**Simulinkモデル構造と生成コード**

**および**

**その処理量との対応**

**使用説明書**

株式会社NEC情報システムズ

2016年3月3日

**目次**

[1. はじめに 2](#_Toc444763061)

[2. ソフトウェア構成 3](#_Toc444763062)

[2.1. 実行環境 4](#_Toc444763063)

[2.2. ファイル一覧 6](#_Toc444763064)

[3. ビルド手順 7](#_Toc444763065)

[3.1. コンパイル 7](#_Toc444763066)

[3.2. サンプルの実行 7](#_Toc444763067)

[4. サンプルコード使用方法 8](#_Toc444763068)

[4.1. rtw\_testコマンド仕様 8](#_Toc444763069)

[4.2. xml\_testコマンド仕様 8](#_Toc444763070)

[4.3. xsd\_driverコマンド仕様 9](#_Toc444763071)

[4.4. add\_block\_infoコマンド仕様 9](#_Toc444763072)

[4.5. dump\_block\_listコマンド仕様 9](#_Toc444763073)

[5. 性能情報付きXML機能の使用方法 10](#_Toc444763074)

[5.1. blxml-perf.sh 10](#_Toc444763075)

[5.2. blxml2code 11](#_Toc444763076)

[5.3. opt(llvm-BLXMLperf.so) 12](#_Toc444763077)

[6. ファイル形式 13](#_Toc444763078)

[6.1. XMLファイル 13](#_Toc444763079)

[6.2. SHIM 13](#_Toc444763080)

[6.3. DOT 13](#_Toc444763081)

[6.3.1. xml\_testコマンドのDOTファイル 13](#_Toc444763082)

[6.3.2. opt(llvm-BLXMLPerf.so)のDOTファイル 14](#_Toc444763083)

[6.3.2.1. ノード 14](#_Toc444763084)

[6.3.2.2. エッジ 14](#_Toc444763085)

[6.4. ブロック情報CSVファイル 15](#_Toc444763086)

[7. モジュール仕様 16](#_Toc444763087)

[7.1. SimulinkModel(データバインディング) 16](#_Toc444763088)

[7.1.1. タグと型(クラス、イテレータ)の対応 17](#_Toc444763089)

[7.1.2. XML入力関数 21](#_Toc444763090)

[7.1.3. XML出力関数 21](#_Toc444763091)

[7.2. SimulinkXML(SimulinkModelの支援モジュール) 22](#_Toc444763092)

[7.2.1. Readerクラス 31](#_Toc444763093)

[7.2.1.1. Readerクラスコンストラクタ 31](#_Toc444763094)

[7.2.1.2. load\_xml() 31](#_Toc444763095)

[7.2.1.3. find\_input\_port() 32](#_Toc444763096)

[7.2.1.4. find\_output\_port() 32](#_Toc444763097)

[7.2.1.5. find\_inport\_input() 32](#_Toc444763098)

[7.2.1.6. find\_outport\_output() 32](#_Toc444763099)

[7.2.1.7. find\_block() 33](#_Toc444763100)

[7.2.1.8. xml\_blocks() 33](#_Toc444763101)

[7.2.1.9. block\_map() 33](#_Toc444763102)

[7.2.1.10. input\_map() 34](#_Toc444763103)

[7.2.1.11. output\_map() 34](#_Toc444763104)

[7.2.1.12. block\_list() 34](#_Toc444763105)

[7.2.1.13. find\_loop() 35](#_Toc444763106)

[7.2.1.14. dump() 35](#_Toc444763107)

[7.2.1.15. キャスト(bool) 35](#_Toc444763108)

[7.2.2. BlockInfoクラス 36](#_Toc444763109)

[7.2.2.1. BlockInfoクラスコンストラクタ 36](#_Toc444763110)

[7.2.2.2. id() 36](#_Toc444763111)

[7.2.2.3. block() 36](#_Toc444763112)

[7.2.2.4. upper() 36](#_Toc444763113)

[7.2.2.5. is\_subsystem() 37](#_Toc444763114)

[7.2.2.6. has\_update() 37](#_Toc444763115)

[7.2.2.7. is\_sfunction() 37](#_Toc444763116)

[7.2.2.8. subsystem\_type() 37](#_Toc444763117)

[7.2.2.9. upper\_subsystem() 37](#_Toc444763118)

[7.2.2.10. lower\_block() 38](#_Toc444763119)

[7.2.2.11. forward() 38](#_Toc444763120)

[7.2.2.12. backward() 38](#_Toc444763121)

[7.2.2.13. top\_subsystem() 38](#_Toc444763122)

[7.2.2.14. top() 39](#_Toc444763123)

[7.2.2.15. contain\_block() 39](#_Toc444763124)

[7.2.2.16. is\_contained() 39](#_Toc444763125)

[7.2.2.17. merged\_to() 39](#_Toc444763126)

[7.2.2.18. merge\_blocks() 39](#_Toc444763127)

[7.2.2.19. bound\_to() 40](#_Toc444763128)

[7.2.2.20. bind\_blocks() 40](#_Toc444763129)

[7.2.2.21. trigger\_port() 40](#_Toc444763130)

[7.2.2.22. enable\_port() 40](#_Toc444763131)

[7.2.2.23. action\_port() 40](#_Toc444763132)

[7.2.2.24. is\_atomic\_subsystem() 41](#_Toc444763133)

[7.2.2.25. dump() 41](#_Toc444763134)

[7.2.2.26. オペレータ(==) 41](#_Toc444763135)

[7.2.2.27. 静的メンバmax\_id() 41](#_Toc444763136)

[7.2.3. IOPortクラス 42](#_Toc444763137)

[7.2.3.1. IOPortクラスコンストラクタ 42](#_Toc444763138)

[7.2.3.2. block\_tag() 42](#_Toc444763139)

[7.2.3.3. ioport\_tag() 42](#_Toc444763140)

[7.2.3.4. blockname() 42](#_Toc444763141)

[7.2.3.5. port() 42](#_Toc444763142)

[7.2.3.6. line() 43](#_Toc444763143)

[7.2.3.7. match() 43](#_Toc444763144)

[7.2.3.8. dump() 43](#_Toc444763145)

[7.2.4. BlockConnectクラス 44](#_Toc444763146)

[7.2.4.1. BlockConnectクラスコンストラクタ 44](#_Toc444763147)

[7.2.4.2. var\_list() 44](#_Toc444763148)

[7.2.4.3. head() 44](#_Toc444763149)

[7.2.4.4. tail() 44](#_Toc444763150)

[7.2.4.5. オペレータ(==) 45](#_Toc444763151)

[7.2.5. codelistクラス 46](#_Toc444763152)

[7.2.5.1. codelistクラスコンストラクタ 47](#_Toc444763153)

[7.2.5.2. process() 47](#_Toc444763154)

[7.2.5.3. mark\_list() 47](#_Toc444763155)

[7.2.6. c\_markクラス 48](#_Toc444763156)

[7.2.6.1. c\_markクラスコンストラクタ 48](#_Toc444763157)

[7.2.6.2. set() 48](#_Toc444763158)

[7.2.6.3. text\_beg() 48](#_Toc444763159)

[7.2.6.4. text\_end() 48](#_Toc444763160)

[7.2.6.5. beg() 48](#_Toc444763161)

[7.2.6.6. end() 49](#_Toc444763162)

[7.2.6.7. type() 49](#_Toc444763163)

[7.2.6.8. num() 49](#_Toc444763164)

[7.2.6.9. name() 49](#_Toc444763165)

[7.2.6.10. キャスト(string) 49](#_Toc444763166)

[7.2.7. const\_block\_list\_Tクラス 50](#_Toc444763167)

[7.2.7.1. const\_block\_list\_Tクラスコンストラクタ 50](#_Toc444763168)

[7.2.7.2. checkinfo() 50](#_Toc444763169)

[7.2.7.3. any() 50](#_Toc444763170)

[7.2.7.4. addinfo() 50](#_Toc444763171)

[7.2.7.5. オペレータ(+=) 50](#_Toc444763172)

[7.2.8. 関数 51](#_Toc444763173)

[7.2.8.1. find\_multirate\_subsystem() 51](#_Toc444763174)

[7.3. SimulinkRTW(SimulinkモデルとRTW-ECのコード処理) 52](#_Toc444763175)

[7.3.1.1. 入力処理関数 52](#_Toc444763176)

[7.3.1.2. XML出力 53](#_Toc444763177)

[7.3.1.3. 使用例 54](#_Toc444763178)

[7.4. blxml2code 55](#_Toc444763179)

[7.4.1.1. block\_attrクラスコンストラクタ 57](#_Toc444763180)

[7.4.1.2. block\_attr ::id() 57](#_Toc444763181)

[7.4.1.3. block\_attr ::block() 57](#_Toc444763182)

[7.4.1.4. block\_attr ::ignore() 57](#_Toc444763183)

[7.4.1.5. block\_attr ::set\_ignore() 57](#_Toc444763184)

[7.4.1.6. block\_attr ::lowers() 58](#_Toc444763185)

[7.4.1.7. block\_attr ::merged 58](#_Toc444763186)

[7.4.1.8. キャスト(bool) 58](#_Toc444763187)

[7.4.1.9. attr\_map ::get 58](#_Toc444763188)

[7.4.1.10. attr\_map::get\_id 58](#_Toc444763189)

[7.4.1.11. attr\_map::add() 59](#_Toc444763190)

[7.4.1.12. BLXMLクラスコンストラクタ 59](#_Toc444763191)

[7.4.1.13. BLXML::load() 59](#_Toc444763192)

[7.4.1.14. BLXML::gencode() 59](#_Toc444763193)

[7.4.1.15. BLXML::check\_switchcase\_if\_block() 59](#_Toc444763194)

[7.4.1.16. BLXML::dump() 59](#_Toc444763195)

[7.4.1.17. キャスト(bool) 60](#_Toc444763196)

[7.5. llvm-BLXMLPerf 61](#_Toc444763197)

[7.5.1.1. FunctionInfoクラスコンストラクタ 62](#_Toc444763198)

[7.5.1.2. FunctionInfo ::name() 62](#_Toc444763199)

[7.5.1.3. FunctionInfo ::best() 62](#_Toc444763200)

[7.5.1.4. FunctionInfo ::typical() 62](#_Toc444763201)

[7.5.1.5. FunctionInfo ::worst() 62](#_Toc444763202)

[7.5.1.6. BLXMLPerfクラスコンストラクタ 63](#_Toc444763203)

[7.5.1.7. BLXMLPerf ::doInitialization() 63](#_Toc444763204)

[7.5.1.8. BLXMLPerf ::doFinalization() 63](#_Toc444763205)

[7.5.1.9. BLXMLPerf ::runOnFunction() 63](#_Toc444763206)

[7.5.1.10. BLXMLPerf ::getAnalysusUsage() 63](#_Toc444763207)

[7.5.2. boost\_llvm 64](#_Toc444763208)

[7.5.3. shim\_llvm 66](#_Toc444763209)

[7.5.3.1. Latencyクラスコンストラクタ 67](#_Toc444763210)

[7.5.3.2. Latency ::best() 67](#_Toc444763211)

[7.5.3.3. Latency ::typical() 67](#_Toc444763212)

[7.5.3.4. Latency ::worst() 67](#_Toc444763213)

[7.5.3.5. Fileクラスコンストラクタ 67](#_Toc444763214)

[7.5.3.6. File ::latencies() 68](#_Toc444763215)

[7.5.3.7. File ::latency() 68](#_Toc444763216)

[7.5.3.8. File ::ComponentSet() 68](#_Toc444763217)

[7.5.3.9. File ::MasterComponent() 68](#_Toc444763218)

[7.5.3.10. File ::CommonInstructionSet() 68](#_Toc444763219)

[7.5.3.11. キャスト(bool) 68](#_Toc444763220)

[7.5.3.12. キャスト(::SHIM::SysmtemConfiguration\*) 68](#_Toc444763221)

[7.5.4. blxml\_llvm 69](#_Toc444763222)

[7.5.4.1. BLXMLクラスコンストラクタ 70](#_Toc444763223)

[7.5.4.2. set\_performance() 70](#_Toc444763224)

[7.5.4.3. set\_filename() 70](#_Toc444763225)

[7.5.4.4. set\_shim\_param() 70](#_Toc444763226)

[7.5.4.5. output() 70](#_Toc444763227)

[7.5.4.6. キャスト(bool) 70](#_Toc444763228)

[7.5.4.7. キャスト(blocks\_T\*) 71](#_Toc444763229)

[7.5.5. Passの処理概要 71](#_Toc444763230)

[7.5.5.1. 関数単位の処理 71](#_Toc444763231)

[7.5.5.2. ループの処理 71](#_Toc444763232)

[7.5.5.3. ループ内の基本ブロック 72](#_Toc444763233)

[8. 制限事項と課題 73](#_Toc444763234)

[8.1. 制限事項 73](#_Toc444763235)

[8.2. 課題 75](#_Toc444763236)

**図目次**

[図 2‑1 プログラムと入出力ファイル 4](#_Toc419367191)

[図 2‑2　ディレクトリ構成 6](#_Toc419367192)

**表目次**

[表 2‑1 ソフトウェア構成 3](#_Toc444763237)

[表 2‑2 ツール・ライブラリ一覧 4](#_Toc444763238)

[表 2‑3 ツールのバージョン 5](#_Toc444763239)

[表 5‑1 blxml-perf.shのオプション 10](#_Toc444763240)

[表 5‑2 blxml2codeのオプション 11](#_Toc444763241)

[表 5‑3 opt(llvm-BLXMLperf.so)のオプション 12](#_Toc444763242)

[表 6‑1 CSVファイルのフィールド 15](#_Toc444763243)

[表 7‑1タグと型の一覧 17](#_Toc444763244)

[表 7‑2タグと属性の一覧 19](#_Toc444763245)

[表 7‑3 SimulinkXMLモジュールのクラス一覧 22](#_Toc444763246)

[表 7‑4 SimulinkXMLモジュールの型一覧 30](#_Toc444763247)

[表 7‑5 SimulinkXMLモジュールの関数の一覧 30](#_Toc444763248)

[表 7‑6 SimulinkRTW入力処理関数引数一覧 52](#_Toc444763249)

[表 7‑7 SimulinkRTW XML出力関数引数一覧 53](#_Toc444763250)

[表 7‑8 blxm2codeのクラス一覧 55](#_Toc444763251)

[表 7‑9 blxml2codeの型一覧 56](#_Toc444763252)

[表 7‑10 llvm-BLXMLPerfのクラス一覧 61](#_Toc444763253)

[表 7‑11 llvm-BLXMLPerfの型一覧 61](#_Toc444763254)

[表 7‑12 namespace boostの型追加一覧 64](#_Toc444763255)

[表 7‑13 boost\_llvmの型一覧 64](#_Toc444763256)

[表 7‑14 shim\_llvmのクラス一覧 66](#_Toc444763257)

[表 7‑15 shim\_llvmの型一覧 66](#_Toc444763258)

[表 7‑16 blxml\_llvmのクラス一覧 69](#_Toc444763259)

[表 7‑17 blxml\_llvmの型一覧 69](#_Toc444763260)

**参考資料**

|  |  |  |  |
| --- | --- | --- | --- |
| 項番 | 仕様書名 | 版数 | 発行元 |
|  |  |  |  |
|  |  |  |  |

**用語一覧**

|  |  |  |
| --- | --- | --- |
| 項番 | 用語 | 意味 |
| 1 | ブロックレベル構造XML | Simulinkモデルから抽出したブロック構造を持つXMLファイル(名大様提供) |
| 2 | CSPトランスレータ | トヨタ様と名大様の共同研究の中でトヨタ様のモデルをターゲット機で実行可能なコードに変換するプログラム |
| 3 | SHIM | Software-Hardware Interface for Multi-Many Coreの略で、マルチコア・チップの属性情報を記述する標準化されたインターフェース仕様 |
| 4 | RTW-EC | RTWはMathWorks社のRealTime Workshopの略で、ECはコード生成を行うEmbedded Codeの略 |
| 5 | LLVM | LLVMはllvm.orgで開発しているコンパイラ基盤 |

# はじめに

本書では、Masworks社MATLAB/Simulinkモデル構造と生成コードとの対応を行うモジュールと、モデルのブロックのコードの性能情報をSHIMから入力した情報に基づき計測、XMLに反映するLLVMのpassの使い方について説明します。

# ソフトウェア構成

　ソフトウェア構成は以下の通りです

表 2‑1 ソフトウェア構成

|  |  |  |
| --- | --- | --- |
| 項番 | 名称 | 説明 |
| 1 | SimulinkModel.xsd | XMLスキーマ。データバインディングのツールを使用し、本スキーマに従ったXMLを取り扱うことができるSimulinkModelのC++コードを生成する |
| 2 | SimulinkXML | SimulinkモデルのXMLの利用を支援するモジュール |
| 3 | SimulinkRTW | RTW-ECのコードを扱うモジュール |
| 4 | xml\_test | rtw\_testが生成したXMLを入力し、依存関係をgraphviz/dot形式で出力するプログラム |
| 5 | rtw\_test | Simulinkモデルのブロック構造とRTW-ECの生成コードを入力し、対応付けを行ったXMLを出力するプログラム |
| 6 | blxml2code | ブロックレベル構造XMLからSimulinkモデルのブロック単位でコンパイル可能なコード生成を行うプログラム |
| 7 | llvm-BLXMLperf.so | blxm2codeで生成されたコードの性能を計測し、性能情報を付加したブロックレベル構造XMLを出力するLLVMのpass |
| 8 | blxml-perf..sh | blxml2codeおよびllvm-BLXMLperf.soを適切に実行する一連の処理を実行するshell script |
| 9 | add\_block\_info | ブロックレベル構造XMLに、コア割り当てとプロセス集約の指示を埋め込むプログラム |
| 10 | dump\_block\_list | ブロックレベル構造XMLに含まれるブロック名とブロックIDを出力するプログラム |

* 名古屋大学様とトヨタ自動車株式会社様の共同研究に関連し、株式会社NEC情報システムズが受注しているCSPトランスレータとの関連を考慮し、Cと連携できるC++を開発言語とします
* XMLのデータバインディングのツールはCodesynthesis XSDを用います。
* xml\_testはSimulinkXMLモジュールの使い方を示すサンプルコードで、実行コードを持つブロックの接続関係をグラフ化します
* rtw\_testは完全な機能を持っていますが、SimulinkRTWモジュールの使い方を示すサンプルコードです。

XML

rtw\_test

コード付き

XML

C(Source)

C(header)

C(data)

blxml2code

ブロック

Cコード

Clang

性能情報付

XML

SHIM

LLVM

アセンブラ

opt(LLVM-BLXMperf.so)

blxml-perf.sh

図 2‑1 プログラムと入出力ファイル

## 実行環境

　本プログラムはUnix上で動作します。コンパイル・実行にあたり、以下のツール・ライブラリが必要です。なお、広く使われているものであり、Linuxであれば、各種ディストリビューションでパッケージ化されているため、容易に導入することができます。

表 2‑2 ツール・ライブラリ一覧

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 名称 | 説明 | 入手先 |
| 1 | Gnu C/C++ | C/C++コンパイラ | http://gcc.gnu.org/ |
| 2 | Codesynthesis XSD | データバインディングツール | http://codesynthesis.com/products/xsd/ |
| 3 | Xerces-C++ | XMLパーザー | http://xerces.apache.org/xerces-c/ |
| 4 | Boostライブラリ | C++オープンソースライブラリ | http://www.boost.org/ |
| 5 | graphviz | グラフ描画ライブラリ | http://graphviz.org/ |
| 6 | Doxygen | ドキュメント生成ツール | http://www.doxygen.jp/ |
| 7 | LLVM | コンパイラ基盤 | http://llvm.org/ |
| 8 | Clang | C/C++コンパイラ | http://clang.llvm.org/ |

なお、実行はFedora 22(x64)およびCentOS6.6, CentOS7.0, Solaris11.2(x64)で、32bitアプリケーションとして確認しています。それぞれの確認したバージョンは以下の通りです。

表 2‑3 ツールのバージョン

|  |  |  |
| --- | --- | --- |
| No. | 名称 | バージョン番号 |
| 1 | GNU C/C++ | 4.8.1 / 4.8.2 / 4.9.0 / 4.9.1 / 5.1.0 |
| 2 | Codesynthesis XSD | 3.3.0 / 4.0.0 |
| 3 | Xerces-C++ | 3.1.1 |
| 4 | Boostライブラリ | 1.53.0 / 1.55.0 / 1.58.0 |
| 5 | graphviz | 2.28.0 / 2.30.1 |
| 6 | Doxygen | 1.7.6.1 / 1.8.3.1 |
| 7 | Clang/LLVM | 3.4 / 3.4.1 / 3.4.2 / 3.5.0  3.6.0 / 3.6.1 / 3.6.2 |

## ファイル一覧

　以下はディレクトリ構成とファイルの一覧です

Simulink\_XSD

├ Makefile makefile

├ README.txt 概要メモ

├ README\_BLXMLperf.txt 性能情報付きBLXMLに関する概要メモ

├ SHIM.xml 性能情報取得用のSHIMのサンプル

├ SimulinkModel.cxx XSDにより生成されたC++ソース

├ SimulinkModel.doxygen doxygen設定ファイル

├ SimulinkModel.hxx XSDにより生成されたC++ヘッダ

├ SimulinkModel.m4 XMLスキーマの生成用m4マクロソース

├ SimulinkRTW.cxx SimulinkRTWモジュール(ソース)

├ SimulinkRTW.h SimulinkRTWモジュール(ヘッダ)

├ SimulinkXML.cxx SimulinkXMLモジュール(ソース)

├ SimulinkXML.h SimulinkXMLモジュール(ヘッダ)

├ SimulinkXSD.xlsx スキーマの説明(Excelファイル)

├ add\_block\_info.cxx add\_block\_infoプログラム

├ blxml-perf.sh 性能情報付きBLXML用実行スクリプト

├ blxml2code.cxx 性能情報計測用コード生成用ソース

├ blxml\_llvm.cxx LLVM用BLXML処理モジュール(ソース)

├ blxml\_llvm.h LLVM用BLXML処理モジュール(ヘッダ)

├ boost\_llvm.cxx LLVM用Boostライブラリ処理モジュール

├ dump\_block\_info.cxx dump\_block\_infoプログラム

├ llvm-BLXMLPerf.cxx BLXML処理用LLVMのパス本体(ソース)

├ llvm-BLXMLPerf.h BLXML処理用LLVMのパス本体(ヘッダ)

├ rtw\_test.cxx rtw\_testサンプルプログラム

├ shim.xsd SHIMのXMLスキーマ

├ shim\_llvm.cxx LLVM用SHIM処理モジュール(ソース)

├ shim\_llvm.h LLVM用SHIM処理モジュール(ヘッダ)

├ xml\_test.cxx xml\_testサンプルプログラム

├ xsd\_driver.cxx SimulinkModelテストコード

├ models

│ ├ pid テスト用入力・出力コード

│ ├ test1 テスト用入力・出力コード

│ └ testmux テスト用入力・出力コード

├ models2

│ ├ Mfunc テスト用入力・出力コード

│ └ cell テスト用入力・出力コード

├ temp テスト用入力・出力コード

└ 0911 テスト用入力・出力コード

図 2‑2　ディレクトリ構成

* SimulinkModel.xsdはSimulinkModel.m4から生成され、Simulinkのブロック構造とRTW-ECから抽出した変数、状態変数、パラメタ、定数パラメタ、初期化コード、実行コード等を保持するためのXMLスキーマで、その説明はSimulinkXSD.xlsxを参照してください。
* このスキーマはCodesynthesis XSDにより、SimulinkModel.cxxおよびSimulinkModel.hxxファイルが生成され、C++から利用可能になります。利用に際してはApache XMLプロジェクトのXerces-C++ライブラリが必要です
* SimulinkModel.doxygenは、Codesynthesis XSDが生成するC++コード中の記述からドキュメントを生成するために用います。
* shim.xsdは2015年2月公開のSHIMスキーマです。

# ビルド手順

## コンパイル

Simulink\_XSDのディレクトリにて以下のようにmakeを実行してください。

$ make

## サンプルの実行

makeコマンドによりサンプルデータを処理することができます。以下のように実行するとsynrmmodel\_mt\_RTWの処理を行います。

$ make test

実行することは以下の通りです。

1. rtw\_testによるXMLの作成
2. xsd\_driverによるSimulinkModelのスキーマの動作確認
3. xml\_testによるgraphvizのdot形式ファイルとPNG形式の画像ファイルの生成
4. blxm2codeによるコード断片のコンパイル可能なコード生成と、LLVMのpassであるLLVM-BLXMLperfによる性能情報付きXML出力

なお、サンプルデータとしては、以下のものを準備する必要があります。

* 2014/9/25ご提供のsynrmmodel\_mt\_RTW.zipを0925以下に展開
* 2014/9/11ご提供のsynrmmodel\_mt\_RTW\_ert\_rtw-140911.zipを0911以下に展開
* 2014/7/1ご提供のtemp.zipをtemp以下に展開
* 2014/4/29ご提供のmodels2.zipをmodels2以下に展開
* 2013/11/29ご提供のmodels.zipをディレクトリmodels配下に展開

マクロTEST\_TARGETを変更することで、test1, testmux等の処理に切り替えられます。例えばtestmuxは以下のように行います。

$ make TEST\_TARGET=testmux test

なおcellについては、XMLが不正であることと、RTW-ECのコードに不一致が見られ、処理できませんのでご注意ください。

詳細についてはMakefileを参照してください。

# サンプルコード使用方法

## rtw\_testコマンド仕様

$ rtw\_test [-n] [-P{x|a}] –c Cコード –h ヘッダ –d パラメタ初期値 [入力XML] [出力XML]

RTW-ECが生成したCコード、ヘッダファイル、パラメタ・定数パラメタの初期値のファイルと、Simulinkモデルのブロック構造の入力XMLファイル、処理結果を出力する出力XMLファイルを指定します。

入力XMLファイルが指定されない場合、標準入力から入力します。

出力XMLファイルが指定されない場合、標準出力へ出力します。

実行時、入力XMLファイルと同じディレクトリにXMLスキーマSimulinkModel.xsdが必要です (XMLファイル入力時にXMLスキーマに基づきフォーマットのチェックが行われます) 。なお、-nオプションを指定した場合このチェックは行いません。

-Pオプションは、Cコードの読み込み動作を変更します。-Pxオプションは、Triggered/Enabled SubSystem内部を分割しBLXMLの各ブロックにコードを反映します。-Paオプションは、同様にAtomic SubSystemの内部を分割しBLXMLの各ブロックにコードを反映します。オプションを指定しない場合のデフォルト動作は、これらのSubSystemに対しては内部のブロックを含んだ一括りのコードとしてSubSystemのコードになります。

## xml\_testコマンド仕様

$ xmltest [-n] [入力XML] [出力DOT]

rtw\_testが生成したXMLファイルを入力し、graphvizのdot形式のファイルを出力します。描画される内容は、<forward>タグを持つブロックおよびその<forward>タグの情報に基づき、実行文があるブロックと、入出力される変数となります。dot形式のファイルはgraphvizのコマンドにより、画像ファイルに変換できます。

実行時、入力XMLファイルと同じディレクトリにXMLスキーマSimulinkModel.xsdが必要です(XMLファイル入力時にXMLスキーマに基づきフォーマットのチェックが行われます)。なお、-nオプションを指定した場合このチェックは行いません。

## xsd\_driverコマンド仕様

$ xsd\_driver [入力XML]

Simulinkモデルのブロック構造の入力XMLファイル、またはrtw\_testが生成したXMLファイルを入力し、成功した場合、内容の一部を標準出力に出力します。これはXMLとスキーマSimulinkModel.xsdとの整合性確認のためのプログラムです。

## add\_block\_infoコマンド仕様

$ add\_block\_info –i 入力XML [-o 出力XML] [CSVファイル]

Simulinkモデルのブロック構造のXMLファイルを入力し、CSVファイルの情報に従って、Simulinkモデルのブロックに、情報を付加します。-oオプションを省略した場合、標準出力にXMLファイルを出力します。CSVファイルを省略した場合、標準入力から読み込みます。

CSVファイルのフォーマットは6.4ブロック情報CSVファイルを参照してください。

## dump\_block\_listコマンド仕様

$ dump\_block\_list [-o 出力CSVファイル] [入力XML]

Simulinkモデルのブロック構造のXMLファイルを入力し、含まれるブロック名とブロックIDをCSVファイルに出力します。-oオプションを省略した場合、標準出力にCSVファイルを出力します。入力XMLファイルを省略した場合、標準入力から読み込みます。

このプログラムにより、4.4add\_block\_infoコマンド仕様で使用するCSVファイルのひな形を生成することが可能です。

CSVファイルのフォーマットは6.4ブロック情報CSVファイルを参照してください。

# 性能情報付きXML機能の使用方法

本機能はrtw\_testが生成した、RTW-ECの生成コードからSimulinkブロックのコードを抽出、保持しているXMLファイルを入力し、SHIMの性能情報を元に、コードの性能情報を計測し、その結果を追加したXMLファイルを生成する機能です。この機能は、Simulinkブロック単位のコードを生成するblxml2codeと、LLVMのoptに追加したPassにより実施される性能計測、およびそれら一連の動作を実行するshell scriptからなります。

## blxml-perf.sh

$ blxml-perf.sh [-n] [-v] [-o output] [-shim=SHIM] [-shim-schema=SCHEMA]

[-C clang-option] [-0 pass-option] [-S pass-module] INPUT-XML

rtw\_testが生成したブロックレベル構造XMLファイル(INPUT-XML)と指定されたSHIMを入力し、性能情報を付加したXMLファイルを生成します。オプションは以下の通りです。

表 5‑1 blxml-perf.shのオプション

|  |  |  |
| --- | --- | --- |
| オプション | 有無 | 説明 |
| -n | 省略可 | BLXMLスキーマのチェックを行いません |
| -v | 省略可 | verboseオプション。指定した場合、冗長なメッセージを出力します |
| -o output | 省略可 | 出力XMLファイル名を指定します。指定しない場合、出力先は標準出力です |
| -shim=SHIM | 省略可 | SHIMのファイル名(SHIM)を指定します。指定しない場合、全ての命令のサイクル数を1とします。 |
| -shim-schema=SCHEMA | 省略可 | SHIMのスキーマ(SCHEMA)を指定します。指定しない場合、SHIMのスキーマのチェックを行いません |
| -C clang-option | 省略可 | clangに渡すオプションを指定します |
| -O pass-option | 省略可 | LLVMのoptに渡すオプションを指定します。オプションの詳細は表 5‑3 opt(llvm-BLXMLperf.so)のオプションを参照してください |
| -S pass-module | 省略可 | LLVMのoptで実行するPassのモジュールを指定します。指定しない場合は./llvm-BLXMLPerf.soになります。 |

例えばpidでは以下のように実行します。

$ blxml-perf.sh -v -o models/pid/test\_blxml.xml \

-shim=SHIM.xml \

-C -Imodels/pid/pid\_controller1\_ert\_rtw -C -O \

models/pid/yy.xml

## blxml2code

$ blxml2code [-n] [-v] [-o output] [-N] [-H user-header] [BLXML]

rtw\_testが生成したブロックレベル構造XMLファイル(BLXML)からブロック毎のtask/init/updateをそれぞれ関数化したCコードを生成します。XMLを指定しない場合、標準入力から入力します。ブロックの入出力変数は関数のパラメタとして出力されます。オプションは以下の通りです。

表 5‑2 blxml2codeのオプション

|  |  |  |
| --- | --- | --- |
| オプション | 有無 | 説明 |
| -n | 省略可 | BLXMLスキーマのチェックを行いません |
| -v | 省略可 | verboseオプション。指定した場合、冗長なメッセージを出力します |
| -o output | 省略可 | 出力Cコードのファイル名を指定します。指定しない場合、出力先は標準出力です |
| -N | 省略可 | 入出力変数を関数のパラメタとして出力しません。入出力変数は大域変数となります |
| -H user-header | 省略可 | 利用者が作成したヘッダファイルを指定します。出力Cコード中にそのヘッダファイルをインクルードするようコード生成を行います |

生成したCコードのコンパイルにはRTW-ECのヘッダファイルが必要です。インクルードして使用するヘッダファイルについては、生成したCコードのincludeディレクティブで指定されているファイルを確認してください。

また、性能計測の際には、clangに指定する-Iオプション等の指定が必要になりますので注意してください。clangでコンパイルする時は以下のように、一旦LLVMアセンブラを出力します。

$ clang -S -O -I<path fof rtwtype.h> -emit-llvm -o out.ll input.c

## opt(llvm-BLXMLperf.so)

$ opt -load=<path for llvm-BLXMLPerf.so> -S -blxml-perf \

[-shim=SHIM] [-shim-schema=SCHEMA] \

[-MasterComponent=name] [-CommonInstructionSet=name] \

[-no-ret] [-no-br] \

[-in-xml=INPUT.XML] [-out-xml=OUTPUT.XML] [-no-blxml-validation]

-o dummy\_output.ll input.ll

blxml2codeが生成したCコードをclangでコンパイルし、出力したLLVMアセンブラコードを入力し、SHIMのCommonInstructionSetで指定される命令のサイクル数を元に、各Simulinkブロックのコードの性能情報を計測し、その情報を付加したブロックレベル構造XMLファイルを出力します。オプションは以下の通りです。

表 5‑3 opt(llvm-BLXMLperf.so)のオプション

|  |  |  |
| --- | --- | --- |
| オプション | 有無 | 説明 |
| -shim=SHIMファイル | 必須 | SHIMのファイル名を指定します。noを指定した場合、全ての命令のサイクル数を1とし、実行時に命令毎にSHIM情報がない旨のエラーメッセージを出力します |
| -shim-schema=SCHEMAファイル | 省略可 | SHIMのスキーマを指定します。指定しない場合、SHIMのチェック(validation)を行いません |
| -MasterComponent=name | 省略可 | 処理対象のMasterComponentの名前を指定します。省略した場合は、-CommonInstructionSetのオプション指定ににより処理対象が決定します。 |
| -CommonInstructionSet=name | 省略可 | 処理対象のCommonInstructionSetの名前を指定します。省略した場合、最初に見つけたCommonInstructionSetを使用します |
| -no-ret | 省略可 | ret命令を無視します(ブロックのCコードは関数として生成するためret命令を含みます) |
| -no-br | 省略可 | 基本ブロック間の分岐命令を無視します |
| -in-xml=INPUT.XML | 必須 | 入力するブロックレベル構造XMLを指定します。XMLはblxml2codeでコード生成に使用したXMLでなければならず、省略できません |
| -out-xml=OUTPUT.XML | 省略可 | 出力する性能付きブロックレベル構造XMLを指定します。省略した場合は標準出力に出力します |
| -out-dot=basename | 省略可 | このオプションを指定した場合、basename-<番号>.dot というGraphVizのファイルがループおよび関数毎に順に生成します |
| -no-blxml-validation | 省略可 | BLXMLスキーマのチェックを行いません |

# ファイル形式

## XMLファイル

SimulinkモデルのXMLファイルは、データバインディング機能を利用するため、名前空間(namespace)で指定するグローバルなタグである<blocks>を使用する必要があります。

そのため、一番最初に出現する<blocks>タグは以下のように記述する必要があります。

<sm:blocks

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://example.com/SimulinkModel SimulinkModel.xsd"

xmlns:sm="http://example.com/SimulinkModel"

name="pid\_controller1"> // これは pid 用の名前です

これにより名前空間xsiはw3のXMLスキーマ、名前空間smは、Simulinkモデルのスキーマに対応することになります。

二度目以降の<blocks>にも、名前空間smを指定する必要があります。

<sm:blocks>

そのほかのタグには、名前空間を指定する必要はありません。

modelsディレクトリ配下にあるxx.xmlファイルは、ご提供いただいたXMLファイルを上記のように修正を行ったものです。

Simulinkモデルで使用するXMLタグについては、SimulinkXSD.xlsxを参照してください。

## SHIM

shim.xsdに準拠し、CommonInstructionSetに命令サイクル数が格納されたSHIMを準備する必要があります。

## DOT

GraphvizのDOT形式のファイルです。xml\_testコマンドではモデルのブロックの相関関係として、opt(llvm-BLXMLPerf.so)が、性能情報を計測した際の経路情報として出力します。関数単位の処理としては、最内ループから順に出力し、最後にコード全体(関数)のグラフを出力します。

### xml\_testコマンドのDOTファイル

ノードは、Simulinkのブロックで、エッジはブロック間の接続を示します。

### opt(llvm-BLXMLPerf.so)のDOTファイル

#### ノード

ノードは基本ブロックまたはまとめられた固定回回るループです。属性は以下のようになっています。

* 赤枠(red)かつ塗りつぶされているノード(plum)は、固定回回るループを一つにまとめたもの
* 青枠(blue)のノードは、最大回数がわかっているループのヘッド(基本ブロック)
* それ以外のノードは通常の基本ブロック
* boxのノードは、ループのヘッドへのbackedgeを示すためのダミーノード
* 内部の数字は上から順に、そのノードのbest/typical/worstの処理サイクル
* 最後の[]内の数字は、終端までの平均処理時間です(successorの各処理時間と等確率分岐によって平均化した処理時間)

#### エッジ

エッジはsuccessorになります。

* 緑(green)は最短経路
* 青(blue)は最長経路。最長経路が最短経路と重なる場合は、最短経路の表示が優先
* 赤(red)の破線はループのバックエッジ
* 黒の破線はダミーノードへの接続を示す
* 数字はヘッドのノードの処理時間を経路探索のための距離(distance)で、上から順にbest/typical/worstとなる

## ブロック情報CSVファイル

XMLのSimulinkブロックに付加する情報を指定します。本情報は、主にコードトランスレータの処理に使用します。CSVの形式は以下の通りです。

ブロック名,ブロックID,プロセッサ情報,結合先ブロック名,結合先ブロックID

各フィールドのは以下の通りです。

表 6‑1 CSVファイルのフィールド

|  |  |  |
| --- | --- | --- |
| フィールド | 意味 | 説明 |
| 1 | ブロック名 | 処理対象のブロック名。ブロックIDが指定されていない場合は必須です。ブロックIDが指定されている場合、省略可能です。また、ブロックIDと両方指定されている場合、ブロック名の指定が優先されます。 |
| 2 | ブロックID | 処理対象のブロックID。ブロック名が指定されていない場合は必須です。ブロック名が指定されている場合、この指定は無視します。 |
| 3 | プロセッサ情報 | コア割り当て等の情報を指定します。  現状は、CPUコア番号(0以上)です。-1はコア割り当てを行いません。  このフィールドは下記の結合先ブロックのフィールドとは排他です。このフィールドを指定する場合は下記の結合先ブロックまたは結合先ブロックIDを指定しません。 |
| 4 | 結合先ブロック名 | 実行時にタスクとして結合するSimulinkブロックを指定します。このフィールドは上記のプロセッサ情報のフィールドとは排他です。このフィールドを指定する場合は上記のプロセッサ情報のフィールドを指定しません。  結合先ブロックIDが指定されている場合、省略可能です。また、結合先ブロックIDと両方指定されている場合、ブロック名の指定が優先されます。 |
| 5 | 結合先ブロックID | 実行時にタスクとして結合するSimulinkブロックのIDを指定します。結合先ブロック名が指定されていない場合は必須です。結合先ブロック名が指定されている場合、この指定は無視します。 |

指定例は以下の通りです。

AAA,1,0 ←ブロックAAA(1)のプロセッサ情報として0を指定します

BBB,2,,CCC,3 ←ブロックBBB(2)をブロックCCC(3)へ結合します

CCC,3,1 ←ブロックCCC(3)のプロセッサ情報として1を指定します

# モジュール仕様

## SimulinkModel(データバインディング)

SimulinkModelは、SimulinkModel.hxx, SimulinkModel.cxxの二つのファイルから構成される、データバインディングのツールCodesynthesis XSDにより自動生成されるC++のコードです。

C++のnamespaceは以下の通りです。

namespace SimulinkModel

{

namespace XSD {

各クラス、関数はこの名前空間に置かれる

}

}

そのため、開発するC++コードでは以下のように名前空間を指定する必要があります。

::SimulinkModel::XSD::blocks\_T blocks;

名前の衝突がないのであれば、以下のようにnamespaceを省略する記述を行うことで、記述を簡単にすることができます。

using namespace SimulinkModel;

using namespace SimulinkModel::XSD;

blocks\_T blocks;

以下、概要を説明しますが、詳細についてはdoxygenで生成されるhtml/latexのドキュメントをご参照ください。SimulinkModelを用いた簡単なサンプルコードについてはxsd\_driver.cxxをご参照ください。

### タグと型(クラス、イテレータ)の対応

以下はSimulinkModelの基本的な構成です

* XMLのタグをツリー構造で保持、ツリーのルートは<blocks>タグ
* タグの型はXMLスキーマで記述したもの
* 子タグは親タグにて1次元配列(子タグの型)で管理
* この配列は ::std::vector を継承したクラスで、タグと同じ関数名で取得化

そのため、XMLのタグを順にアクセスするには基本的には目的のタグの配列に対してイテレータ(反復子)を用いた方法を主に利用します。以下は、XMLタグとクラス、1次元配列取得関数とイテレータの一覧です。

表 7‑1タグと型の一覧

|  |  |  |  |
| --- | --- | --- | --- |
| XMLタグ | クラス | 配列取得関数 | 配列の型/イテレータ |
| <blocks> | blocks\_T | block\_T::blocks() | block\_T::blocks\_sequence |
| block\_T::blocks\_iterator  block\_T::blocks\_const\_iteator |
| <file> | file\_T | blocks\_T::file() | blocks\_T::file\_sequence |
| blocks\_T::file\_iterator  blocks\_T::file\_const\_iterator |
| <shim-params> | shim\_params\_T | blocks\_T::shim\_params() | blocks\_T::shim\_params\_sequcece |
| blocks\_T::shim\_params\_iterator  blocks\_T::shim\_params\_const\_iterator |
| <block> | block\_T | blocks\_T::block() | blocks\_T::block\_sequence |
| blocks\_T::block\_iterator  blocks\_T::block\_const\_iterator |
| <code> | code\_T | blocks\_T::code()  (<blocks>タグ内) | blocks\_T::code\_\_sequence |
| blocks\_T::code\_iterator  blocks\_T::code\_const\_iterator |
| block\_T::code()  (<block>タグ内) | block\_T::code\_\_sequence |
| block\_T::code\_iterator  block\_T::code\_const\_iterator |
| <object> | object\_T | blocks\_T::object() | blocks\_T::object\_sequence |
| blocks\_T::object\_iterator  blocks\_T::object\_const\_iterator |
| <deftype> | deftype\_T | blocks\_T::deftype() | blocks\_T::deftype\_sequcece |
| blocks\_T::deftype\_iterator  blocks\_T::deftype\_const\_iterator |
| <includes> | includes\_T | blocks\_T::includes() | blocks\_T::includes\_sequcece |
| blocks\_T::includes\_iterator  blocks\_T::includes\_const\_iterator |
| <function> | function\_T | blocks\_T::function() | blocks\_T::function\_sequcece |
| blocks\_T::function\_iterator  blocks\_T::function\_const\_iterator |
| <input> | ioport\_T | block\_T::input() | block\_T::input\_sequence |
| block\_T::input\_iterator  block\_T::input\_const\_iterator |
| <output> | ioport\_T | block\_T::output() | block\_T::output\_sequence |
| block\_T::output\_iterator  block\_T::output\_const\_iterator |
| <connect> | connection\_T | ioport\_T::connect() | ioport\_T::connect\_\_sequence |
| ioport\_T::connect\_iterator  ioport\_T::connect\_const\_iterator |
| <var> | var\_T | block\_T::var()  (<block>タグ内) | block\_T::var\_sequence |
| block\_T::var\_iterator  block\_T::var\_const\_iterator |
| link\_T::var()  (<connect>タグ内) | link\_T::var\_sequence |
| link\_T::var\_iterator  link\_T::var\_const\_iterator |
| <signal> | param\_T | block\_T::signal() | block\_T::signal\_sequcence |
| block\_T::signal\_iterator  block\_T::signal\_const\_iterator |
| <state> | param\_T | block\_T::state() | block\_T::state\_sequence |
| block\_T::state\_iterator  block\_T::state\_const\_iterator |
| <trigger\_state> | param\_T | block\_T::trigger\_state() | block\_T::trigger\_state\_sequence |
| block\_T::trigger\_state\_iterator  block\_T::trigger\_state\_const\_iterator |
| <invariant\_signal> | param\_T | block\_T::invariant\_signal() | block\_T::invariant\_signal\_sequence |
| block\_T::invariant\_signal\_iterator  block\_T::invariant\_signal\_const\_iterator |
| <param> | param\_T | block\_T::param() | block\_T::param\_sequence |
| block\_T::param\_iterator  block\_T::param\_const\_iterator |
| <const\_param> | param\_T | block\_T::const\_param() | block\_T::const\_param\_sequence |
| block\_T::const\_param\_iterator  block\_T::const\_param\_const\_iterator |
| <performance> | performance\_T | block\_T::performance() | block\_T::performance\_sequcence |
| block\_T::performance\_iterator  block\_T::performance\_const\_iterator |
| <forward> | link\_T | block\_T::forward() | block\_T::forward\_sequence |
| block\_T::forward\_iterator  block\_T::forward\_const\_iterator |
| <backward> | link\_T | block\_T::backward() | block\_T::backward\_sequence |
| block\_T::backward\_iterator  block\_T::backward\_const\_iteraror |
| <merged\_to> | block\_name\_T | block\_T::merged\_to() | block\_T::merged\_to\_sequence |
| block\_T::merged\_to\_iterator  block\_T::merged\_to\_const\_iterator |
| <merge\_block> | block\_name\_T | block\_T::merge\_block() | block\_T::merge\_block\_sequence |
| block\_T:: merge\_block\_iterator  block\_T:: merge\_block\_const\_iterator |
| <bind> | block\_name\_T | block\_T::bind() | block\_T::bind\_sequence |
| block\_T::bind\_iterator  block\_T::bind\_const\_iterator |
| <bound\_to> | block\_name\_T | bock\_T::bound\_to() | block\_T::bound\_to\_sequence |
| block\_T::bound\_to\_iterator  block\_T::bound\_to\_const\_iterator |
| <initcode> | code\_T | param\_T::initcode() | param\_T::initcode\_sequence |
| param\_T::initcode\_iterator  param\_T::initcode\_const\_iterator |

各タグの属性については、その属性名と同じメンバ関数でC++の参照(リファレンス)が取得できます。ただし、必須かそうでないかで、型が異なるので注意してください。

表 7‑2タグと属性の一覧

|  |  |  |  |
| --- | --- | --- | --- |
| XMLタグ | 属性名 | 属性取得関数 | 取得される参照の型 |
| <blocks> | name | blocks\_T::name() | blocks\_T::name\_optional |
| <file> | type | file\_T::type() | file\_T::type\_type |
| name | file\_T::name() | ::xml\_schema::string |
| timestamp | file\_T::timestamp() | ::xml\_schema::string |
| size | file\_T::size() | int |
| <shim-params> | element | shim\_params\_T::element() | shim\_params\_T::element\_type |
| name | shim\_params\_T::name() | ::xml\_schema::string |
| <block> | blocktype | block\_T::blocktype() | ::xml\_schema::string |
| name | block\_T::name() | ::xml\_schema::string |
| rate | block\_T::rate() | block\_T::rate\_optional |
| Period | block\_T::Period() | block\_T::Period\_optional |
| PulseType | block\_T::PulseType() | block\_T::PulseType\_optional |
| PulseWidth | block\_T::PulseWidth() | block\_T::PulseWidth\_optional |
| id | block\_T::id() | block\_T::id\_optional |
| peinfo | block\_T::peinfo() | block\_T::peinfo\_optional |
| priority | block\_T::priority() | block\_T::priority\_optional |
| sequence | block\_T::sequence() | block\_T::sequcence\_optional |
|  | stateflow | block\_T::stateflow() | block\_T::stateflow\_optional |
|  | DataStoreName | block\_T::  DataStoreName() | block\_T:: DataStoreName\_optional |
|  | TreatAsAtomicUnit | block\_T:: TreatAsAtomicUnit() | block\_T:: TreatAsAtomicUnit\_optional |
|  | SampleTimeType | block\_T:: SampleTimeType() | block\_T:: SampleTimeType\_optional |
|  | TriggerType | block\_T:: TriggerType() | block\_T:: TriggerType\_optional |
|  | TriggerSignalSampleTime | block\_T:: TriggerSignalSampleTime() | block\_T:: TriggerSignalSampleTime\_optional |
|  | measuringResult | block\_T:: measuringResult() | block\_T:: measuringResult\_optional |
| <code> | type | code\_T::type() | ::xml\_schema::string |
| file | code\_T::file() | code\_T::file\_optional |
| line | code\_T::line() | code\_T::line\_optional |
| index | code\_T::index() | code\_T::index\_optional |
| <object> | type | object\_T::type() | ::xml\_schema::string |
| has\_struct | object\_T::has\_struct() | object\_T::has\_struct\_optional |
| name | object\_T::name() | ::xml\_schema::string |
| size | object\_T::size() | object\_T::size\_optional |
| value | object\_T::value() | object\_T::value\_optional |
| kind | object\_T::kind() | object\_T::kind\_optional |
| <deftype> | type | deftype\_T::type() | ::xml\_schema::string |
| has\_struct | deftype \_T::has\_struct() | deftype\_T::has\_struct\_optional |
| name | deftype \_T::name() | ::xml\_schema::string |
| size | deftype \_T::size() | deftype\_T::size\_optional |
| value | deftype \_T::value() | deftype\_T::value\_optional |
| kind | deftype \_T::kind() | deftype\_T::kind\_optional |
| <includes> | index | includes\_T::index() | int |
| from | includes\_T::from() | includes\_T::from\_optional |
| file | includes\_T::file() | ::xml\_schema::string |
| mark | includes\_T:mark() | includes\_T::mark\_optional |
| <function> | name | function\_T::name() | ::xml\_schema::string |
| type | function\_T::type() | function\_T::type\_type |
| <input>  <output> | line | ioport\_T::line() | ::xml\_schema::string |
| port | ioport\_T::port() | ::xml\_schema::string |
| <connect> | block | connection\_T::block() | ::xml\_schema::string |
| port | connection\_T::port() | ::xml\_schema::string |
| <var> | name | var\_T::name() | ::xml\_schema::string |
| type | var\_T::type() | ::xml\_schema::string |
| mode | var\_T::mode() | var\_T::mode\_type |
| line | var\_T::line() | var\_T::line\_optional |
| port | var\_T::port() | var\_T::port\_optional |
| size | var\_T::size() | var\_T::size\_optional |
| start | var\_T::start() | var\_T::start\_optional |
| end | var\_T::end() | var\_T::end\_optional |
| <signal>  <state>  <trigger\_state>  <invariant\_signal>  <param>  <const\_param> | storage | param\_T::storage() | ::xml\_schema::string |
| name | param\_T::name() | ::xml\_schema::string |
| type | param\_T::type() | ::xml\_schema::string |
| size | param\_T::size() | param\_T::size\_optional |
|  | value | param\_T::value() | param\_T::value\_optional |
| <performance> | type | performance\_T::type() | performance\_T::type\_type |
| best | performance\_T::best() | float |
| typical | performance\_T::typical() | float |
| worst | performance\_T::worst() | float |
| <forward>  <backward> | block | link\_T::block() | ::xml\_schema::string |
| type | link\_T::type() | ::xml\_schema::string |
| label | link\_T::label() | link\_T::label\_optional |
| <merged\_to>  <merge\_block>  <bind>  <bound\_to> | block | block\_name\_T::block() | ::xml\_schema::string |

各タグのoptionalな型は、boolへのキャストを使い、if文等の条件判定でその有無をテストできます。

例えば<block>の情報および、<block>の<input>の<connect>を全てテキスト表示するのであれば、以下のような反復子を用いて各タグを順に取得し、属性をメンバ関数で取得する、ということになります。

blocks\_T blocks;

for (blocks\_T::block\_iterator bi (blocks.begin()); bi != blocks.end(); bi++) {

block\_T& block = \*bi; // blockの参照はこのように取得できる

for (block\_T::input\_iterator ii (block.input().begin();

ii !=block.input().end(); ii++) {

::std::cout << "block=" << block.name() << " blocktype=" << block.blocktype();

block\_T::rate\_optional& rate = block.rate(); // rate取得

if (rate)

::std::cout << " rate=" << rate; // rateがある場合はrateを出力

::std::cout << ::std::endl;

for (ioport\_T::connect\_iterator ci (ii->connect().begin();

ci != ii->connect().end(); ci++) {

::std::cout<< " input line=" ii->line()

<< " port=" << ii->port() << " connect=" << ci->block()

<< " port=" << ci->port() << ::std::endl;

}

}

}

### XML入力関数

SimulinkModelのグローバルなルートタグは、<blocks>です。XMLの入力関数は、タグと同じ名前、blocks()になります。blocks()には入力元に応じた種類の関数を持っていますが、通常はファイルを指定したものになります。

using namespace ::SimulinkModel;

using namespace ::SimulinkModel::XSD:

::std::auto\_ptr< blocks\_T > blocks (blocks ("model.xml"));

### XML出力関数

出力関数も入力関数と同様にblocks()です。出力は出力ストリームに出すようになっていて、引数の差はエラーハンドラの違いで、基本的にはXMLのルートタグのオブジェクトとnamespaceの情報を与えて出力します。

using namespace ::SimulinkModel;

using namespace ::SimulinkModel::XSD:

blocks\_T blocks;

xml\_schema::namespace\_infomap map;

map["SM"].name = "http://example.com/SimulinkModel"; // namespace is "SM"

map["SM"].schema = "SimulinkModel.xsd"; // SimulinkModel XML Schema

::std::ofstream os ("output.xml");

blocks (os, blocks, map);

## SimulinkXML(SimulinkModelの支援モジュール)

SimulinkXMLモジュールは、SimulinkXML.h, SimulinkXML.cxxのC++のコードです。このモジュールのC++のnamespaceは以下の通りです。

namespace Simulink

{

namespace XML {

各クラス、関数はこの名前空間に置かれる

}

}

そのため、開発するC++コードでは以下のように名前空間を指定する必要があります。

::Simulink::XML::Reader rdr ("model.xml");

このモジュールに含まれるクラス、クラスメンバ、型、および関数は以下の通りです。

表 7‑3 SimulinkXMLモジュールのクラス一覧

|  |  |  |
| --- | --- | --- |
| クラス名 | 内容 | |
| Reader | SimulinkモデルのXMLを入力し、そのXMLファイルのブロック名をキーとして<block>タグを検索する機能と、<connect>タグにあるブロック名とポート名を指定し、対応する<input>または<output>のタグを検索する機能を提供する | |
| コンストラクタ | 説明 |
| Reader(xml\_file,validation) | SimulinkモデルのXMLファイルを指定し、XMLファイルの読み込みを行うコンストラクタ。validationでスキーマとの整合性をチェック可能 |
| Reader() | デフォルトコンストラクタ。load\_xml()を使用してXMLファイルを読み込む必要がある |
| 公開メンバ | 説明 |
| load\_xml(xml\_fil,validation) | デフォルトコンストラクタで作成したオブジェクトにXMLファイルを読み込ませる関数。validationでスキーマとの整合性をチェック可能 |
| find\_input\_port() | 入力のIOPortの連想配列から指定したブロック名とポート名のIOPortを探し、あれば返す |
| find\_output\_port() | 出力のIOPortの連想配列から指定したブロック名とポート名のIOPortを探し、あれば返す |
| find\_inport\_input() | Inportのブロックに対応するSubSystemの<input>を保持しているIOPortのポインタを返す |
| find\_outport\_output() | Outportのブロックに対応するSubSystemの<output>を保持しているIOPortのポインタを返す |
| find\_block() | ブロック名の連想配列から指定されたブロック名のBlockInfoを探し、あれば返す |
| xml\_blocks() | Simulinkモデル全体のXMLの<blocks>タグを返す |
| block\_map() | ブロック名をキーとした<block>タグの連想配列(map)を返す |
| input\_map() | ポート名をキーとした<input>タグを格納している入力IOPortの連想配列(multimap)を返す |
| outport\_map() | ポート名をキーとした<output>タグを格納している出力IOPortの連想配列(multimap)を返す |
| block\_list() | BlockInfoのポインタのvectorを返す。BlockInfoのIDの値を配列の添え字とすれば、そのBlockInfoのポインタを取得できる |
| find\_loop() | 指定したブロックを起点にループがあるか調べる |
| dump() | クラスの内容をテキストで標準出力に出力する |
| キャスト | 説明 |
| (bool) | Readerクラスが使用可能であればtrueに、使用不可であればfalseになる |
| 限定公開メンバ | 説明 |
| add\_block\_map() | <block>のタグをブロック名の連想配列に追加する |
| add\_inport\_map() | <input>と<block>のタグを入力IOportの連想配列に追加する |
| add\_outport\_map() | <output>と<block>のタグを出力IOPortの連想配列に追加する |
| process\_blocks() | 全ての<block>タグを連想配列に登録する |
| process\_blocks\_complete() | マージされたブロック、プロセス集約したブロックを関連付ける |
| process\_port() | 全ての<input>/<output>タグを連想配列に登録する |
| process\_graph() | BlockInfoのforward/backwardの接続関係の構築と伝播する変数を設定する |
| process\_top\_subsystem() | process\_blocks\_complete()から呼び出される、一括りのコードとして切り出されたSubSystemの処理を行う |
| process\_all() | process\_block()、process\_blocks\_complete()、process\_graph()、process\_port()を順に実行後、BlockInfoのポインタのvectorを作成する |
| depth\_search\_for\_loop() | find\_loopで使用するdepth first search関数 |
| BlockInfo | ブロックの情報としてSimulinkモデルのブロックのタグ<block>と上位階層のブロックのタグ<block>を格納し、連想配列の要素として使うクラス | |
| コンストラクタ | 説明 |
| BlockInfo(block,upper) | 対象としているブロックのタグ<block>とその上位階層のブロックのタグ<block>を指定したコンストラクタ |
| BlockInfo() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| id() | XML内に出現する順序でつけられたIDを返す |
| block() | SimulinkモデルのブロックXMLタグ<block>を返す |
| upper() | ブロックの上位階層のブロックのXMLタグ<block>を返す |
| is\_subsystem() | ブロックがSubsystemの場合trueを返す |
| has\_update() | ブロックがupdate部を持つ場合trueを返す |
| is\_sfunction() | ブロックがS-Functionであればtrueを返す |
| subsyste\_type() | サブシステムの種別を返す |
| upper\_subsystem() | このブロックを含む上位のサブシステムを返す |
| lower\_block() | このブロック(Subsystem)の下位のブロックのvectorを返す |
| forward() | このブロックに接続する後続ブロックのリストを返す(BlockConnectのポインタのvector) |
| backward() | このブロックに接続する先行ブロックのリストを返す(BlockConnectのポインタのvector) |
| top\_subsystem() | このブロックがコードを一体として切り出したSubsystemのブロックの場合、そのブロック情報を返す。(SwitchCase/If, Iterator等) |
| top() | このブロックがコードを一体として切り出したSubsystemのブロックの場合、そのSubSystemのXMLの<block>を返す。 |
| contain\_block() | コードを一体として切り出したこのSubsystemに含まれるブロック情報のリストを返す |
| is\_contained() | 指定したブロック情報が、コードを一体として切り出したこのSubsystemに含まれるブロック情報にあるか調べる |
| merged\_to() | このブロックをマージしているブロックのブロック情報のリストを返す(通常は空か1) |
| merge\_blocks() | このブロックがマージしているブロックのブロック情報のリストを返す |
| bound\_to() | このブロックをプロセス集約している先頭プロセスのリストを返す(通常は空か1) |
| bind\_blocks() | このブロックがプロセス集約しているブロックのリストを返す |
| trigger\_port() | Triggered Subsystemの場合、Trigger Portのブロック情報を返す |
| enable\_port() | Enabled Subsystemの場合、Enable Portのブロック情報を返す |
| action\_port() | Action Subsystemの場合、Action Portのブロック情報を返す |
| is\_atomic\_subsystem() | AtomicなSubsystem(IteratorやTreatAsAtomicUnit=”on”であるもの)かどうか調べる |
| dump() | クラスの内容をテキストで標準出力に出力する |
| オペレータ | 説明 |
| == | 名前が等しい場合true |
| 静的メンバ | 説明 |
| max\_id() | ブロックの最大のID+1を返す |
| IOPort | Simulinkモデルの入出力を示すタグ<input>または<output>と、それが含まれるSimulinkモデルのブロックのタグ<block>を格納し、連想配列の要素として使われるクラス | |
| コンストラクタ | 説明 |
| IOPort(block,inout) | タグ<block>とタグ<input>または<output>を指定したコンストラクタ |
| IOPort() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| block\_tag() | XMLタグ<block>を返す |
| ioport\_tag() | XMLタグ<input>または<outpu>を返す |
| blockname() | ブロックの名前を返す |
| port() | XMLタグ<input>または<output>のport名を返す |
| line() | XMLタグ<input>または<output>のline名を返す |
| match() | そのIOPortが指定したブロック名とport名に一致しているか調べる関数 |
| dump() | クラスの内容をテキストで標準出力に出力する |
| BlockConnet | Simulinkのブロックの接続情報を保持するクラス  BlockInfoのポインタをhead/tailとして記録し有向エッジを表す | |
| コンストラクタ | 説明 |
| BlockConnect(tail,head) | 有向エッジ(tail,head)を指定したコンストラクタ |
| BlockConnect() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| var\_list() | ブロック間で伝播する変数(var\_T型)のリストを返す |
| head() | 有向エッジの頭側のブロック(BlockInfo)を返す |
| tail() | 有向エッジの尾側のブロック(BlockInfo)を返す |
| オペレータ | 説明 |
| == | 同じ有向エッジかどうかを比較する |
| codelist | ブロックのコードにはブロック固有のコードと実行時に関係する自動変数やSubsystemのコードが混在し、それらを分離するためのクラス | |
| コンストラクタ | 説明 |
| codelist(beg, end) | コードの開始、終了のポインタを指定したコンストラクタ |
| codelist(string) | コード全体のstringを指定したコンストラクタ |
| codelist() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| process(beg, end) | デフォルトコンストラクタを使った場合で、コードの開始、終了のポインタを指定してコードの分離を実行する |
| mark\_list() | 分離したコード断片(c\_mark)のvectorを返す |
| c\_mark | ブロックのコードを分離した断片を保持するクラスでcodelistから利用される | |
| コンストラクタ | 説明 |
| c\_mark(…) | コード断片の情報を指定したコンストラクタ |
| 公開メンバ | 説明 |
| set() | コードの断片の情報を設定する |
| text\_beg() | コメント(定型化した分離情報)を含まないのコードの開始点を返す |
| text\_end() | コメントを含まないブロックのコードの終了点を返す |
| beg() | コメントを含むブロックのコードの開始点を返す |
| end() | コメントを含むブロックのコードの終了点を返す |
| type() | 定型化したコメントのタイプを返す(Sはブロックの開始コード、Eはブロックの終端コード、Xはサブブロックが切り替わった情報) |
| num() | コメントのネスティングのレベルを返す |
| name() | 対象となるコードのブロックの名前を返す |
| キャスト | 説明 |
| (::std::string) | beg(),end()の範囲を::std::stringへ変換する |
| const\_block\_list\_T | 書き換え不可能なBlockInfoのポインタのリストを扱うためのクラス | |
| コンストラクタ | 説明 |
| const\_block\_list\_T(size,initvalue) | あらかじめサイズと初期値を指定したコンストラクタ |
| const\_block\_list\_T() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| checkinfo(binfo) | binfoで指定したBlockInfoのポインタがリストに含まれるかチェックする |
| add\_info(binfo) | binfoで指定したBlockInfoのポインタをリストに追加する |
| any() | 任意に利用可能なフラグの参照 |
| オペレータ | 説明 |
| += | リストに指定したBlockInfoのポインタを追加する |

表 7‑4 SimulinkXMLモジュールの型一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| ioport\_map\_T | portをキーにしたIOPortの連想配列(multimap) |
| ioport\_map\_iterator | 上記の状態変更用イテレータ |
| ioport\_map\_const\_iterator | 上記の状態参照用イテレータ |
| ioport\_map\_iterator\_pair | IOPortの連想配列の同一キーの要素の範囲を得る状態変更用の型 |
| ioport\_map\_const\_iterator\_pair | IOPortの連想配列の同一キーの要素の範囲を得る状態参照用の型 |
| block\_map\_T | ブロック名をキーとしたBlockInfoの連想配列(map) |
| block\_map\_iterator | 上記の状態変更用イテレータ |
| block\_map\_const\_iterator | 上記の状態参照用イテレータ |
| block\_list\_T | BlockInfoのポインタのvector |
| block\_list\_iterator | 上記の状態変更用イテレータ |
| block\_list\_const\_iterator | 上記の状態参照用イテレータ |
| const\_block\_list\_iterator | 書き換え不可能(const)のBlockInfoのポインタの状態変更用イテレータ |
| const\_block\_list\_const\_iterator | 上記の状態参照用イテレータ |
| var\_list\_T | <var>タグに対応する型var\_Tのポインタのvecotor |
| block\_connect\_list\_T | BlockConnectのvector |
| marklist\_T | c\_markのvector |

表 7‑5 SimulinkXMLモジュールの関数の一覧

|  |  |
| --- | --- |
| 関数 | 説明 |
| find\_multirate\_subsystem (binfo) | 指定したBlockInfoの上位がTrigger/Enable SubsystemであればそのBlockInfoのポインタを返す。そうでなければNULLが返る |

### Readerクラス

Readerクラスは、データバインディングの機能であるSimulinkModelに、連想配列を用いた探索機能を提供し、使用者を支援する機能です。ここではクラスの公開メンバを説明します。限定公開メンバについては実装を参照してください。

#### Readerクラスコンストラクタ

Readerクラスのコンストラクタです。XMLファイルを指定する形式のコンストラクタとデフォルトコンストラクタの二種類があります。

// XMLファイルを読み込むコンストラクタ

Reader (const char\* xml, bool validation = true);

Reader (); // デフォルトコンストラクタ

コンストラクタの引数xmlに、SimulinkモデルのXMLファイルを指定することで、XMLファイルの入力と、ブロック名やIOポートの連想配列作成を行います。また、validationはデフォルトではtrueで、XMLスキーマを用いてXMLをチェックするため、入力するXMLファイルと同じディレクトリにSimulinkModel.xsdがないと、入力エラーになりますので、実行時には注意してください。falseを指定するとこのチェックを行いません。

コンストラクタ引数を指定しない場合、load\_xml()関数を使用して別途XMLファイルを読み込む必要があります。

boolのキャストを定義しているため、if文等の条件判定部分にオブジェクトそのものを指定した論理式が書けます。例えば、演算子`!’を使えば以下のように、使用可能かどうかを判定する事ができます。

::Simulink::XML::Reader xrd;

if (!xrd) { ::std::cerr << “Error” << ::std::endl; }

#### load\_xml()

SimulinkモデルのXMLファイルを読み込み、ブロック名やIOポートの連想配列作成を行います。

int load\_xml (const char\* xml, bool validation = true);

この関数は、デフォルトコンストラクタを使用した場合に使用します。

validationがtrueの場合、XMLスキーマを用いてXMLをチェックするため、入力するXMLファイルと同じディレクトリにSimulinkModel.xsdがないと、入力エラーになりますので、実行時には注意してください。falseを指定するとこのチェックを行いません。

#### find\_input\_port()

ブロック名とポート名を指定し、そのブロックのポート名が一致するタグ<input>を探します。

const IOPort\* find\_input\_port(const ::std::string& block,

const ::std::string& port) const; // 状態参照

IOPort\* find\_input\_port (const ::std::string& block,

const ::std::string& port); // 状態変更

例えば<output>の<connect>タグにあるブロック名、ポート名を指定すると、接続先のブロックのタグ<block>とタグ<input>をIOPortクラスのポインタで返します。見つからない場合はNULLを返します。IOPortクラスについては、7.2.3を参照してください。

#### find\_output\_port()

ブロック名とポート名を指定し、そのブロックのポート名が一致するタグ<output>を探します。

const IOPort\* find\_output\_port(const ::std::string& block,

const ::std::string& port) const; // 状態参照

IOPort\* find\_output\_port (const ::std::string& block,

const ::std::string& port); // 状態変更

例えば<input>の<connect>タグにあるブロック名、ポート名を指定すると、接続先のブロックのタグ<block>とタグ<outport>をIOPortクラスのポインタで返します。見つからない場合はNULLを返します。IOPortクラスについては、7.2.3を参照してください。

#### find\_inport\_input()

InportのブロックのBlockInfoクラスのポインタを指定し、対応するSubSystemの<input>を保持するIOPortクラスのポインタを取得します。Inportは<output>のみなので、SubSystemの<input>が実際の入力元となります。

const IOPort\* find\_inport\_input(const BlockInfo\* port) const; // 状態参照

IOPort\* find\_inport\_input (const BlockInfo\* port); // 状態変更

#### find\_outport\_output()

OutportのブロックのBlockInfoクラスのポインタを指定し、対応するSubSystemの<output>を保持するIOPortクラスのポインタを取得します。Outportは<input>のみなので、SubSystemの<output>が実際の出力際となります。

const IOPort\* find\_outport\_output(const BlockInfo\* port) const; // 状態参照

IOPort\* find\_outport\_output (const BlockInfo\* port); // 状態変更

#### find\_block()

ブロック名を指定し、そのブロック名と一致するタグ<block>を探します。

const BlockInfo\* find\_block (const ::std::string& block) const; //状態参照

BlockInfo\* find\_block (const ::std::string& block); //状態変更

見つかった場合は、BlockInfoクラスのポインタを返します。見つからない場合は、NULLを返します。BlockInfoクラスは、そのブロックの<block>タグと、上位階層の<block>を保持しています。BlockInfoクラスについては7.2.2を参照してください。

#### xml\_blocks()

読み込んだSimulinkModelのXMLのタグ<blocks>を返します。

const ::SimulinkModel::XSD::blocks\_T& xml\_blocks (void) const; // 状態参照

::SimulinkModel::XSD::blocks\_T& xml\_blocks (void); // 状態変更

取得したタグ<blocks>の利用方法については7.1を参照してください。

#### block\_map()

読み込んだSimulinkモデルのブロック名をキーとした、BlockInfoの連想配列を返します。

const block\_map\_T& block\_map (void) const; // 状態参照

block\_map\_T& block\_map (void); // 状態変更

連想配列は ::std::map を用いていますので、mapの機能を使うことができます。

この時使用するイテレータは以下の通りです。

block\_map\_const\_iterator // 状態参照

block\_map\_iterator // 状態変更

find\_block()と同じような処理は、次のように書くことができます。

Reader rdr (“model.xml”);

block\_map\_const\_iterator i = rdr.block\_map().find(“ABC”);

if (i == rdr.block\_map().end())

{ 見つかりません; }

else

{

BlockInfo& info = i->second; // mapの値はBlockInfo

block\_T\* abc = info.block(); // ABCの<block>タグ

block\_T\* upper\_abc = info.upper(); // 上位階層の<block>タグ

見つかった場合の処理;

}

BlockInfoクラスについては7.2.2を参照してください。

#### input\_map()

読み込んだSimulinkモデルの、<input>のポート名をキーとした入力のIOPortの連想配列を返します。IOPortクラスについては7.2.3を参照してください。

const ioport\_map\_T& input\_map (void) const; //状態参照

ioport\_map\_T& input\_map (void); //状態変更

連想配列は ::std::multimap を用いていますので、multimapの機能(例えばequal\_range()等)を使うことができます。

この時使用するイテレータと利用可能な型は以下の通りです。

ioport\_map\_const\_iterator // 状態参照

ioport\_map\_iterator // 状態変更

ioport\_map\_const\_iterator\_pair // 状態参照イテレータのペア(equal\_range等)

ioport\_map\_iterator\_pair // 状態変更イテレータのペア

#### output\_map()

読み込んだSimulinkモデルの、<output>のポート名をキーとした出力のIOPortの連想配列を返します。IOPortクラスについては7.2.3を参照してください。

const ioport\_map\_T& output\_map (void) const;

ioport\_map\_T& output\_map (void);

連想配列は ::std::multimap を用いていますので、multimapの機能を使うことができます。

この時使用するイテレータと利用可能な型は以下の通りです。

ioport\_map\_const\_iterator // 状態参照

ioport\_map\_iterator // 状態変更

ioport\_map\_const\_iterator\_pair // 状態参照イテレータのペア

ioport\_map\_iterator\_pair // 状態変更イテレータのペア

#### block\_list()

読み込んだSimulinkモデルのBlockInfo\*のvectorを返します。BlockInfoのid()で取得した値をvectorの添え字でアクセスすれば、当該BlockInfoのポインタを得ることができます。

const block\_list\_T& block\_list (void) const;

block\_list\_T& block\_list (void);

#### find\_loop()

指定したブロックと探索報告(前方、後方)を指定して、ループがあるか探索します。

enum search\_dir { SEARCH\_FORWARD, SEARCH\_BACKWARD };

int find\_loop (int index, int \*route,

earch\_dir dir, bool break\_multirate) const;

int find\_loop (int index, search\_dir dir, bool break\_multirate) const;

indexは探索を開始するBlockInfoのid()で取得したIDです。最初の形式では、検出したルートをintの配列に保持します。dirにSEARCH\_FORWARDを指定した場合は後方を探します。SEARCH\_BACKWARDは前方を探します。break\_multirateはTriggerd Subsystem等マルチレート要因となるブロックで探索を打ち切る場合はtrueを指定します。

ループが見つからない場合は0を返します。見つかった場合は1を返します。

routeを指定した場合、始点indexからのルートが記録されます。route[X]は、XのIDを持つブロックから探索したブロックのIDが保持されます。routeを指定するやりかたは下記を参考にしてください。

Reader rdr;

/\* routeのvectorを0で初期化 \*/

::std::vector < int > route (rdr.block\_list().size (), 0);

int start = blockinfo.id();

if (xrd.find\_loop (start, route.data ()) {

int i = start;

int j;

do {

j = route[i];

::std::cout << i << “ to “ << j << ::std::endl;

i = j;

}while (j != start && j != 0);

}

#### dump()

Readerクラスの内容をテキストにして標準出力にダンプします。

void dump (void) const

本メンバ関数は、動作確認用です。

#### キャスト(bool)

Readerクラスが使用可能か判定できるようにboolのキャストが使用できます。即ち、if文等の条件としてReaderクラスのオブジェクトをそのまま指定できます。

operator bool () const;

使用方法については、7.2.1.1のコンストラクタの説明を参照してください。

### BlockInfoクラス

BlockInfoクラスは、ブロックの<block>タグと、上位階層の<block>を持ち、Readerクラスのブロックの連想配列の要素として使われるクラスです。このクラスは通常参照するだけで、使用者が作ることがないクラスです。ここではクラスの公開メンバを説明します。

#### BlockInfoクラスコンストラクタ

BlockInfoクラスのコンストラクタです。ブロックの<block>タグと上位階層の<block>タグを指定するコンストラクタとデフォルトコンストラクタがあります。

BlockInfo(::SimulinkModel::XSD::block\_T\* block,

::SimulinkModel::XSD::block\_T\* upper); // ブロックの指定

BlockInfo(); // デフォルトコンストラクタ

このコンストラクタは呼ばれるたびに内部シーケンス番号を1から順に増やし、オブジェクトのIDとします。

#### id()

オブジェクトのIDを返します。

unsigned int id (void) const

オブジェクトのIDはコンストラクタが呼ばれるたびに1から順に増え、一意です。これはオブジェクトを識別する目的(例えばタスクIDの付与)に使用します。

#### block()

ブロックの<block>タグのポインタを返します。

::SimulinkModel::XSD::block\_T\* block (void) const; // 状態参照

::SimulinkModel::XSD::block\_T\*& block (void); // 状態変更

#### upper()

ブロックの上位階層ブロックの<block>タグのポインタを返します。

::SimulinkModel::XSD::block\_T\* upper (void) const; // 状態参照

::SimulinkModel::XSD::block\_T\*& upper (void); // 状態変更

上位階層がない場合はNULLを返します。

#### is\_subsystem()

ブロックがSubsystemであればtrueを、そうでなければfalseを返します。

bool is\_subsystem (void) const; // 状態参照

bool& is\_subsystem (void); // 状態変更

#### has\_update()

ブロックがUnitDelay等の遅延ブロックで、update部を持てばtrueを、そうでない場合はfalseを返します。

bool has\_update (void) const; // 状態参照

bool& has\_update (void); // 状態変更

#### is\_sfunction()

ブロックがS-Functionであればtrueを、そうでない場合はfalseを返します。

bool is\_sfunction (void) const; // 状態参照

bool& is\_sfunction (void); // 状態変更

#### subsystem\_type()

ブロックがSubsystemの場合、その種別を返します。種別はATOMIC, ENABLED, TRIGGERED, ENABLED\_TRIGGERD, ACTION, ITERATOR, STATEFLOWのいずれかです。(ACTIONはSwitchCaseやIfブロックのAcction Subsystemです)

typedef enum { NONE, ATOMIC, ENABLED, TRIGGERED, ENABLED\_TRIGGERED,

ACTION, ITERATOR, STATEFLOW } subsystem\_type\_T;

subsystem\_type\_T subsystem\_type (void) const; // 状態参照

subsystem\_type\_T& subsystem\_type (void); // 状態変更

ブロックがSubsystemでない場合はATOMICが返ります。ただし、ブロックはSubsystemではないので、この結果に意味はありません。

#### upper\_subsystem()

ブロックがSubsystem内のブロックである場合、SubsystemのBlockInfoのポインタを返します。ブロックがSubsystem内のブロックではなく、最上位のブロックの場合、NULLを返します。

BlockInfo\* upper\_subsystem (void) const; // 状態参照

BlockInfo\*& upper\_subsystem (void); // 状態変更

#### lower\_block()

ブロックがSubsystemの場合、Subsystem直下のブロックのBlockInfoのポインタのvectorを返します。なお、Subsystemが含まれる場合、そのSubsystem内のブロックは含まれません。

const block\_list\_T& lower\_block (void) const; // 状態参照

block\_list\_T& lower\_block (void); // 状態変更

#### forward()

ブロックの後続ブロックの情報を保持しているBlockConnectクラスのvectorを返します。なお、後続ブロックは、<block>タグの<input>で記述されるブロックではなく、SubsystemやInportを経由した先のブロックです。BlockConnectクラスについては7.2.4を参照してください。

const block\_connect\_list\_T& forward (void); // 状態参照

block\_connect\_list\_T& forward (void); // 状態変更

#### backward()

ブロックの前方ブロックの情報保持しているBlockConnectクラスのvectorを返します。なお、後続ブロックは、<block>タグの<output>で記述されるブロックではなく、SubsystemやOutportを経由した先のブロックです。BlockConnectクラスについては7.2.4を参照してください。

const block\_connect\_list\_T& backward (void) const; // 状態参照

block\_connect\_list\_T& backward (void); // 状態変更

#### top\_subsystem()

ブロックがRTW-ECのコードから一体で切り出されたSubsystem(Triggered, Enabled, Iterator)に含まれるブロックであれば、この関数はSubsystemのBlockInfoのポインタを返します。そうでなければNULLを返します。

BlockInfo\* top\_subsystem (void) const; // 状態参照

BlockInfo\*& top\_subsystem (void); // 状態変更

#### top()

ブロックがRTW-ECのコードから一体で切り出されたSubsystem(Triggered, Enabled, Iterator)に含まれるブロックであれば、この関数はSubsystemの<block>タグのポインタを返します。そうでなければNULLを返します。

::SimulinkModel::XSD::block\_T\* top (void) const; // 状態参照

::SimulinkModel::XSD::block\_T\*& top (void); // 状態変更

#### contain\_block()

ブロックがRTW-ECのコードから一体で切り出されたSubsystem(Triggered, Enabled, Iterator)であれば、この関数は、このSubsystemに含まれるBlockInfoのポインタの空でないvectorを返します(含まれるブロックがなければ空のvectorを返す)。

const block\_list\_T& contain\_block (void) const; // 状態参照

block\_list\_T& contain\_block (void); // 状態変更

#### is\_contained()

ブロックがRTW-ECのコードから一体で切り出されたSubsystem(Triggered, Enabled, Iterator)であり、指定したBlockInfoのポインタまたは<block>タグのポインタが、このSubsystemに含まれていれば、trueを返します。

bool is\_contained (const BlockInfo\* b) const; // 状態参照

bool is\_contained (const ::SimulinkModel::XSD::block\_T\* b) const; // 状態変更

#### merged\_to()

このブロックをマージしているブロックのBlockInfoのポインタのvectorを返します。vectorは通常は空かサイズは1です。

const block\_list\_T& merged\_to (void) const; // 状態参照

block\_list\_T& merged\_to (void); // 状態変更

#### merge\_blocks()

このブロックがマージしているブロックのBlockInfoのポインタのvectorを返します。

const block\_list\_T& merge\_blocks (void) const; // 状態参照

block\_list\_T& merge\_blocks (void); // 状態変更

#### bound\_to()

このブロックをプロセス集約しているブロックのBlockInfoのポインタのvectorを返します。vectorは通常は空かサイズは1です。

const block\_list\_T& bound\_to (void) const; // 状態参照

block\_list\_T& bound\_to (void); // 状態変更

#### bind\_blocks()

このブロックにプロセス集約しているブロックのBlockInfoのポインタのvectorを返します。

const block\_list\_T& bind\_blocks (void) const; // 状態参照

block\_list\_T& bind\_blocks (void); // 状態変更

#### trigger\_port()

このブロックがTriggered Subsystemの場合、Trigger PortのBlockInfoのポインタを返します。

const BlockInfo\* trigger\_port (void) const; // 状態参照

BlockInfo\*& trigger\_port (void); // 状態変更

#### enable\_port()

このブロックがEnabled Subsystemの場合、Enable PortのBlockInfoのポインタを返します。

const BlockInfo\* enable\_port (void) const; // 状態参照

BlockInfo\*& enable\_port (void); // 状態変更

#### action\_port()

このブロックがAction Subsystemの場合、Action PortのBlockInfoのポインタを返します。

const BlockInfo\* action\_port (void) const; // 状態参照

BlockInfo\*& action\_port (void); // 状態変更

#### is\_atomic\_subsystem()

このブロックがAtomic Subsystem(Atomic, Iteratorまたは<block>で、TreatAsAtomicUnit=”on”の属性が指定されているSubsystem)の場合、trueを返します。そうでなければfalseを返します。

bool is\_atomic\_subsystem (void) const

#### dump()

BlockInfoクラスの内容をテキストにして標準出力にダンプします。

void dump (void) const

本メンバ関数は、動作確認用です。

#### オペレータ(==)

比較する文字列とBlockInfoのブロック名が一致する場合、trueを返します。

bool operator== (const ::std::string& name) const;

#### 静的メンバmax\_id()

最大のブロックのID+1を返します。

static unsigned int max\_id (void);

### IOPortクラス

IOPortクラスは、ブロックの一つの<input>または<output>タグと、それを持つブロックの<block>タグを持っています。このクラスはReaderクラスの入力、または出力信号の連想配列の要素として使われるクラスです。ここでは公開メンバを説明します。

#### IOPortクラスコンストラクタ

IOPortクラスのコンストラクタです。ブロックの<block>タグと<input>または<output>のタグを指定するコンストラクタとデフォルトコンストラクタがあります。

IOPort (::SimulinkModel::XSD::block\_T\* block,

::SimulinkModel::XSD::ioport\_T\* ioport); // ブロックとIOを指定

IOPort (); // デフォルトコンストラクタ

#### block\_tag()

対象IOPortの<block>タグを返します。

::SimulinkModel::XSD::block\_T\* block\_tag (void) const; //状態参照

::SimulinkModel::XSD::block\_T\*& block\_tag (void); //状態変更

#### ioport\_tag()

対象IOPortの<input>または<output>タグを返します。

::SimulinkModel::XSD::ioport\_T\* ioport\_tag (void) const; //状態参照

::SimulinkModel::XSD::ioport\_T\*& ioport\_tag (void); //状態変更

#### blockname()

<block>タグのブロック名を返します。

const ::std::string& blockname (void) const; //状態参照

::std::string& blockname (void); //状態変更

なお、ブロック名を書き換えるようなことはしないようにしてください。

#### port()

<input>または<output>タグのポート名を返します。

const ::std::string& port (void) const; //状態参照

::std::string& port (void); //状態変更

なお、ポート名を書き換えるようなことはしないようにしてください。

#### line()

<input>または<output>タグのline名を返します

const ::std::string& line (void) const; //状態参照

::std::string& line (void); //状態変更

なお、line名を書き換えるようなことはしないようにしてください。

#### match()

IOPortのブロックとポートが、指定したブロック名とポート名と一致しているか調べます

bool match (const ::std::string& b, const ::std::string& p) const

この関数は、Readerクラスのfind\_input\_port()やfind\_output\_port()で使われます。

#### dump()

IOPortクラスの内容をテキストにして標準出力にダンプします。

void dump (void) const

本メンバ関数は、動作確認用です。

### BlockConnectクラス

BlockConnectクラスは、ブロックの接続情報を保持するためのクラスです。情報としては、有向エッジのtail(尾)とhead(頭)の情報と、伝播する変数情報(<var>のポインタ)のvectorです。

BlockConnect (BlockInfo\* tail, BlockInfo\* head); // エッジの指定

BlockConnect (); // デフォルトコンストラクタ

#### BlockConnectクラスコンストラクタ

BlockConnectクラスのコンストラクタです。ブロックの接続関係として有向エッジのtail(尾)とhead(頭)を指定するコンストラクタとデフォルトコンストラクタがあります。

BlockConnect (BlockInfo\* tail, BlockInfo\* head); // エッジの指定

BlockConnect (); // デフォルトコンストラクタ

このコンストラクタは、ReaderクラスでXMLファイルを入力する際に使用されるもので、通常ユーザが使用することはありません。

#### var\_list()

ブロック間で伝播する変数の<var>タグのポインタのvectorを返します。

const var\_list\_T& var\_list (void) const; //状態参照

var\_list\_T& var\_list (void); //状態変更

#### head()

有向エッジのhead(頭)のBlockInfoのポインタを返します。

BlockInfo\* head (void) const; //状態参照

BlockInfo\*& head (void); //状態変更

const BlockInfo\* tail (void) const

#### tail()

有向エッジのtail(尾)のBlockInfoのポインタを返します。

BlockInfo\* tail (void) const; //状態参照

BlockInfo\*& tail (void); //状態変更

#### オペレータ(==)

同じ有向エッジであるかを判定します。比較する二つのBlockConnectクラスで、tailとheadがそれぞれ一致する場合はtrueを返し、そうでない場合はfalseを返します。

bool operator== (const BlockConnect& bc) const

### codelistクラス

ブロックの<block>タグにある<code>タグに格納されているコードを、SubsystemのコードやCのブロック構造などのコードに分離するためのクラスです。コードは次のような定形のコメントにより分割されています。

/\*(<<<|>>>) [ESX] [0-9]+ (block\_name|”none”) (<<<|>>>)\*/

|  |  |  |
| --- | --- | --- |
| field | 値 | 説明 |
| 1 | <<< | 当該コードの開始マーク |
| >>> | 当該コードの終了マーク |
| 2 | S | コードは他のブロックの先頭(Start)のコード  (例えばTriggered Subsystemの場合はif文) |
| E | コードは他のブロックの終端(End)のコード  (例えばTriggered Subsystemの場合は`}’) |
| X | 他のブロックの切り替え情報 |
| 3 | 正数 | 出現順のシーケンス番号 |
| 4 | ブロック名または”none” | コードの出自(ブロック名。ブロックの一部でなければ”none”) |
| 5 | 1と同じ |  |

例えば<code>タグのコードでは以下のようにコードが混在しているため、当該ブロックのコードのみを取り出す場合、コード内のコメントを元にコードを分離する必要があります。

/\*<<< S 1 none <<<\*/

(共通の先頭コード)

/\*>>> S 1 none >>>\*/

/\*<<< S 2 OTHER <<<\*/

(OTHERの先頭コード)

/\*>>> S 2 OTHER >>>\*/

(当該ブロックのコード)

/\*<<< E 2 OTHER <<<\*/

(OTHERの終端コード)

/\*>>> E 2 OTHER >>>\*/

/\*<<< E 1 none <<<\*/

(共通の終端コード)

/\*>>> E 1 none >>>\*/

共通の先頭コードには、自動変数等の定義が含まれ、OTHERのブロックや当該ブロックのコードの実行のために必要な場合があります。またOTHERのブロックはSubsystemのコードが含まれる場合があります。

#### codelistクラスコンストラクタ

codelistクラスのコンストラクタです。コードを分離する際に使用します。char型のポインタで、処理範囲の開始(beg)と終端(end)を指定するコンストラクタとstringを指定するコンストラクタ、それとデフォルトコンストラクタがあります。デフォルトコンストラクタを使用した場合は、7.2.5.2のprocess()を呼び出す必要があります。

codelist (const char\* beg, const char\* end); // コードの開始終了を指示

codelist (const ::std::string& str); // stringを指定

codelist() // デフォルトコンストラクタ

#### process()

デフォルトコンストラクタを使用した場合、分離するコードの範囲の開始(beg)と終端(end)を指定して本メンバ関数を呼び出す必要があります。

void process (const char\* beg, const char\* end);

#### mark\_list()

分離されたコードの情報を保持するc\_markクラスのvectorを返します。

const marklist\_T& mark\_list (void) const; //状態参照

marklist\_T& mark\_list (void); //状態変更

c\_markクラスについては7.2.6を参照してください。

### c\_markクラス

7.2.5のcodelistクラスにより、分離されたコードの情報を保持するクラスです。

#### c\_markクラスコンストラクタ

c\_markクラスのコンストラクタです。コードの開始、終了、コメントを含む開始、コメントを含む終了、開始・終了の識別、番号、ブロック名を指定します。

c\_mark (const char\* b, const char\* e, const char\* tb, const char\* te,

char t, unsigned int i, const ::std::string& n)

このコンストラクタは、codelistクラスでコードを分離する際に使用されるもので、通常ユーザが使用することはありません。

#### set()

コードの開始、終了、コメントを含む開始、コメントを含む終了、開始・終了の識別、番号、ブロック名を設定します。

void set (const char\* b, const char\* e, const char\* tb, const char\* te,

char t, unsigned int i, const ::std::string& n);

この関数は通常ユーザが使用することはありません。

#### text\_beg()

分離されたコードの開始アドレスを返します。

const char\* text\_beg (void) const; //状態参照

const char\*& text\_beg (void); //状態変更

#### text\_end()

分離されたコードの終了アドレスを返します。

const char\* text\_end (void) const; //状態参照

const char\*& text\_end (void); //状態変更

#### beg()

定形のコメントを含む分離されたコードの開始アドレスを返します。前のc\_markのend()と比較することで、完全に連続しているかそうでないかを判定する場合などに使用します。

const char\* beg (void) const; //状態参照

const char\*& beg (void); //状態変更

#### end()

定形のコメントを含む分離されたコードの終了アドレスを返します。後のc\_markのbeg()と比較することで、完全に連続しているかそうでないかを判定する場合などに使用します。

const char\* end (void) const; //状態参照

const char\*& end (void); //状態変更

#### type()

分離されたコードが先頭か終端かの種別を返します。先頭のコードであれば`S’を、終端であれば`E’を返します。

char type (void) const; //状態参照

char& type (void); //状態変更

#### num()

分離されたコード中のブロックの出現順番を返します。通常使う必要はありません。

unsigned int num (void) const; //状態参照

unsigned int& num (void); //状態変更

#### name()

このコードに対応するブロック名を返します。当該ブロックと一致しない場合は、当該ブロックのコードではなく上位のSubsystemやRTW-ECが生成する自動変数を持つCのブロックを意味します。

const ::std::string& name (void) const; //状態参照

::std::string& name (void); //状態変更

#### キャスト(string)

分離されたコードのbeg(), end()の範囲をstringに変換するキャストです。一般に分離されたコードを簡単に利用する際には、このキャストを使用し、stringへ変換します。

operator ::std::string () const;

### const\_block\_list\_Tクラス

書き換え不可能なBlockInfoのポインタのリスト(vector)を扱うクラスです。

::std::vector < const BlockInfo \*>を継承します。

#### const\_block\_list\_Tクラスコンストラクタ

const\_block\_list\_Tクラスのコンストラクタです。このクラスは、複数のBlockInfoの情報を保持したい場合に使用します。

const\_block\_list\_T (size\_t size, const BlockInfo\* b) // サイズ・初期値指示

const\_block\_list\_T () // デフォルトコンストラクタ

あらかじめ扱うBlockInfoの数が決まっている場合や、配列としてアクセスする場合は、サイズと初期値(通常はNULL)を指定します。

#### checkinfo()

指定したBlockInfoのポインタが既に登録されているかどうかをチェックします

bool checkinfo (const BlockInfo\* b) const;

#### any()

任意に利用可能なフラグを参照します。

unsigned long any (void) cons; //状態参照

unsigned long& any (void); //状態変更

#### addinfo()

指定したBlockInfoのポインタを追加します

void addinfo (const BlockInfo\* b);

#### オペレータ(+=)

指定したbinfoを追加します。

const BlockInfo\* operator+= (const BlockInfo\* b)

### 関数

#### find\_multirate\_subsystem()

BlockInfo\* find\_multirate\_subsystem (const BlockInfo\* binfo);

指定したBlockInfoの上位にTriggeredまたはEnabled Subsystemがある場合は、そのSubsystemのBlockInfoのポインタを返します。そうでない場合はNULLを返します。

二つのサブブロックに対する本関数の戻り値が異なる場合、それらのブロックは一般に、異なるレートで動作すると考えられます。

## SimulinkRTW(SimulinkモデルとRTW-ECのコード処理)

このモジュールは以下の名前空間のクラスと関数を提供します。

namespace Simulink

{

namespace RTW {

各クラス、関数はこの名前空間に置かれる

}

}

このモジュールを使用し、Simulinkモデルのブロック構造の入力と、RTW-ECが生成するCコード、ヘッダファイル、パラメタ・定数パラメタの初期値ファイルを入力して以下の内容を抽出・生成し、それらをXMLの情報として出力することができます。

* Simulinkモデルのブロックに対応する実行コードと変数や状態変数の初期化コード
* Simulinkモデルのブロックで入出力する変数
* Simulinkモデルのブロックで使用する状態変数、パラメタ、定数パラメタ
* Simulinkモデルのブロックで使用するパラメタと定数パラメタの初期値
* 実行コードを持つSimulinkモデルのブロックおよび最外のInport/Outportのブロック間の相互接続関係と、その接続で入出力される変数

このモジュールについては、入出力処理に利用する関数の使い方について説明します。

#### 入力処理関数

Simulinkモデルのブロック構造、RTW-ECが生成するコードを読み込む処理です。

::std::auto\_ptr <Blocks >

parse\_all\_code (const char\* xml, const char\* header,

const char\* source,const char\* data, bool validation);

入力処理関数引数には、以下のファイルを指定します。

表 7‑6 SimulinkRTW入力処理関数引数一覧

|  |  |  |
| --- | --- | --- |
| 引数 | 内容 | 省略有無 |
| xml | Simulinkモデルのブロック構造を抽出したXMLファイル | 必須 |
| header | RTW-ECが生成したモデルのヘッダファイル | 必須 |
| source | RTW-ECが生成したモデルの実行コード | 必須 |
| data | RTW-ECが生成したモデルのパラメタ・定数パラメタの初期値 | 必須 |
| validation | BLXMLスキーマのチェックを実行するかどうか | 必須 |

モデルがパラメタ・定数パラメタを使用しないものであれば、dataは省略可能です。

エラー時は、auto\_ptrにNULLポインタが入ります。また、Blocksが利用可能かはboolのキャストがあるので、if文の条件判定でオブジェクトを評価すればテストできます。

::std::auto\_ptr <Blocks > blocks (parse\_all\_code ("model.xml",

"model.h",

"model.c",

"model\_data.c",

false);

if (!\*blocks) { // blocksは利用不可。(blocks.get()==NULL)でも同様

::std::cerr << "Error";

}

#### XML出力

入力処理を行った後、XMLに出力する処理です。出力先に応じて三つの関数があります。

void Blocks::out\_xml (::std::ostream &os);

void Blocks::out\_xml (const char\* file);

void Blocks::out\_xml (void);

表 7‑7 SimulinkRTW XML出力関数引数一覧

|  |  |
| --- | --- |
| 引数 |  |
| os(出力ストリーム) | 指定した出力ストリームにXMLを出力 |
| file(ファイル名) | 指定したファイルにXMLを出力 |
| なし | 標準出力にXMLを出力 |

#### 使用例

以下は簡単な使用例です(パラメタのエラーチェックは省略しています)。

int main (int argc, char\* const\* argv)

{

int c;

const char\* xml\_input = NULL;

const char\* xml\_output = NULL;

const char\* c\_header = NULL;

const char\* c\_source = NULL;

const char\* c\_data = NULL;

while ((c = getopt (argc, argv, “h:c:d:”)) != -1) {

switch (c) {

case 'h': c\_header = optarg; break; // Cヘッダー

case 'c': c\_source = optarg; break; // Cソース

case 'd': c\_data = optarg; break; // パラメタ初期値ファイル

}

}

xml\_input = argv[optind]; // 入力XML

xml\_output = argv[optind+1]; // 出力XML

// 入力実行

::std::auto\_ptr< Blocks >

bn (parse\_all\_code (xml\_input, c\_header, c\_source, c\_data));

if (!\*bn) return 1; // エラー

// XML出力

if (xml\_output)

bn->out\_xml (xml\_output);

else

bn->out\_xml ();

return 0;

}

## blxml2code

このプログラムは以下のクラスを有します。

表 7‑8 blxm2codeのクラス一覧

|  |  |  |
| --- | --- | --- |
| クラス名 | 内容 | |
| block\_attr | <block>のタグに含まれているブロックの属性や処理に必要な情報を保持するクラス | |
| コンストラクタ | 説明 |
| block\_attr(upper, block, subsystem) | 上位のSubsystem(upper)と<block>タグ、サブシステムかどうかを指定したコンストラクタ |
| 公開メンバ | 説明 |
| id() | ブロックのID(0以上)を返す |
| block() | <block>タグのポインタを返す |
| ignore() | 処理対象でなければtrueを返す |
| set\_ignore(bool) | 処理対象でないか、そうでないか設定する |
| upper() | 上位のSubsystemのblock\_attrのポインタを返す |
| lowers() | Subsystemの下位のブロックのblock\_attrのポインタのvectorを返す |
| merged() | SwitchCase/Ifの場合、マージされている<block>のポインタのdequeリストを返す |
| キャスト | 説明 |
| (bool) |  |
| attr\_map | <block>のアドレスをキーとし、block\_attrを値とした連想配列クラスを継承したクラス | |
| 公開メンバ | 説明 |
| get(block) | <block>のポインタに対応するblock\_attrのポインタを返す |
| get\_id (block) | <bloc>のポインタに対応するブロックのIDを返す |
| add (upper, block, subsystem) | 上位のSubsystem(upper)と<block>タグ、サブシステムかどうかを指定し、連想配列に追加する |
| BLXML | BLXMLの情報を元に、コード生成を行う本体のクラス | |
| コンストラクタ | 説明 |
| BLXML(istream) | XMLファイルの入力ストリームを指定したコンストラクタ |
| BLXML(blocks) | <blocks>タグのauto\_ptrを指定したコンストラクタ |
| BLXML() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| load(istream) | デフォルトコンストラクタを使用した場合、XMLの入力ストリームを指定した入力処理を行う |
| gencode(ostream) | コード生成を行い、出力ストリームへ出力する |
| check\_switchcase\_if\_block() | SwitchCase/Ifブロックの処理を実行する |
| dump() | デバッグのためBLXMLの内容をダンプする(未実装) |
| キャスト | 説明 |
| (bool) | XMLが読み込まれており、処理可能かを確認する。処理可能であればtrueが返る |

表 7‑9 blxml2codeの型一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| block\_map | ブロック名をキーとし<block>タグのポインタを値とした連想配列 |
| block\_map\_iterator | 上記の状態変更用イテレータ |
| block\_map\_const\_iterator | 上記の状態参照用イテレータ |
| block\_stack | <block>タグのポインタのdequeリスト |
| block\_stack\_const\_iterator | 上記の状態参照用イテレータ |
| sig\_list | <signal>/<state>/<param>/<const\_param>等のポインタのvector |
| sig\_list\_iterator | 上記の状態変更用イテレータ |
| sig\_list\_const\_iterator | 上記の状態参照用イテレータ |
| var\_map | 変数名をキーとし、任意のポインタを値とした連想配列(任意のポインタは再利用しない) |
| var\_map\_iterator | 上記の状態変更用イテレータ |
| var\_map\_const\_iterator | 上記の状態参照用イテレータ |
| var\_map\_res | var\_mapへのinsert時の結果を確認するための型 |
| attr\_list | block\_attrのvector |

プログラムの処理の流れは以下の通りです。

BLXML blxml;

::std::istream input;

::std::osgream output;

blxml.load (input);

blxml.check\_switchcase\_if\_block ();

blxml.gencode (output);

#### block\_attrクラスコンストラクタ

block\_attrクラスのコンストラクタです。上位Subsystemのblock\_attrのポインタ、当該ブロックの<block>のポインタ、Subsystemかどうかのboolean値を指定します。

block\_attr (block\_attr\* u, const block\_T\* b, bool sb);

このコンストラクタはBLXML::load()の中で実行されます。

#### block\_attr ::id()

当該ブロックのIDを返します。IDはBLXMLクラスで内部のvectorへのアクセスに使用されます。

int id (void) const;

#### block\_attr ::block()

ブロックの<block>のポインタを返します。

const block\_T\* block (void) const;

#### block\_attr ::ignore()

当該ブロックのコード生成をしない場合はtrueを返します。この場合、このブロックはSwitchCaseまたはIfブロックの一部です。

bool ignore (void) const;

#### block\_attr ::set\_ignore()

当該ブロックのコード生成をしない場合はtrueを設定します。そうでない場合はfalseを設定します。SwitchCaseまたはIfブロックの一部であればtrueを設定します。

void set\_ignore (bool i);

#### block\_attr ::lowers()

ブロックがSubsystemの場合、含まれるブロックのblock\_attrのポインタをvectorとして返します。下位ブロックがSubsystemの場合、その内部のブロックは含まれません。

attr\_list& lowers (void);

#### block\_attr ::merged

ブロックがSwitch/Ifブロックの場合、含まれるブロックの<block>のポインタがdequeリストとして返されます。ブロックがSubsystemの場合、その内部のブロックが含まれます。

const block\_stack& merged (void) const; // 状態参照

block\_stack& merged (void); // 状態変更

#### キャスト(bool)

当該ブロックのコード生成をしない場合はtrueを返します。この場合、このブロックはSwitchCaseまたはIfブロックの一部です。

operator bool() const

#### attr\_map ::get

ブロックの<block>のポインタを指定し、該当するblock\_attrのポインタを取得します。ブロックが存在しない場合はNULLを返します。

const block\_attr\* get (const block\_T\* b) const; // 状態参照

block\_attr\* get (const block\_T\* b); // 状態変更

#### attr\_map::get\_id

ブロックの<block>のポインタを指定し、該当するblock\_attrのIDを返します。ブロックが存在しない場合は-1を返します。

int get\_id (const block\_T\* b) const;

#### attr\_map::add()

ブロックを登録します。上位Subsystemのblock\_attrのポインタ、当該ブロックの<block>のポインタ、Subsystemかどうかのboolean値を指定します。

block\_attr& add (block\_attr\* u, const block\_T\* b, bool sb)

この関数はBLXML::load()で使用されます。

#### BLXMLクラスコンストラクタ

BLXMLクラスのコンストラクタです。XMLの入力ストリームを指定するか、入力したXMLファイルの<blocks>のポインタのauto\_ptrを渡すコンストラクタがあります。デフォルトのコンストラクタを使用した場合はload()を実行する必要があります。

BLXML (::std::istream& is);

BLXML (::std::auto\_ptr < blocks\_T> b);

BLXML ();

#### BLXML::load()

ブロックレベル構造XMLを読み込む処理です。XMLの入力ストリームを指定します。

デフォルトのコンストラクタを使用した場合は、この関数を呼び出す必要があります。

int load (::std::istream& is);

#### BLXML::gencode()

Cコードを生成する関数です。指定した出力ストリームにCコードを出力します。

int gencode (::std::ostream& os) const;

#### BLXML::check\_switchcase\_if\_block()

ブロックレベル構造XMLのSwitchCaseブロックまたはIfブロックに含まれるブロックの情報を記録し、出力しないブロックにマークを設定します。この結果SwitchCase/Ifブロックには含まれるブロックの入出力変数がマージされ、マージされたブロックの関数は出力されません。

void check\_switchcase\_if\_block (void);

#### BLXML::dump()

BLXMLクラスの内部をダンプする関数です。現在実装はなく本体は空です。

void dump (void) const

#### キャスト(bool)

ブロックレベル構造XMLが読み込まれ、利用可能な状態であればtrueが返ります。

operator bool () const

## llvm-BLXMLPerf

LLVM-BLXMLPerfモジュールは、LLVMのPassですが、LLVMが使用するgccのコンパイルオプションは他のモジュールのコンパイルに対し問題があり、SHIMやブロックレベル構造XML、boostのライブラリ等を含むコードのコンパイルができません。そのため、使用するそれぞれのモジュールはインターフェースを定義したヘッダファイルをそれぞれ参照し、コードを分離し、それぞれ独立にコンパイルして、最終的にリンクする、という構成を取っています。

これらはnamespaceの中で統合されています。

namespace Simulink

{

namespace LLVM {

各クラス、関数はこの名前空間に置かれる

}

}

表 7‑10 llvm-BLXMLPerfのクラス一覧

|  |  |  |
| --- | --- | --- |
| クラス | 説明 | |
| FunctionInfo | 性能計測対象の関数の情報(best/worst/typical)を保持するクラス | |
| コンストラクタ | 説明 |
| FunctionInfo(block, best, typical, worst) | ブロック名とbest/typical/worstを指定したコンストラクタ |
| FunctionInfo() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| name() | ブロック名を返す |
| best() | 計測のベスト値を返す |
| typical() | 計測のtypical値を返す |
| worst() | 計測のワースト値を返す |
| BLXMLPerf | LLVMのFunctionPassを継承した計測用のPassのクラス | |
| コンストラクタ | 説明 |
| BLXMLPerf() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| doInitialization(m) | LLVMのPassのdoInitializaion() |
| doFinalization(m) | LLVMのPassのdoFinalization() |
| runOnFunction(f) | LLVMのPassのrunOnFunction() |
| getAnalysusUsage(au) | LLVMのPassのgetAnalysusUsage() |

表 7‑11 llvm-BLXMLPerfの型一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| FunctionInfoList | FunctionInfoのvector |
| functioninfo\_iterator | 上記の状態変更用イテレータ |
| functioninfo\_const\_iterator | 上記の状態参照用イテレータ |

#### FunctionInfoクラスコンストラクタ

FunctionInfoクラスのコンストラクタです。FunctionInfoは関数単位にベスト/typical/ワーストのサイクル数を保持します。関数名を指定し、各サイクル数を指定したコンストラクタとデフォルトコンストラクタがあります。デフォルトコンストラクタを使用した場合は、各値を別途設定する必要があります。

FunctionInfo (::std::string n,

InsnCycle b = 0, InsnCycle t = 0, InsnCycle w = 0);

FunctionInfo ()

このコンストラクタはBLXMLPerfクラスから呼ばれます。

#### FunctionInfo ::name()

関数名にアクセスします。

const ::std::string& name () const; // 状態参照

::std::string& name (); // 状態変更

#### FunctionInfo ::best()

関数のサイクルのベスト値にアクセスします。

InsnCycle best () const; // 状態参照

InsnCycle& best (); // 状態変更

#### FunctionInfo ::typical()

関数のサイクルのtypical値にアクセスします。

InsnCycle typical () const; // 状態参照

InsnCycle& typical (); // 状態変更

#### FunctionInfo ::worst()

関数のサイクルのワースト値にアクセスします。

InsnCycle worst () const; // 状態参照

InsnCycle& worst (); // 状態変更

#### BLXMLPerfクラスコンストラクタ

BLXMLPerfクラスのコンストラクタです。

BLXMLPerf();

このコンストラクタは、LLVMのoptから呼ばれます。

#### BLXMLPerf ::doInitialization()

LLVMのPassのdoInitialization()関数です。LLVMのoptから呼ばれます。

virtual bool doInitialization (::llvm::Module& m);

#### BLXMLPerf ::doFinalization()

LLVMのPassのdoFinalization ()関数です。LLVMのoptから呼ばれます。

virtual bool doFinalization (::llvm::Module& m);

#### BLXMLPerf ::runOnFunction()

LLVMのPassのrunOnFunction ()関数です。LLVMのoptから呼ばれます。

virtual bool runOnFunction (::llvm::Function& f);

#### BLXMLPerf ::getAnalysusUsage()

LLVMのPassのgetAnalysisUsage ()関数です。LLVMのoptから呼ばれます。

virtual void getAnalysisUsage (::llvm::AnalysisUsage& au) const;

### boost\_llvm

このモジュールではboostの名前空間を拡張し、以下の型を追加します。

表 7‑12 namespace boostの型追加一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| vertex\_current\_cycle\_t | 現在のcycle数を保持するためのvertexのプロパティ用enumerate |
| vertex\_best\_cycle\_t | ベストのcycle数を保持するためのvertexのプロパティ用enumerate |
| vertex\_typical\_cycle\_t | typicalのcycle数を保持するためのvertexのプロパティ用enumerate |
| vertex\_worst\_cycle\_t | ワーストのcycle数を保持するためのvertexのプロパティ用enumerate |
| vertex\_basicblock\_t | 基本ブロックのポインタを保持するためのvertexのプロパティ用enumerate |
| vertex\_average\_cycle\_t | 平均のcycle数を保持するためのvertexのプロパティ用enumerate |
| vertex\_type\_t | ノードの種別を保持するためのvertexのプロパティ用enumerate |
| edge\_best\_cycle\_t | ベストのサイクルを保持するためのedgeのプロパティ用enumerate |
| edge\_typical\_cycle\_t | typicalのサイクルを保持するためのedgeのプロパティ用enumerate |
| edge\_worst\_cycle\_t | ワーストのサイクルを保持するためのedgeのプロパティ用enumerate |
| edge\_type\_t | エッジの種別を保持するためのedgeのプロパティ用enumerate |

表 7‑13 boost\_llvmの型一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| InsnCycle | 命令のサイクル数の型(浮動小数点型) |
| edge\_property | boostのグラフのedgeのプロパティ |
| vertex\_property | boostのグラフのvertexのプロパティ |
| graph\_property | boostのグラフのgraphのプロパティ |
| Graph | グラフの型 |
| Vertex | 頂点の型 |
| Edge | エッジの型 |
| EdgeIterator | エッジのイテレータ |
| VertexIterator | 頂点のイテレータ |
| OutEdgeIterator | 外向きエッジのイテレータ |
| InEdgeIterator | 内向きエッジのイテレータ |
| VertexPair | 頂点のpairの型 |
| VertexPairList | 頂点のペアのvectorの型 |

### shim\_llvm

このモジュールではSHIMにアクセスするためのnamespaceとクラスを提供します。

namespace SHIM

{

class SystemConfiguration;

class ComponentSet;

namespace XML {

各クラス、関数はこの名前空間に置かれる

}

}

表 7‑14 shim\_llvmのクラス一覧

|  |  |  |
| --- | --- | --- |
| クラス | 説明 | |
| Latency | 命令のlatencyを保持するクラス | |
| コンストラクタ | 説明 |
| Latency(best,typical,worst) | best/typical/worstを指定したコンストラクタ |
| 公開メンバ | 説明 |
| best() | ベスト値にアクセスする |
| typical() | typical値にアクセスする |
| worst() | ワースト値にアクセスする |
| File | SHIMのXML情報にアクセスするクラス | |
| コンストラクタ | 説明 |
| File(file, component, master, instset) | SHIMファイル、ComponentSet名、MasterComponent名、CommonInstructionSet名を指定したコンストラクタ |
| File() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| latencies() | 読み込まれた連想配列(latency\_map)を返す |
| latency() | 命令に対するLatencyを返す |
| ComponentSet() | SHIMのComponentSet名にアクセスする |
| MasterComponent() | SHIMのMasterComponent名にアクセスする |
| CommonInstructionSet() | SHIMのCommonInstructionSet名にアクセスする |
| キャスト | 説明 |
| (bool) | SHIMが読み込まれているかどうか確認する |
| (SysmtemConfiguration\*) | SHIMの<SystemConfiguration>のポインタに変換する |

表 7‑15 shim\_llvmの型一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| latency\_map | 命令の名前をキーとし、Latencyを値とした連想配列 |
| latency\_map\_iterator | 上記の状態変更用イテレータ |
| latency\_map\_const\_iterator | 上記の状態参照用イテレータ |

#### Latencyクラスコンストラクタ

Latencyクラスのコンストラクタです。SHIMの<performance>から取得されるベスト、typical、ワーストの値指定します。

Latency (double b = 0, double t = 0, double w = 0);

#### Latency ::best()

ベスト値にアクセスします。

const double& best (void) const; // 状態参照

double& best (void); // 状態変更

#### Latency ::typical()

typical値にアクセスします。

const double& typical (void) const; // 状態参照

double& typical (void); // 状態変更

#### Latency ::worst()

ワースト値にアクセスします。

const double& worst (void) const; // 状態参照

double& worst (void); // 状態変更

#### Fileクラスコンストラクタ

Fileクラスのコンストラクタです。SHIMのファイル名、ComponentSet名、MasterComponentSet名、CommonInstructionSet名とSHIMスキーマを指定するコンストラクタです。stringは空文字列だと省略しているとみなされます。schemaにNULLを指定するとスキーマを省略したとみなされます。

File (const char\* file, const ::std::string& component,

const ::std::string& master, const ::std::string& instset,

const char\* schema);

#### File ::latencies()

命令の名前とLatencyの連想配列を返します。

const latency\_map& latencies (void) const; // 状態参照

latency\_map& latencies (void); // 状態変更

#### File ::latency()

命令の名前に対するLatencyのポインタを返します。命令がない場合はNULLが返ります。

const Latency\* latency (const ::std::string &name) const; // 状態参照

Latency\* latency (const ::std::string &name); // 状態変更

#### File ::ComponentSet()

処理に使用したSHIMのComponentSet名を返します。

const ::std::string& ComponentSet (void) const;

#### File ::MasterComponent()

処理に使用したSHIMのMasterComponent名を返します。

const ::std::string& MasterComponent (void) const;

#### File ::CommonInstructionSet()

処理に使用したSHIMのCommonInstructionSet名を返します。

const ::std::string& CommonInstructionSet (void) const;

#### キャスト(bool)

SHIMを読み込み、利用可能であればtrueを返します。

operator bool () const;

#### キャスト(::SHIM::SysmtemConfiguration\*)

SHIMの<SystemConfiguration>タグのポインタを返します。

operator const ::SHIM::SystemConfiguration\* () const;

### blxml\_llvm

このモジュールではブロックレベル構造XMLにアクセスするためのnamespaceとクラスを提供します。

namespace SimulinkModel

{

namespace BLXML {

各クラス、関数はこの名前空間に置かれる

}

}

表 7‑16 blxml\_llvmのクラス一覧

|  |  |  |
| --- | --- | --- |
| クラス | 説明 | |
| BLXML | ブロックレベル構造XMLにアクセスするためのクラス | |
| コンストラクタ | 説明 |
| BLXML(file, validation) | 入力XMLファイルを指定したコンストラクタ。validationは省略可で省略時はtrue |
| BLXML(istream, validation) | 入力XMLファイルの入力ストリームを指定したコンストラクタ。validationは省略可で省略時はtrue |
| BLXML(blk) | <blocks>のポインタ(auto\_ptr)を指定したコンストラクタ |
| BLXML() | デフォルトコンストラクタ |
| 公開メンバ | 説明 |
| set\_performance() | <block>タグに、指定した値で<performance>タグを追加する |
| set\_filename() | 入力したXMLファイルの情報を<blocks>のタグに<file>タグとして追加する |
| set\_shim\_param() | 入力したSHIMの情報を<blocks>のタグに<shim-params>として追加する |
| output() | XML出力処理 |
| キャスト | 説明 |
| (bool) | ブロックレベル構造XMLが利用可能か確認する |
| (blocks\_T\*) | <blocks>のポインタに変換するキャスト |

表 7‑17 blxml\_llvmの型一覧

|  |  |
| --- | --- |
| 型 | 説明 |
| file\_type | SHIM、SHIMのスキーマ、ブロックレベル構造XMLを識別する型 |
| element\_type | ComponentSet, MasterComponent, CommonInstructionSetの種別 |

#### BLXMLクラスコンストラクタ

BLXMLクラスのコンストラクタです。ブロックレベル構造XMLのファイル名、入力ストリーム、ありは読み込み済みの<blocks>のポインタのauto\_ptrを指定することができます。

BLXML (const char\* file);

BLXML (::std::istream& is);

BLXML (::std::auto\_ptr < ::SimulinkModel::XSD::blocks\_T > b);

#### set\_performance()

nameで指定したブロックの<block>タグに<performance>タグを追加します。typeにはtask/update/initのいずれかを指定します。

void set\_performance (const ::std::string& name,

const ::std::string& type,

double best, double typical, double worst);

#### set\_filename()

XMLの<blocks>タグにnameで指定したファイルの情報を<file>タグを追加します。

void set\_filename (file\_type type, const ::std::string& name);

#### set\_shim\_param()

処理したSHIMのパラメタを<shim-params>タグとしてXMLの<blocks>タグに追加します。element\_typeにはSHIMのタグの種別、nameにはComponentSet等の名前を指定します。

void set\_shim\_param (element\_type type, const ::std::string& name);

#### output()

性能情報付きのXMLファイルを出力します。出力ストリームか、ファイル名を指定します。

int output (::std::ostream& os);

int output (const char\* file);

#### キャスト(bool)

ブロックレベル構造XMLファイルが読み込まれ、利用可能であればtrueが返ります。

operator bool () const;

#### キャスト(blocks\_T\*)

ブロックレベル構造XMLのトップレベルの<blocks>タグのポインタに変換します。

operator const ::SimulinkModel::XSD::blocks\_T\* () const;

### Passの処理概要

本Passは次のようになっています

1. FunctionPassを継承しており、関数単位に処理を行う
2. 事前に LoopInfo Passと ScalarEvolution Passを実行する
3. 初期化時にSHIMとブロックレベル構造XMLを読み込む
4. 終了時に性能付きブロックレベル構造XMLを出力する

#### 関数単位の処理

関数単位の処理は以下のようになっています。

1. 基本ブロックのポインタをキーとした連想配列を作成(付随情報の保持のため)
2. 基本ブロックのサイクル数(best/typical/worst)を算出し、連想配列に保存
3. LoopInfoとScalarEvolutionを元にループの処理を実施
4. 基本ブロックのsuccessorおよび基本ブロックをまとめたループの情報 (LoopInfoによる連想配列)から、有向グラフを作成する(boostのグラフライブラリを使用する)
5. 最短経路/best、最長経路/worst、分岐確率50%/typicalによる経路探索を実施し、関数のサイクル数(best/typical/worst)を算出し、関数の情報として保存

#### ループの処理

ループの処理は次のように行います

1. LoopInfo最内ループから処理するよう再帰処理する
2. ScalarEvolutionの結果から固定回回るループなら、ループとして基本ブロックを一つのグラフとして、ループ内の基本ブロックのサイクル数を最短経路/best, 分岐確率50%/typical、最長経路/worst別に算出、best/worst/typicalともループ回数分加算する
3. 最大回数がわかるループ(途中で抜け出すことがある)なら、ループとして基本ブロックを一つのグラフとして、ループ内の基本ブロックのサイクル数をbest/typical/worst別に加算、worstだけループ回数分加算する。best/typicalは一回だけ回るものとする。
4. 上記のループの基本ブロックの情報をLoopInfoのポインタをキーとした連想配列に保存する
5. 上記いずれでもないループは、基本ブロックをループとしてまとめない (つまり、一回だけ回るものとして取り扱うことになる)

#### ループ内の基本ブロック

ループ内の基本ブロックも関数全体についてもグラフ探索は次のように行います。

1. best値をエッジに距離として設定しdijkstraのアルゴリズムで最短経路探索を行う(boostのアルゴリズムを使用)
2. worst値の負数をエッジに距離として設定し、bellman-fordのアルゴリズムで最短経路探索を行う(結果として最長経路になる)。(boostのアルゴリズムを使用)
3. 分岐確率50%(同確率分岐)のtypicalの探索は、再帰的なdepth firstで行う。ループを検出(GRAY)した場合は探索を打ちきる(つまりその分岐は当たらない)。到達済みノード(BLACK)であれば、下位ノードの計算済みのサイクルを加算して行く。最後に分岐数で割ってサイクル数とする

# 制限事項と課題

## 制限事項

現在の制限事項は以下の通りです

1. global stateには対応していません。なお、これはこれまでトヨタ様対応にも含まれていないものです
2. step関数で自動変数が使用されている場合、全てのブロックの実行文に、それら全ての自動変数含まれます。
3. initialize関数で自動変数が使用されている場合、全てのブロックの初期化コードに、それら全ての自動変数含まれます。
4. initialize関数でSimulinkブロックの識別がない外部変数の初期化コードは単一変数または配列要素の初期化、ループによる配列の初期化、memset()による初期化に対応しています。変数名を識別する都合上、より複雑な構文による初期化コードがある場合、正しいコード抽出が行えない場合があります
5. ブロック変数が構造体メンバになっているコードには対応していません。全て外部変数にしてください
6. 変数の型が構造体等の場合、Simulinkブロックに固有の型が定義されている場合はその構造体を抽出しますが、それ以外ではその構造体の情報は持ちません。変数の型が基本型あるいはSimulinkブロックに固有の型以外の場合、その型はコンパイル時にインクルードするヘッダファイルに記述する必要があります。
7. Demuxは p=n で、配列サイズと分割数が一致するもののみ取り扱います。また、複数のスカラ変数を分離するモデルには対応していません。Mux/Demuxは信号のバスとして扱います。また、最終的にDemuxの入力がベクタである場合のDemuxの出力は、p=nおよびp<nの既定動作に従ったベクタの要素となり、Muxの入力がベクタの要素である場合のMuxの出力は、ベクタになります。なお、Mux/Demuxがそれぞれ多段で組合せが非対称の場合には対応できません。
8. SimulinkのモデルのブロックのうちInport/Outport/Demux/Mux/From/Gotoについては接続関係を認識、それらをまたがる変数の伝播を追跡します。Scope/Workspace等は追跡を打ち切ります
9. Constantは独立したブロックとして取り扱います(他のブロックと結合しません)
10. ブロックレベル構造XMLにないブロックで、RTW-ECが生成したコードにあるブロックは、無視します。ただし、VectorConcatanateがSignalConversionになるケースと何らかの理由によりTmp SignalConversionAt…になるケースは対応済みです。
11. SwitchCase/IfブロックまたはAtomic/Triggered/Enabled/Iterator Subsystemの抽出は最上位のものに対して行います。SwitchCase/IfやAtomic/ Triggered/Enabled/Iterator SubsystemがさらにSwitchCase/IfやAtomic/ Triggered/Enabled/Iterator Subsystemを含む場合、内部のSwitchCase/IfやAtomic/ Triggered/Enabled/Iterator Subsystemのコードは最も上位のSwitchCase/IfあるいはAtomic/ Triggered/Enabled/Iterator Subsystemのブロックと一体で抽出されます
12. 単一のforループ内に複数のブロックが含まれる場合は、それぞれのブロックにループ構造を結合して切り出します。なお二重ループなどには対応していません。

## 課題

1. 制限事項4はSimulinkブロックの識別がなく、より複雑な初期化コードがある場合、そのようなコードパタンに対応する必要があります
2. 制限事項7についてはDemux/Muxによる配列の分割・結合を取り扱う必要が考えられますMux/Demuxの設定情報をBLXMLに含むようにして、処理はそれに従うようにすることが必要です
3. 制限事項8ではInport/Outport/Demux/Mux/From/Gotoで、信号やポートを切り替えて追跡しますが、それ以外のブロックではInput/Outputの情報のみを使用します。Inport/Outportのように信号やポートを切り替えるたり、Demux/Muxのように変数に変更を加えるような、他に考慮すべきブロックがないか確認・検討が必要です。
4. 制限事項10は自動的に作られるブロックについては、モデルに明示的にそのブロックを追加するというのが、望ましい解決ですが、それができない場合は、コード付きブロックレベル構造XMLファイルに、ブロックを自動追加するような対応が必要です