `pipelining`

Description: sets the number of pipelining stages to add to the clock enable signal.

When to use: Increase the number of pipelining stages if you have timing issues with the clock enable signal.

setCEReplicationNumPartitions(int log2NumPartitions)

Default: 0 (i.e. $2^0 = 1$ partition)

Description: sets the number of partitions for the clock enable signal.

When to use: Increase the number of partitions to reduce the fanout from the clock enable signal.

setUseGlobalClockBuffer(boolean enabled)

Default: disabled

Description: sets a dedicated, global clock buffer for a Kernel.

When to use: This can improve timing for Kernels, but there a limited number of global clock buffers available. Use for larger Kernels in a multi-Kernel design.

Push-Pop Optimizations

Accessed via: optimization.push<OptName>

How to apply:

```
optimization.pushPipeliningFactor(0.5);
// code that optimization applies to
optimization.popPipeliningFactor();
```

pushPipeliningFactor(double pipelining)

Arguments: $0.0 \leq$ `pipelining` ≤ 1.0

Default: 1.0

Description: decrease the pipeline depth of all operations. 0.0 creates unpipelined operations. 1.0 creates fully pipelined operations.

When to use: adjust this setting if you are overmapped on flip-flops or latency is an issue on a section of your Kernel.

pushDSPFactor(double dsp_factor)
pushDSPFactor(double dsp_factor, MathOps operations)

Arguments: $0.0 \leq$ `dsp_factor` ≤ 1.0

Default: 0.5

Arguments: operations = {ADD, SUB, MUL, DIV, NEG, ALL, ADD_SUB, MUL_DIV}

Description: increase or decrease the number of DSPs used for all operations or a specific type of operation.

DSP Factor Mapping to Xilinx CoreGen Options

DSP Factor mapping for **floating-point multiplication**:

DSP Factor (d)	Type	DSP Usage
$0.0 \leq d < 0.25$	8,24 or 11,53	No_Usage
$0.25 \leq d < 0.5$	8,24 or 11,53	Medium_Usage
$0.5 \leq d < 1.0$	8,24 or 11,53	Full_Usage
1.0	8,24 or 11,53	Max_Usage
$0.0 \leq d < 0.25$	custom	No_Usage
$0.5 \leq d < 1.0$	custom	Full_Usage
1.0	custom	Max_Usage

DSP Factor mapping for **floating-point addition**:

DSP Factor (d)	Type	DSP Usage
$0.0 \leq d < 0.75$	8,24 or 11,53	No_Usage
$0.75 \leq d < 1.0$	8,24 or 11,53	Full_Usage
$0.0 \leq d \leq 1.0$	custom	No_Usage

DSP Factor mapping for **fixed-point multiplication**:

DSP Factor (d)	Multiplier Construction	Opt Goal
$0.0 \leq d < 0.166$	Use_LUTs	n/a
$0.166 \leq d < 0.333$	Use_Mults	Area
$0.333 \leq d \leq 1.0$	Use_Mults	Speed

Placement Constraints

Accessed via: optimization.setGroup<Prim>AreaConstraint

How to apply:

```
pushGroup("name_of_area");
// <Prim> is one of: 'DSP', 'RAMB18', 'RAMB36' or 'Slice'
optimization.setGroup<Prim>AreaConstrant(x0, y0, x1, y1);
// code that optimization applies to
popGroup();
```

Default: no area groups specified.

Description: constrain placement of the specified primitives to the area defined by `(x0, y0)` - `(x1, y1)`.

When to use: improves timing in cases where build tools perform sub-optimial placement.

Input Registering

Accessed via: io.pushInputRegistering

How to apply:

```
io.pushInputRegistering(false);
// code that setting applies to
io.popInputRegistering();
```

Default: enabled.

Description: disables input registering.

When to use: reduces latency and flip-flop usage.

Warning: Disabling input registering changes the behavior of an input: reading when the input is not enabled reads *undefined data*.

Per-Stream Optimizations

Accessed via: optimization.<OptName>(streamName)

How to apply:

```
DFEVar result = x*x + x;
result = optimization.pipeline(result);
```

pipeline(T stream)

Description: adds a layer of registers to the stream.

When to use: Enable this if have a timing failure due to fanout from a stream.

limitFanout(T stream, int max_fanout)

Description: adds a layer of registers to the stream and tries to duplicate any of these registers where the fanout would be more than `max_fanout`.

When to use: Enable this if you have a timing failure due to fanout from a stream.

Manager Optimizations

LMem Clock Frequency

How to apply:

```
config.setOnCardMemoryFrequency(LMemFrequency clock);
```

Default: MAX3_303 (MAX3) or MAX2_200 (MAX2).

Arguments: LMemFrequency {MAX2_200, MAX2_300, MAX3_303, MAX3_333, MAX3_400}

Description: sets the base clock frequency for the LMem.

When to use: Decrease this if there are infrastructure timing errors reported against the LMem. Increase this to get more bandwidth out of the LMem.

The **timing report** can be found in the build directory in `timingreport/index.html`

Stream Clock Frequency

Default Stream Clock Frequency

How to apply:

```
config.setDefaultStreamClockFrequency(120);
```

Default: 100.

Description: sets the default clock frequency for Manager blocks.

When to use: Increase this to improve performance. Decrease this to meet timing.

Configurable Stream Clock Frequency

How to apply:

```
ManagerClock myClk = generateStreamClock("myClk", 120);
myKernel.setClock(myClock);
```

Description: sets a clock frequency for a Manager block.

When to use: Increase this to improve performance. Decrease this to meet timing.

Build Configuration

Accessed via: `BuildConfig.set<OptName>`

How to apply:

```
BuildConfig bc = m.getBuildConfig();
bc.setBuildEffort(Effort.VERY_HIGH);
bc.setMPPRCostTableSearchRange(1, 10);
```

setBuildEffort(Effort effort)

Arguments: `Effort {LOW, MEDIUM, HIGH, VERY_HIGH}`

Description: sets the effort level for the Xilinx place and route tools.

Mapping to Xilinx `par` options:

	`-ol`	`-xe`
`LOW`	`std`	`""`
`MEDIUM`	`high`	`""`
`HIGH`	`high`	`"n"`
`VERY_HIGH`	`high`	`"c"`

setMPPRCostTableSearchRange(int min, int max)

Arguments: $1 \leq$ `min` $\leq$ `max` ≤ 100

Description: enables multi-pass place and route with a range of cost tables from `min` to `max`.

setMPPRParallelism(int p)

Description: specifies the number of place and route processes to run in parallel.