XTA command-line manual

IN THIS DOCUMENT

This chapter lists all the commands and options supported by the XTA, reference classes, and a reference to the grammar.

1 Commands

1.1 add

`add branch <from BRANCH> [<to INSTRUCTION>]+`
: Adds the given from/to references to the branches list

`add tile <tile id|*>`
: Add xCORE tile to active set

`add exclusion <ANY>`
: Adds the given reference to the list of exclusions

`add functiontime <FUNCTION> <value> <MODE>`
: Adds the given function time to the list of defines

`add instructiontime <ENDPOINT> <value> <MODE>`
: Adds the given instruction time to the list of defines

`add loop <ANY> <iterations>`
: Adds the given loop count define to the list of defines

`add looppath <ANY> <iterations>`
: Adds the given loop path count define to the list of defines

`add loopscope <ANY> <SCOPE>`
: Adds the given loop scope define to the list of defines

Publication Date: 2013/11/13 REV A

`add pathtime <from ENDPOINT> <to ENDPOINT> <value> <MODE>`
: Adds the given path time to the list of defines

1.2 analyze

`analyze endpoints <from ENDPOINT> <to ENDPOINT>`
: Analyzes between the specified endpoints

`analyze function <FUNCTION>`
: Analyzes the given function

`analyze loop <ANY>`
: Analyzes the given loop

1.3 config

`config case <best/worst>`
: Sets the case (currently: worst)

`config Ewarning <on/off>`
: Treats errors as warnings or not (currently: off)

`config freq <node id> <tile frequency>`
: Sets the operating frequency in MHz for the given node

`config from <ENDPOINT>`
: Sets the from endpoint

`config looppoint <ANY>`
: Sets the loop point

`config scale <true/false>`
: Configures whether results are scales (currently: true)

`config srcpaths <paths>`
: Sets the (semicolon separated) source search path

`config cores <tile id> <num cores>`
: Sets number of cores currently executing for the given tile

`config timeout <seconds>`
: Sets the tools timeout on load

`config Terror <on/off>`
: Treat timing failures as errors or not (currently: on)

`config to <ENDPOINT>`
: Sets the to endpoint

`config verbosity <level>`
: Sets the tool verbosity level (range: -10 -> +10, default: 0)

`config Werror <on/off>`
: Treats warnings as errors or not (currently: off)

1.4 clear

`clear()`
: Clears the screen (GUI mode only)

1.5 debug

`debug dumpactiveexclusions()`
: Dumps a list of PCs that the exclusions have resolved to

`debug dumpcachedfunction <FUNCTION>`
: Dumps the cached function structure

`debug dumpcallgraph()`
: Dumps the call graph for all tiles in dot (graphviz) format

`debug dumpcontrolflow <FUNCTION>`
: Dumps the control flow graph for the given function in dot (graphviz) format

`debug dumpmanual()`
: Dumps the console reference chapter of the manual in tex format

`debug dumpstacknodes <REFERENCE>`
: Dumps the stack nodes for the given reference

`debug dumpunresolvedinstructions()`
: Dumps a list of instructions that are unresolved

`debug verifyreference <ANY>`
: Verifies the existance of the given reference

`debug frompoints()`
: Displays the from endpoints currently configured

`debug topoints()`
: Displays the to endpoints currently configured

`debug instructiontime <route id> <node id>`
: Displays the instruction time set for the given node in the given route

`debug loop <route id> <node id>`
: Displays the loop iterations set for the given node in the given route

`debug looppath <route id> <node id>`
: Displays the loop path iterations set for the given node in the given route

`debug loopscope <route id> <node id>`
: Displays the loop scope set for the given node in the given route

`debug listglobalreferences <ANY>`
: Lists all the matching references for the given reference on the global tree

`debug listroutereferences <route id> <ANY>`
: Lists all the matching references for the given reference on the given route

`debug memusage()`
: Displays the current memory usage for the JVM

`debug getmemthreshold()`
: Displays the current memory usage threshold

`debug setmemthreshold <threshold>`
: Sets the memory threshold to the given value (0.0 - 1.0)

1.6 echo

`echo "text"`
: Prints the text to the console

1.7 exit

`exit()`
: Quits the application

1.8 help

`help [command|command subcommand|option]`
: Displays help message for the given arguments

1.9 history

`history()`
: Displays the command history

1.10 load

`load <xe file>`
: Loads the given XMOS executable file

1.11 list

`list allcalls()`
: Lists all the possible locations for calls

`list allendpoints()`
: Lists all the possible locations for endpoints

`list branches [route id]`
: Lists the branches - optionally for the specified route

`list calls()`
: Lists the calls

`list tiles()`
: Lists the active xCORE tiles

`list endpoints()`
: Lists the endpoints

`list exclusions [route id]`
: Displays the exclusions - optionally for the specified route

`list functions()`
: Lists the functions in the loaded application

`list functiontimes()`
: Displays the function time defines

`list instructiontimes()`
: Displays the instruction time defines

`list knowns <route id>`
: Displays the list of knowns set for the given route

`list labels()`
: Lists the labels

`list loops()`
: Displays the loop defines

`list looppaths()`
: Displays the loop path defines

`list loopscopes()`
: Displays the loop scope defines

`list pathtimes()`
: Displays the path time defines

`list sources()`
: Lists the source files

`list srccommands()`
: Displays the command list embedded in the loaded executable

`list srcloops()`
: Displays the loop counts embedded in the loaded executable

`list srclabels()`
: Lists the source labels

`list allsrclabels()`
: Lists all the possible locations for source labels

`list corestartpoints()`
: Lists the logical core start points

`list corestoppoints()`
: Lists the logical core stop points

`list unknowns <route id>`
: Displays the list of unknowns for the given route

1.12 print

`print summary()`
: Shows routes summary (verbosity -2|-1|0)

`print structure <route id> [node id]`
: Displays the structure for given route/node (verbosity 0|1)

`print asm <route id> [node id]`
: Displays annotated assembly for the given route/node

`print src <route id> [node id]`
: Displays annotated source file(s) for given route/node

`print trace <route id> [node id]`
: Displays instruction trace for the worst case path of the given route/node

`print routeinfo <route id>`
: Shows detailed information for the given route

`print nodeinfo <route id> <node id>`
: Shows detailed information for the given node in the given route

`print warnings()`
: Prints all timing warnings

`print distribution <route id> [node id]`
: Displays time distribution for the given route/node

1.13 pwd

`pwd()`
: Displays the current working directory

1.14 remove

`remove branch <from BRANCH|*> [<to INSTRUCTION|*>]+`
: Removes the given from/to references from the branches list

`remove tile <tile id|*>`
: Removes xCORE tile from active set

`remove exclusion <ANY|*>`
: Removes the given reference (or all if '*') from the list of exclusions

`remove functiontime <FUNCTION|*>`
: Removes the given functon time from the list of defines

`remove instructiontime <ENDPOINT|*>`
: Removes the given instruction time from the list of defines

`remove loop <ANY|*>`
: Removes the given loop count define to the list of defines

`remove looppath <ANY|*>`
: Removes the given loop path count define to the list of defines

`remove loopscope <ANY|*>`
: Removes the given loop scope define to the list of defines

`remove pathtime <from ENDPOINT|*> <to ENDPOINT|*>`
: Removes the given path time from the list of defines

`remove route <route id>`
: Removes the route with the given id from the current analysis

1.15 scripter

`scripter disable <ANY>`
: Disables a mapping

`scripter dump()`
: Dumps script which represents the current state - also embeds active pragmas into source

`scripter embed <filename>`
: Embeds the script into the designated file - also embed active pragmas into source

`scripter enable <ANY>`
: Enables a mapping

`scripter listrefs()`
: Lists all references which will be used in the script

`scripter rename <ANY> <TO_NAME>`
: Renames a mapping

1.16 set

`set exclusion <route id> <ANY>`
: Sets an exclusion on the given reference

`set functiontime <route id> <FUNCTION> <value> <MODE>`
: Sets timing requirement for the given function on the given route

`set instructiontime <route id> <ENDPOINT> <value> <MODE>`
: Sets the time taken for the instruction at the given pc

`set loop <route id> <ANY> <iterations>`
: Sets the number of iterations for the loop identified

`set looppath <route id> <ANY> <iterations>`
: Sets the number of iterations for the path identified

`set loopscope <route id> <ANY> <SCOPE>`
: Sets the scope of the referenced loop

`set pathtime <route id> <from ENDPOINT> <to ENDPOINT> <value> <MODE>`
: Sets timing requirement for the given path on the given route

`set required <route id> <value> <MODE>`
: Sets the maximum allowed time taken for the given route

1.17 source

`source <file name> [args]`
: Sources the given script file

1.18 status

`status()`
: Displays current status

1.19 version

`version()`
: Displays the version information

2 Pragmas

`#pragma xta label "name"`
: Provides a label that can be used to specify timing constraints.

`#pragma xta endpoint "name"`
: Specifies an endpoint. It may appear before an input or output statement.

`#pragma xta call "name"`
: Defines a label for a (function) call point. Use to specify a particular called instance of a function. For example, if a function contains a loop, the iterations for this loop can be set to a different value depending on which call point the function was called from.

`#pragma xta command "command"`
: Allows XTA commands to be embedded into source code. All commands are run every time the binary is loaded into the XTA. Commands are executed in the order they occur in the file, but the order between commands in different source files is not defined.

`#pragma xta loop "integer"`
: Applies the given loop XTA iterations to the loop containing the pragma.

3 Timing Modes

The available timing modes are:

`ns()`
: nanoseconds

`us()`
: microseconds

`ms()`
: milliseconds

`MHz()`
: megahertz

`KHz()`
: kilohertz

`Hz()`

hertz

`cycles()`

The core cycle count is the number of scheduled slots that the logical core required to perform the sequence. The relationship between core cycles and time is a function of the number of cores currently running and the xCORE tile frequency.

4 Loop Scopes

Supported values for scope are:

`relative/r()`

Iteration number propagates to the enclosing path (Default)

`absolute/a()`

Absolute number of iterations

5 Reference Classes

5.1 FUNCTION

`FunctionPc()`

Raw program counter specified in the format: 0x*

`Function()`

Any function

5.2 BRANCH

`EndpointPC()`

Raw program counter specified in the format: 0x*

`CallPc()`

Call specified in the format: 0x*

`CallFileLine()`

Call specified in the format: 'file name:line number'

`Call()`

Call specified using the source level pragma mechanism

`Label()`

Any source or assembly level symbol defined with respect to an executable section

`CallLabel()`

Any source or assembly level symbol defined with respect to an executable section

5.3 INSTRUCTION

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`FunctionPc()`
: Raw program counter specified in the format: 0x*

`Function()`
: Any function

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

5.4 ENDPOINT

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`EndpointFileLine()`
: Endpoint specified in the format: 'file name:line number'

`Endpoint()`
: Endpoint specified using the source level pragma mechanism

`CallPc()`
: Call specified in the format: 0x*

`CallFileLine()`
: Call specified in the format: 'file name:line number'

`Call()`
: Call specified using the source level pragma mechanism

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

`CallLabel()`
: Any source or assembly level symbol defined with respect to an executable section

5.5 ANY

`SrcLabelPc()`
: Raw program counter specified in the format: 0x*

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`EndpointFileLine()`
: Endpoint specified in the format: ‘file name:line number’

`Endpoint()`
: Endpoint specified using the source level pragma mechanism

`CallPc()`
: Call specified in the format: 0x*

`CallFileLine()`
: Call specified in the format: ‘file name:line number’

`Call()`
: Call specified using the source level pragma mechanism

`SrcLabelFileLine()`
: Source label specified in the format: ‘file name:line number’

`SrcLabel()`
: Source label specified using the source level pragma mechanism

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

`CallLabel()`
: Any source or assembly level symbol defined with respect to an executable section

5.6 FUNCTION_WITH_EVERYTHING

`EverythingReference()`
: Matches everything: ‘*’

`FunctionPc()`
: Raw program counter specified in the format: 0x*

`Function()`
: Any function

5.7 BRANCH_WITH_EVERYTHING

`EverythingReference()`
: Matches everything: ‘*’

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`CallPc()`
: Call specified in the format: 0x*

`CallFileLine()`
: Call specified in the format: ‘file name:line number’

`Call()`
: Call specified using the source level pragma mechanism

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

`CallLabel()`
: Any source or assembly level symbol defined with respect to an executable section

5.8 INSTRUCTION_WITH_EVERYTHING

`EverythingReference()`
: Matches everything: ‘*’

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`FunctionPc()`
: Raw program counter specified in the format: 0x*

`Function()`
: Any function

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

5.9 ENDPOINT_WITH_EVERYTHING

`EverythingReference()`
: Matches everything: ‘*’

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`EndpointFileLine()`
: Endpoint specified in the format: ‘file name:line number’

`Endpoint()`
: Endpoint specified using the source level pragma mechanism

`CallPc()`
: Call specified in the format: 0x*

`CallFileLine()`
: Call specified in the format: ‘file name:line number’

`Call()`
: Call specified using the source level pragma mechanism

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

`CallLabel()`
: Any source or assembly level symbol defined with respect to an executable section

5.10 ANY_WITH_EVERYTHING

`EverythingReference()`
: Matches everything: '*'

`SrcLabelPc()`
: Raw program counter specified in the format: 0x*

`EndpointPC()`
: Raw program counter specified in the format: 0x*

`EndpointFileLine()`
: Endpoint specified in the format: 'file name:line number'

`Endpoint()`
: Endpoint specified using the source level pragma mechanism

`CallPc()`
: Call specified in the format: 0x*

`CallFileLine()`
: Call specified in the format: 'file name:line number'

`Call()`
: Call specified using the source level pragma mechanism

`SrcLabelFileLine()`
: Source label specified in the format: 'file name:line number'

`SrcLabel()`
: Source label specified using the source level pragma mechanism

`Label()`
: Any source or assembly level symbol defined with respect to an executable section

`CallLabel()`
: Any source or assembly level symbol defined with respect to an executable section

6 XTA Jython interface

The Jython interface to the global xta object is as follows:

6.1 Load methods

```
void load(String fileName) throws Exception
```

6.2 Route creation/deletion methods

```
List<Integer> analyzeFunction(String functionName) throws Exception
List<Integer> analyzeEndpoints(String fromRef, String toRef) throws Exception
List<Integer> analyzeLoop(String loopRef) throws Exception
void removeRoute(int routeId) throws Exception
```

6.3 Add/remove methods

```
void addTile(String tileReference) throws Exception
void removeTile(String tileReference) throws Exception
Collection<String> getTiles() throws Exception

void addExclusion(String ref) throws Exception
void removeExclusion(String ref) throws Exception
Collection<String> getExclusions() throws Exception

void addBranch(String fromRefString, Collection<String> toRefStrings)
throws Exception
void removeBranch(String fromRefString, Collection<String> toRefStrings)
throws Exception
Collection<String> getBranches() throws Exception
Collection<String> getBranchTargets(String branch) throws Exception

void addLoop(String ref, long iterations) throws Exception
void removeLoop(String ref) throws Exception
Collection<String> getLoops() throws Exception

void addLoopPath(String ref, long iterations) throws Exception
void removeLoopPath(String ref) throws Exception
Collection<String> getLoopPaths() throws Exception

void addLoopScope(String ref, boolean absolute) throws Exception
void removeLoopScope(String ref) throws Exception
Collection<String> getLoopScopes() throws Exception

void addInstructionTime(String ref, double value, String units)
throws Exception
void removeInstructionTime(String ref) throws Exception
Collection<String> getInstructionTimes() throws Exception
```

```
void addFunctionTime(String ref, double value, String units)
throws Exception
void removeFunctionTime(String ref) throws Exception
Collection<String> getFunctionTimes() throws Exception

void addPathTime(String fromRef, String toRef, double value, String units)
throws Exception
void removePathTime(String fromRef, String toRef) throws Exception
Collection<String> getPathTimes() throws Exception
```

6.4 Set methods

```
void setRequired(int routeId, double value, String units) throws Exception
void setFunctionTime(int routeId, String refString, double value,
String units) throws Exception
void setPathTime(int routeId, String fromRef, String toRef, double value,
String units) throws Exception
void setInstructionTime(int routeId, String refString, double value,
String units) throws Exception
void setLoop(int routeId, String refString, long iterations)
throws Exception
void setLoopPath(int routeId, String refString, long iterations)
throws Exception
void setLoopScope(int routeId, String refString, boolean absolute)
throws Exception
void setExclusion(int routeId, String refString) throws Exception
```

6.5 Get methods

```
double getRequired(int routeId, String units) throws Exception
double getWorstCase(int routeId, String units) throws Exception
double getBestCase(int routeId, String units) throws Exception
List<String> getWarnings(int routeId) throws Exception
List<String> getErrors(int routeId) throws Exception
List<Integer> getRouteIds() String getRouteDescription(int routeId)
throws Exception
```

6.6 Config methods

```
void configCores(String tileReference, int numCores) throws Exception
void configFreq(String nodeId, double tileFrequency) throws Exception
```

7 Code reference grammar

A code reference constructed of a *back trail* and a base reference of the form:

code-ref ::= *back-trail base-ref*

back-trail ::= *base-ref*
| *base-ref , back-trail*

base-ref ::= *pc-ref*
| *label-ref*
| *function-ref*
| *endpoint-ref*
| *srclabel-ref*
| *call-ref*

pc-ref ::= *pc-class hex-constant*

label-ref ::= *label-class label-string*

function-ref ::= *function-class function-name*
| *function-class hex-constant*

endpoint-ref ::= *endpoint-class file-line*
| *endpoint-class endpoint-label*
| *endpoint-class hex-constant*

srclabel-ref ::= *srclabel-class label-string*

call-ref ::= *call-class label-string*
| *call-class hex-constant*

pc-class ::=
| *PC:*

label-class ::=
| *LABEL:*

functionclass ::=
| *FUNCTION:*

endpointclass ::=
| *ENDPOINT:*

srclabelclass ::=
| *SRCLABEL:*

call-class ::=
| *CALL:*

file-line ::= *file-name : integer-constant*