
SWORDSフレームワーク 解説&使い方



SoftWare ORiented Design & Synthesis framework

高瀬 英希

(京都大学)

takase@i.kyoto-u.ac.jp



京都大学
KYOTO UNIVERSITY

SWORDSフレームワーク

- ソフトウェア志向の組込みシステム協調設計環境
 - SWORDS: SoftWare ORiented Design & Synthesis
 - 現在はXilinx社Zynq-7000を対象に実装中
- SWの開発知識のみでシステム設計を進められる
 - HWや通信IFの開発知識は不要とする
 - テキストファイルの軽微な修正のみで様々な構成を合成できるようにする
 - システムレベルの高い抽象度で設計できる

高品質な組込みシステムを実現するための
設計生産性の向上に寄与する

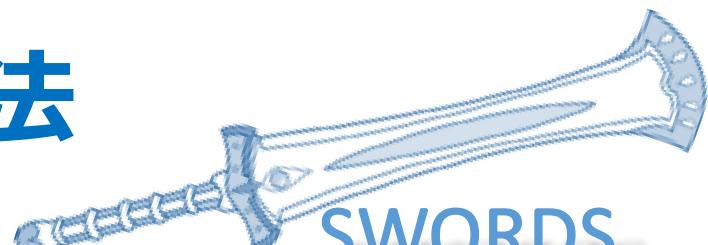
目次

- 導入方法
- 使用方法
 - JSON記述の詳細
 - (準備中)サンプルの例
- 今後の対応予定
- SWORDSフレームワークとは？
- (準備中)関連する論文リスト
- 本資料の変更履歴



github.com/tlk-emb/SWORDS

導入方法



SWORDS

SoftWare ORiented Design & Synthesis framework

導入方法

- 下記を参照

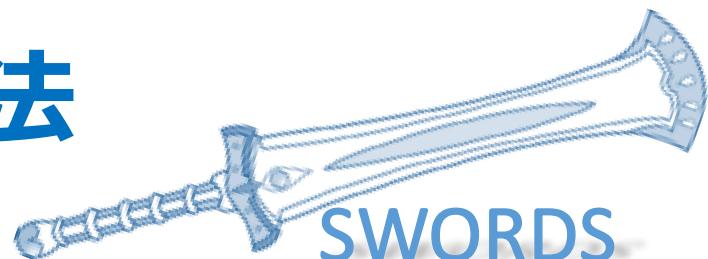
https://github.com/tlk-emb/SWORDS/blob/master/readme_setup.txt

– 本資料に転記予定

- 使用ツール（全て無償入手可能）

ツール	バージョン	用途
python	2.7.1	SW/HWタスク切分け・生成, IF合成, 各tclファイルの生成
clang	3.4以降	設計記述の解析
Vivado HLS	2015.4	HWタスクの高位合成
Vivado	2015.4	HWモジュールの論理合成
Xilinx SDK	2015.4	SWバイナリのコンパイル

使用方法



SoftWare ORiented Design & Synthesis framework

環境設定

- setenv.bat にて環境を設定する
- 例 :

```
SET xilinxpath=C:\Xilinx\2015.4
```

```
SET llvmfilepath="C:\Program Files (x86)\LLVM\bin\libclang.dll"
```

- xilinxpath: Xilinx社各種ツールのインストール位置
 - llvmfilepath: libclangの位置
- ✓ 実行時にコマンドラインで指定も可能（省略も可能）

使用方法

```
$ <gitrepodir>\swords.bat <design_file>.c &  
    <config_file>.json <proj_name> [<llvm-libfile>]
```

- Windows cmd.exe 上で実行
 - <gitrepodir>: GitHubからpullしたディレクトリ
 - ✓ パスを通しててもよい
 - <design_file>: 設計記述のCファイル
 - <config_file>: 構成記述のJSONファイル
 - <proj_name>: プロジェクト名
 - <llvm_libfile>: libclang.dllの位置 (絶対パス指定)
 - ✓ 省略可能 省略時のデフォルトは
" C:\Program Files (x86)\LLVM\bin\libclang.dll"

生成ファイルと実行方法

- HWビットストリーム
 - .¥<proj_name>¥<proj_name>.runs¥impl_1¥<proj_name>_system_wrapper.bit
- SWバイナリ
 - .¥<proj_name>¥<proj_name>.sdk¥<proj_name>¥Debug¥<proj_name>.elf
- ボード実行用スクリプト
 - <gitrepodir>¥batch¥executesystem.bat ¥<design_file>.c <config_file>.c <proj_name>
 - ✓ configsystem.bat はビルドと書き込みまで

設計記述

- 標準的なC言語でシステム設計を記述する
 - HW設計・SW/HW分割はあまり意識する必要はない
 - ✓ HW化対象関数には高位合成適用可能な記述のみ使用可
 - 記述不可の例：動的メモリ確保，ファイルポインタ，等
 - ✓ （高位合成可能なものなら）標準ライブラリ関数が記述可能
 - SW関数にはXilinx固有のライブラリ関数も記述可能
 - ✓ 例：Xtime_GetTime(), Xil_DCacheEnable(), 等
 - HW化対象関数の子関数もHW化される
 - ✓ SW関数と共に用する場合はSWとHWの両方が生成される
 - Vivado HLSの指示子も記述可能
 - ✓ 例：#pragma HLS PIPELINE II=25
#pragma HLS ARRAY_PARTITION variable=x dim=2

設計記述

- 現時点でのSWORDS固有の制約
 - 設計記述は1ファイルのみ指定可能
 - ✓ 複数ファイルの場合は対象関数／関係する記述（マクロ等）を1ファイルにまとめてから適用する
 - ✓ bitstream生成後のSWビルドはSDK上での手作業となる
 - 引数として記述できるのは配列のみ
 - ✓ 要素数を明示的に指定する必要がある（マクロは使用可）
 - ✓ スカラー・ポインタに対応予定

構成記述

- JSON形式の記述によってシステム構成を指定する
 - SWとして実行する関数
“software_tasks”
 - HW化する関数と通信I/F
“hardware_tasks”
 - 実行環境の指定
“environments”
- ✓ OS指定も対応予定

```
{  
  "software_tasks": [  
    {"name": "main"},  
    {"name": "func1"}],  
  "hardware_tasks": [  
    {"name": "func2", "mode": "s_axilite",  
     "arguments": [  
       {"name": "x", "mode": "m_axi",  
        "bundle": "bundle_a", "offset": "slave",  
        "direction": "in"},  
       {"name": "y", "mode": "m_axi",  
        "bundle": "bundle_a", "offset": "slave",  
        "direction": "in"},  
       {"name": "z", "mode": "s_axilite",  
        "bundle": "bundle_b",  
        "direction": "out"}],  
    "bundles": [  
      {"bundle": "bundle_a", "port": "ACP"},  
      {"bundle": "bundle_b", "port": "GPO"}  
    ]},  
  "environments": [  
    {"board": "zedboard"}  
  ]}
```

構成記述

- “software_tasks”
 - “name”: SW実行の関数名
 - ✓ “main”含めて省略可

```
{  
  "software_tasks": [  
    {"name": "main"},  
    {"name": "func1"}],  
  "hardware_tasks": [  
    {"name": "func2", "mode": "s_axilite",  
     "arguments": [  
       {"name": "x", "mode": "m_axi",  
        "bundle": "bundle_a", "offset": "slave",  
        "direction": "in"},  
       {"name": "y", "mode": "m_axi",  
        "bundle": "bundle_a", "offset": "slave",  
        "direction": "in"},  
       {"name": "z", "mode": "s_axilite",  
        "bundle": "bundle_b",  
        "direction": "out"}],  
    "bundles": [  
      {"bundle": "bundle_a", "port": "ACP"},  
      {"bundle": "bundle_b", "port": "GPO"}  
    ]},  
  "environments": [  
    {"board": "zedboard"}  
  ]}  
}
```

構成記述

- “hardware_tasks”
 - “name”: HW化する関数名
 - “mode”: 実行完了の検知に
 使用する通信プロトコル
 - ✓ 現在は“s_axilite”のみ指定可
 - “arguments”: 引数の通信に
 使用するI/F
 - ✓ 次ページ参照
 - “bundles”: HWモジュールの
 入出力ポートの情報
 - ✓ 次ページ参照

```
{  
    "software_tasks": [  
        {"name": "main"},  
        {"name": "func1"}],  
    "hardware_tasks": [  
        {"name": "func2", "mode": "s_axilite"},  
        "arguments": [  
            {"name": "x", "mode": "m_axi",  
            "bundle": "bundle_a", "offset": "slave",  
            "direction": "in"},  
            {"name": "y", "mode": "m_axi",  
            "bundle": "bundle_a", "offset": "slave",  
            "direction": "in"},  
            {"name": "z", "mode": "s_axilite",  
            "bundle": "bundle_b",  
            "direction": "out"}],  
        "bundles": [  
            {"bundle": "bundle_a", "port": "ACP"},  
            {"bundle": "bundle_b", "port": "GPO"}]  
    ],  
    "environments": [  
        {"board": "zedboard"}  
    ]  
}
```

構成記述

- “arguments”: 通信I/F
 - “name”: 引数名
 - “mode”: 通信プロトコル
 - ✓ 現状は“m_axi”, “s_axilite”のみ動作サポート
 - “offset”: (m_axi使用時のみ)
アドレスオフセット
 - ✓ “off”: 不使用 (デフォルト)
 - ✓ “direct”: 32ビットポートを追加
 - ✓ “slave”: AXI4-Lite内に
32ビットポートを追加
 - “direction”: 入出力方向
 - ✓ “in”, “out”のみ

```
{  
  "software_tasks": [  
    {"name": "main"},  
    {"name": "func1"}],  
  "hardware_tasks": [  
    {"name": "func2", "mode": "s_axilite"},  
    "arguments": [  
      {"name": "x", "mode": "m_axi",  
       "bundle": "bundle_a", "offset": "slave",  
       "direction": "in"},  
      {"name": "y", "mode": "m_axi",  
       "bundle": "bundle_a", "offset": "slave",  
       "direction": "in"},  
      {"name": "z", "mode": "s_axilite",  
       "bundle": "bundle_b",  
       "direction": "out"}],  
    "bundles": [  
      {"bundle": "bundle_a", "port": "ACP"},  
      {"bundle": "bundle_b", "port": "GPO"}]  
    ],  
    "environments": [  
      {"board": "zedboard"}  
    ]  
}
```

構成記述

- “bundles”: 通信バンドル
 - “bundle”: バンドル名
 - “port”: 通信ポートの指定
 - ✓ “GP0”, “GP1”, “ACP”
 - ✓ GPはMaster/Slave区別なし
 - プロトコルに応じて決まる
 - ✓ ACPはm_axiのみに対応
 - ✓ HP0-3は動作未検証

```
{  
  "software_tasks": [  
    {"name": "main"},  
    {"name": "func1"}],  
  "hardware_tasks": [  
    {"name": "func2", "mode": "s_axilite"},  
    "arguments": [  
      {"name": "x", "mode": "m_axi",  
       "bundle": "bundle_a", "offset": "slave",  
       "direction": "in"},  
      {"name": "y", "mode": "m_axi",  
       "bundle": "bundle_a", "offset": "slave",  
       "direction": "in"},  
      {"name": "z", "mode": "s_axilite",  
       "bundle": "bundle_b",  
       "direction": "out"}],  
    "bundles": [  
      {"bundle": "bundle_a", "port": "ACP"},  
      {"bundle": "bundle_b", "port": "GP0"}  
    ]  
  ],  
  "environments": [  
    {"board": "zedboard"}  
  ]  
}
```



構成記述

- プロトコル
- ポート
- バンドル

ポート	マスタ	ポート数	用途
GP-AXI	PS/PL	2/2	汎用
AXI-HP	PL	4	メモリアクセス
AXI-ACP	PL	1	キャッシュアクセス

プロトコル	バースト転送	特徴
AXI4-Lite	不可	省リソース
AXI4	256回まで	大量データ向き
AXI4-Stream	無制限	ストリーミング特化

構成記述

- “environments”
 - “board”: 使用するボード
 - ✓ zedboard, zc702
 - ✓ 省略可 (省略時はzedboard)

```
{  
  "software_tasks": [  
    {"name": "main"},  
    {"name": "func1"}],  
  "hardware_tasks": [  
    {"name": "func2", "mode": "s_axilite"},  
    "arguments": [  
      {"name": "x", "mode": "m_axi",  
       "bundle": "bundle_a", "offset": "slave",  
       "direction": "in"},  
      {"name": "y", "mode": "m_axi",  
       "bundle": "bundle_a", "offset": "slave",  
       "direction": "in"},  
      {"name": "z", "mode": "s_axilite",  
       "bundle": "bundle_b",  
       "direction": "out"}],  
    "bundles": [  
      {"bundle": "bundle_a", "port": "ACP"},  
      {"bundle": "bundle_b", "port": "GPO"}  
    ]  
},  
  "environments": [  
    {"board": "zedboard"}  
]  
}
```

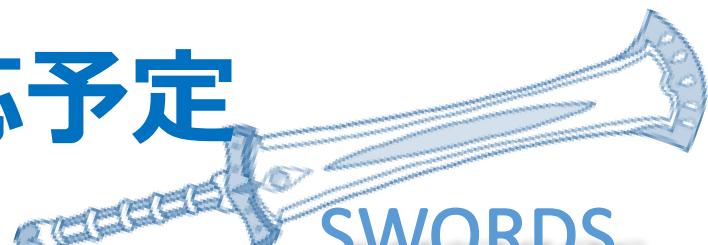
構成記述

- 現時点でのSWORDS固有の制約
 - HW関数は1つのみ指定可能
 - axis (AXI4-Stream)およびHPポートの指定不可
 - ボードはzedboard, zc702 (ZC702評価キット)のみ
 - standaloneアプリのみ対応

サンプルの例

- T.B.A.

今後の対応予定



SWORDS

SoftWare ORiented Design & Synthesis framework

直近の対応予定／着手中

- スカラー・ポインタの引数への対応
- 複数bundleで同ポートを共用する際の対応
- 複数HWタスクへの対応
- 他ボード対応 (Zybo, MicroZed, MPSoC+等)

中長期的な開発計画

- AXI-Streamプロトコル／HPポートへの対応
- ACPポート使用時の高速化サポート
 - 現状では前後でキャッシュの無効／有効化が必要
- 複数ファイルによる設計記述への対応
- Linuxアプリ設計への対応
- リアルタイムシステム設計（TOPPERS）への対応

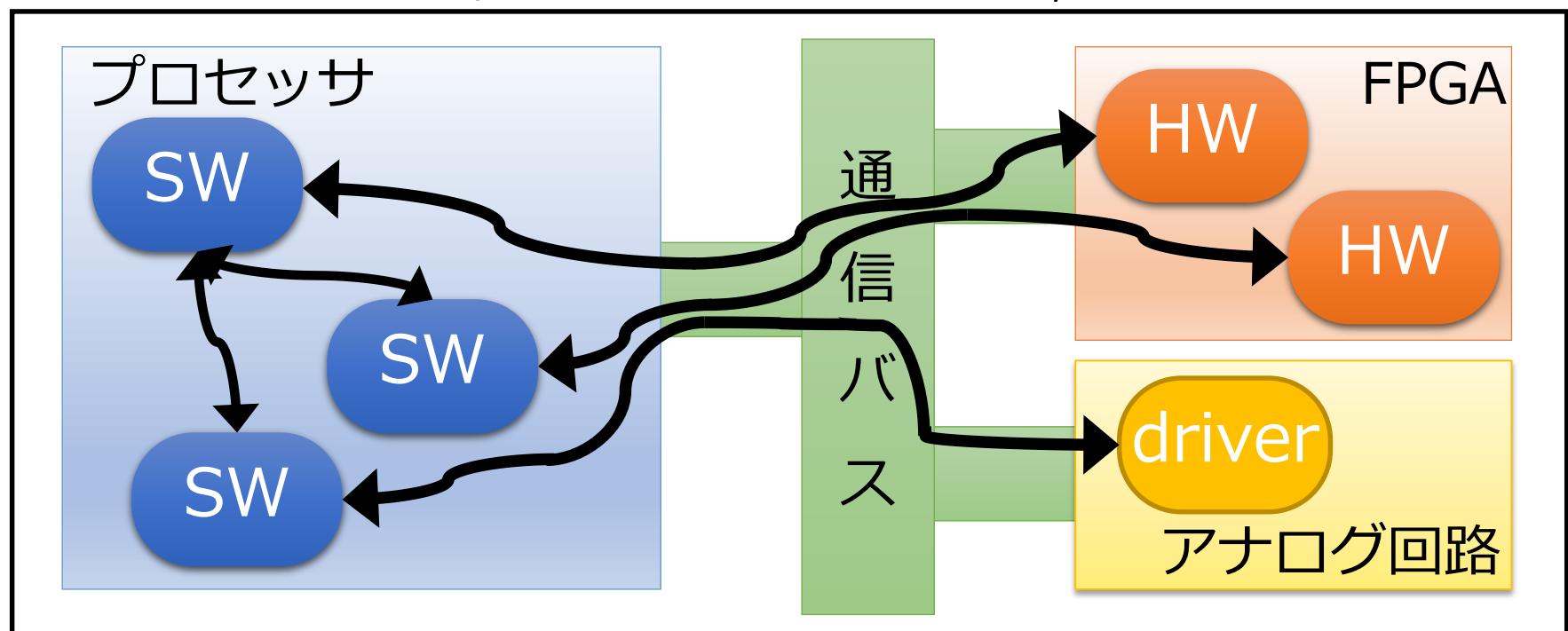
SWORDSフレームワーク とは？



SoftWare ORiented Design & Synthesis framework

混載型のデバイス

- プロセッサ + FPGA + 専用回路 + 専用バス
 - 柔軟性の必要な逐次処理はプロセッサで、高速応答な並列処理はFPGAで、外部環境とのインタラクションは専用アナログ回路で
 - 高品質な組込み/IoTシステムを実現できる、かもしれない？



協調設計の課題

ソフトはいつから開発できるかな,,,

どれをどこで処理するか？？

ハードも設計する,,,

```
always@(posedge CLK) begin
  if (!RES) XOUT <= 0;
  else begin
    case (IN)
      b0001 : TMP <= 1;
      b0010 : TMP <= 5;
      b01100 : TMP <= 7;
      default : TMP <= 0;
    endcase
  end
end
assign OUT = ~{REG[7:4]},
```

プロセッサ

```
int_t main () {
  a = func1(xxx);
  b = func2(yyy);
  c = a + b;
}
int_t func1(x) {
  ...
  return val;
}
int_t func2(y) {
  ...
  z = func3(y);
  ...
  return val;
}
```

V

SW

W

通信
バス

HW

HW

driver

アナログ回路

ドライバはどう用意する？

データ通信はどう実現する？

SWORDSフレームワークとは？

- ソフトウェア志向の組込みシステム協調設計環境
 - SWORDS: SoftWare ORiented Design & Synthesis
 - 現在はXilinx社Zynq-7000を対象に実装中
- SWの開発知識のみでシステム設計を進められる
 - HWや通信IFの開発知識は不要とする
 - テキストファイルの軽微な修正のみで様々な構成を合成できるようにする
 - システムレベルの高い抽象度で設計できる

高品質な組込みシステムを実現するための
設計生産性の向上に寄与する

対象とするデバイス

- Xilinx社 Zynq-7000 All Programmable SoC

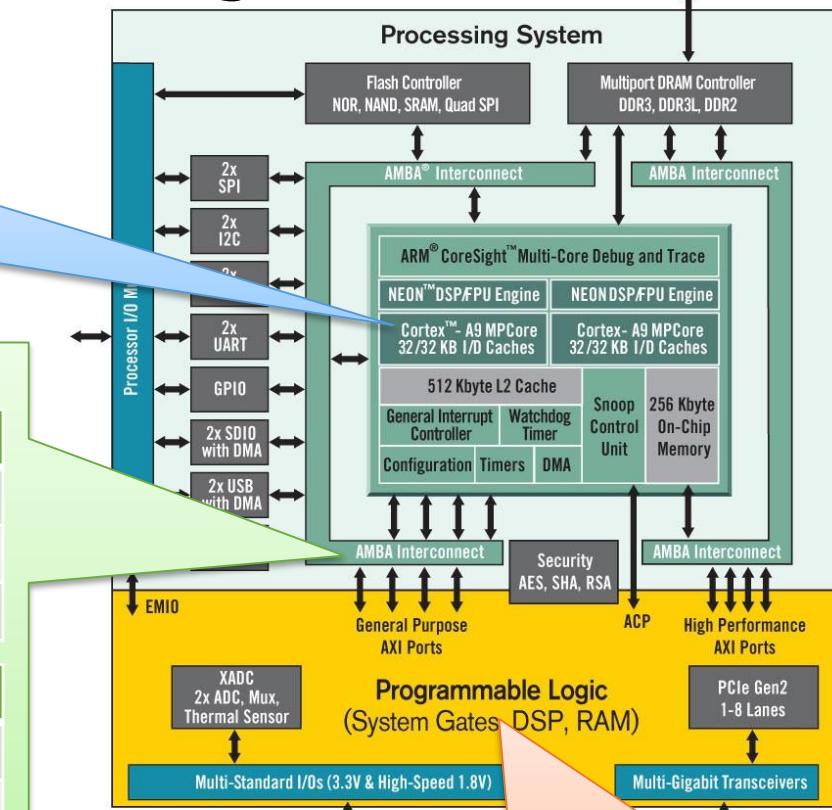
PS (プロセッシングシステム)

- ARM Cortex A9デュアルコア
- キヤッシュ
- メインメモリ
- ペリフェラル
- ...

ARM AMBA AXI : PSとPLを接続

ポート	マスタ	ポート数	用途
GP-AXI	PS/PL	2/2	汎用
AXI-HP	PL	4	メモリアクセス
AXI-ACP	PL	1	キヤッシュアクセス

プロトコル	バースト転送	特徴
AXI4-Lite	不可	省リソース
AXI4	256回まで	大量データ向き
AXI4-Stream	無制限	ストリーミング特化



PL (プログラマブルロジック)

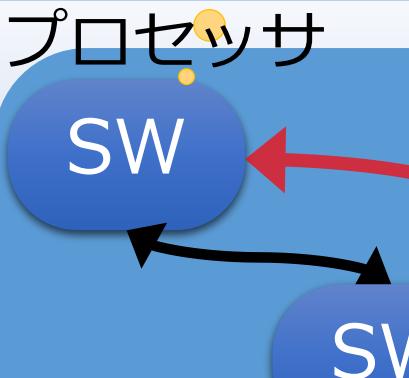
- 構成可能論理ブロック
- デジタル信号処理ブロック
- ブロックメモリ
- ...

SWORDSフレームワーク



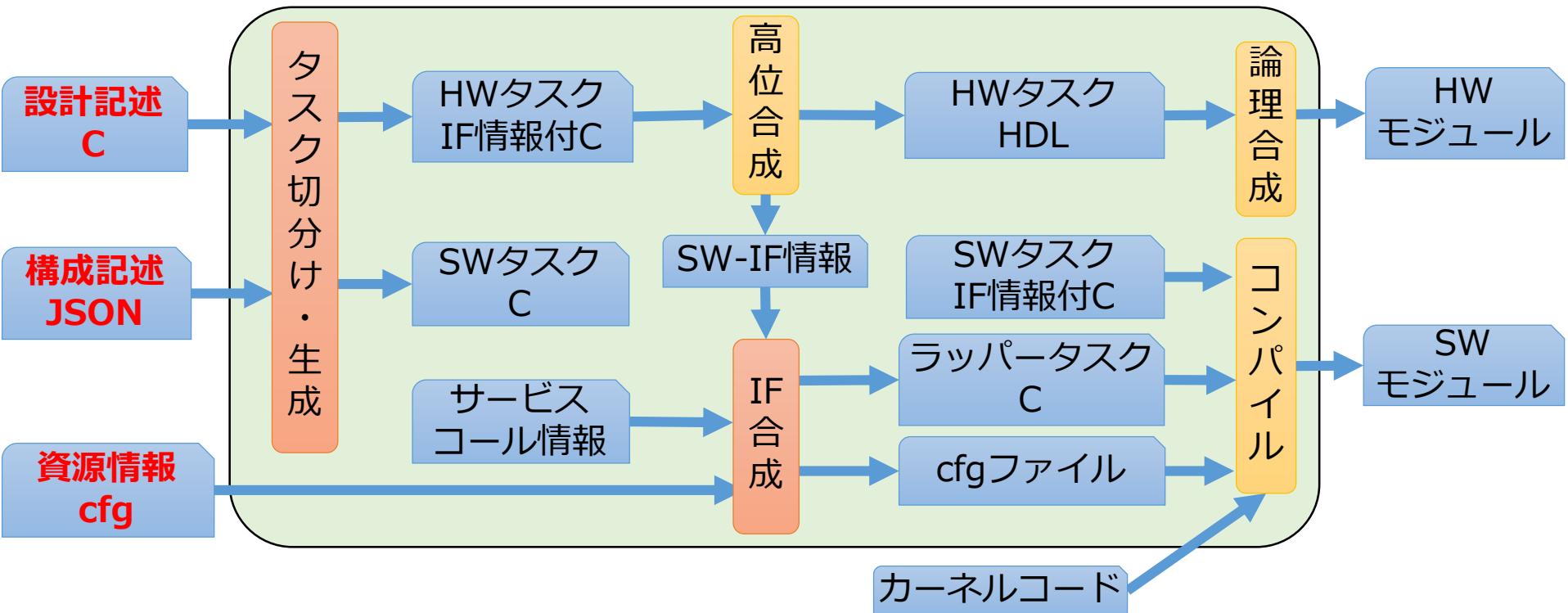
SWの知識だけで
設計する

SWからHWを
自動合成する

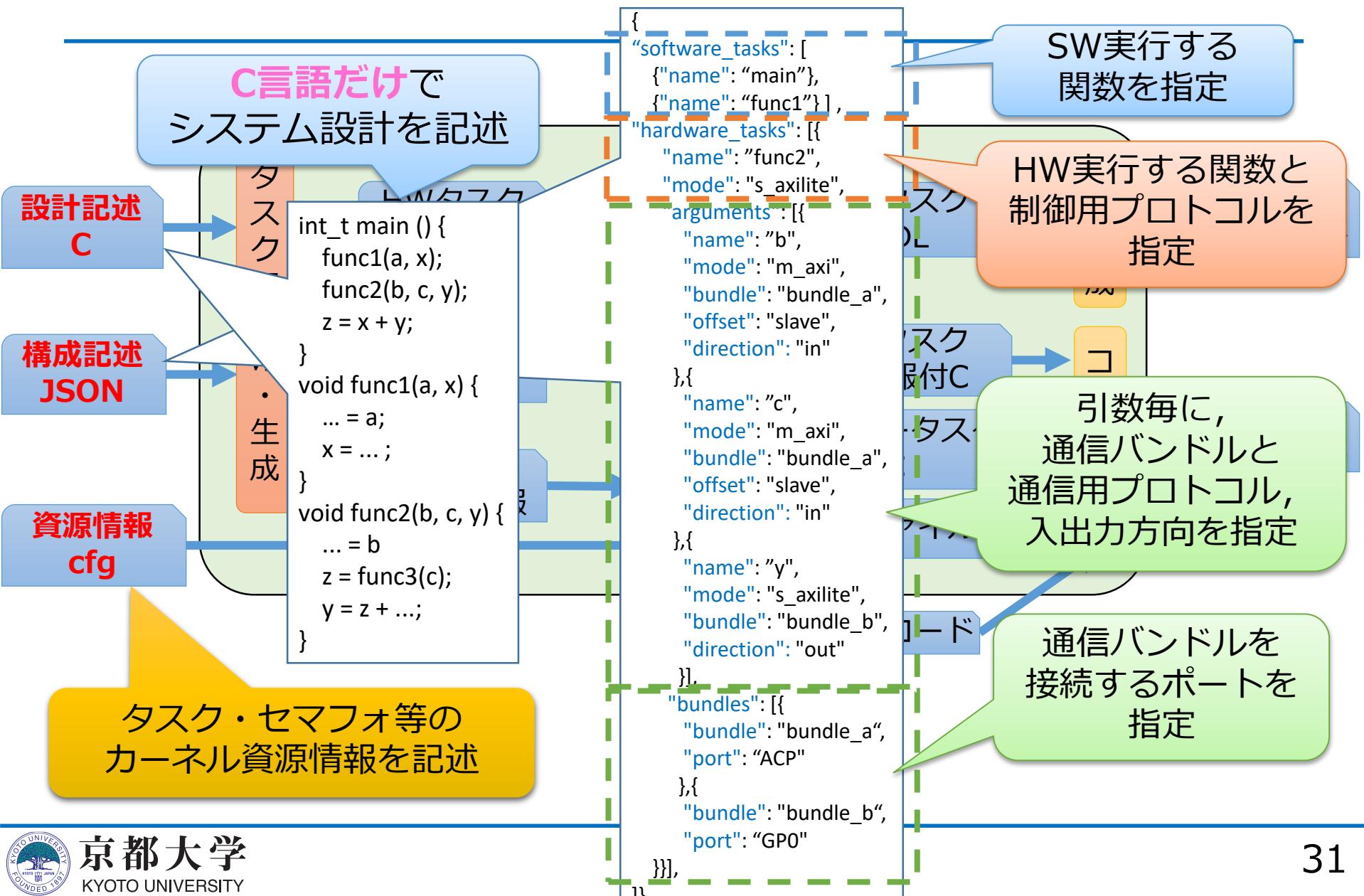


HWをSWと同じように
管理できるOSも提供する

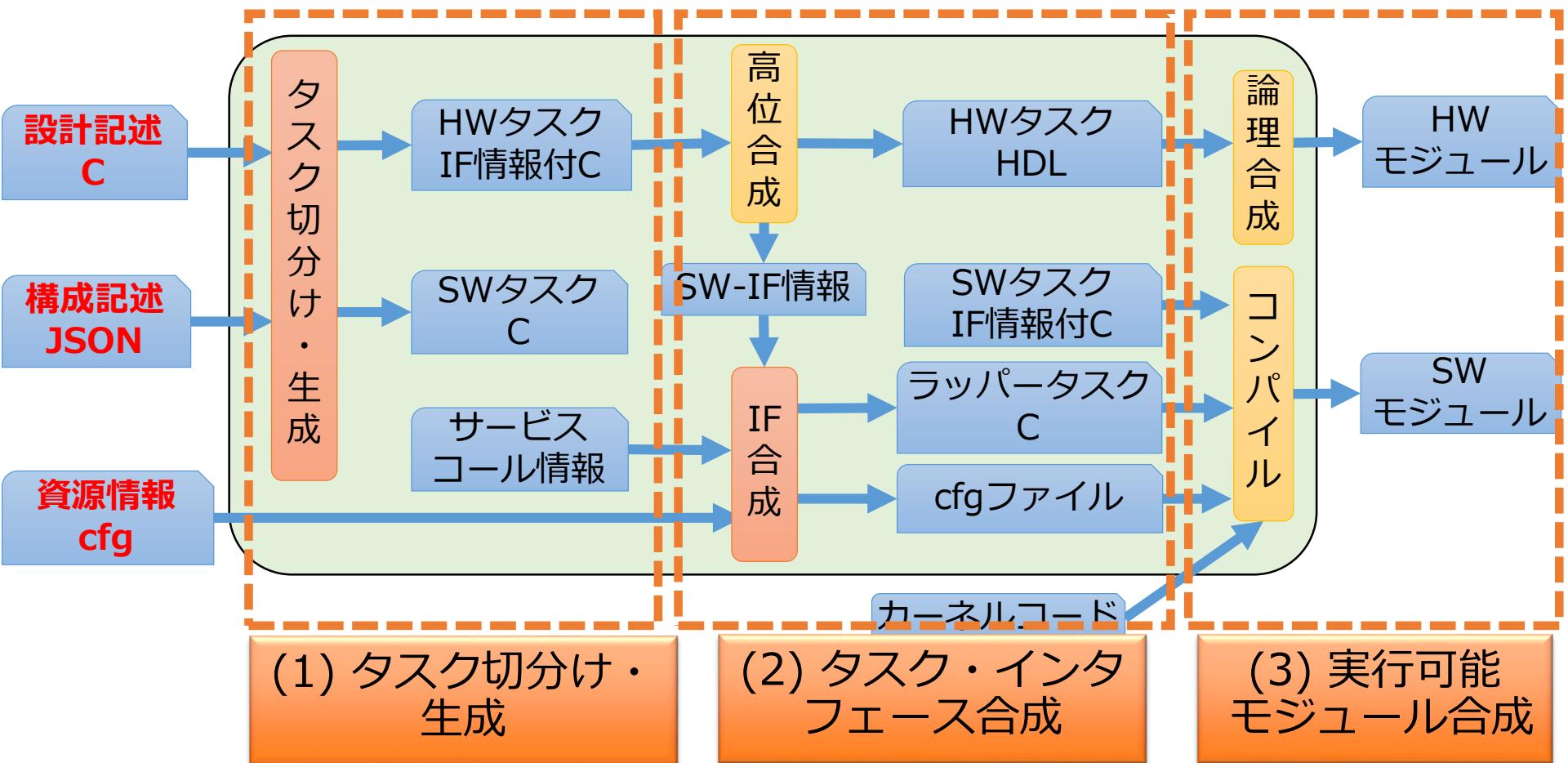
SWORDTSの全体像



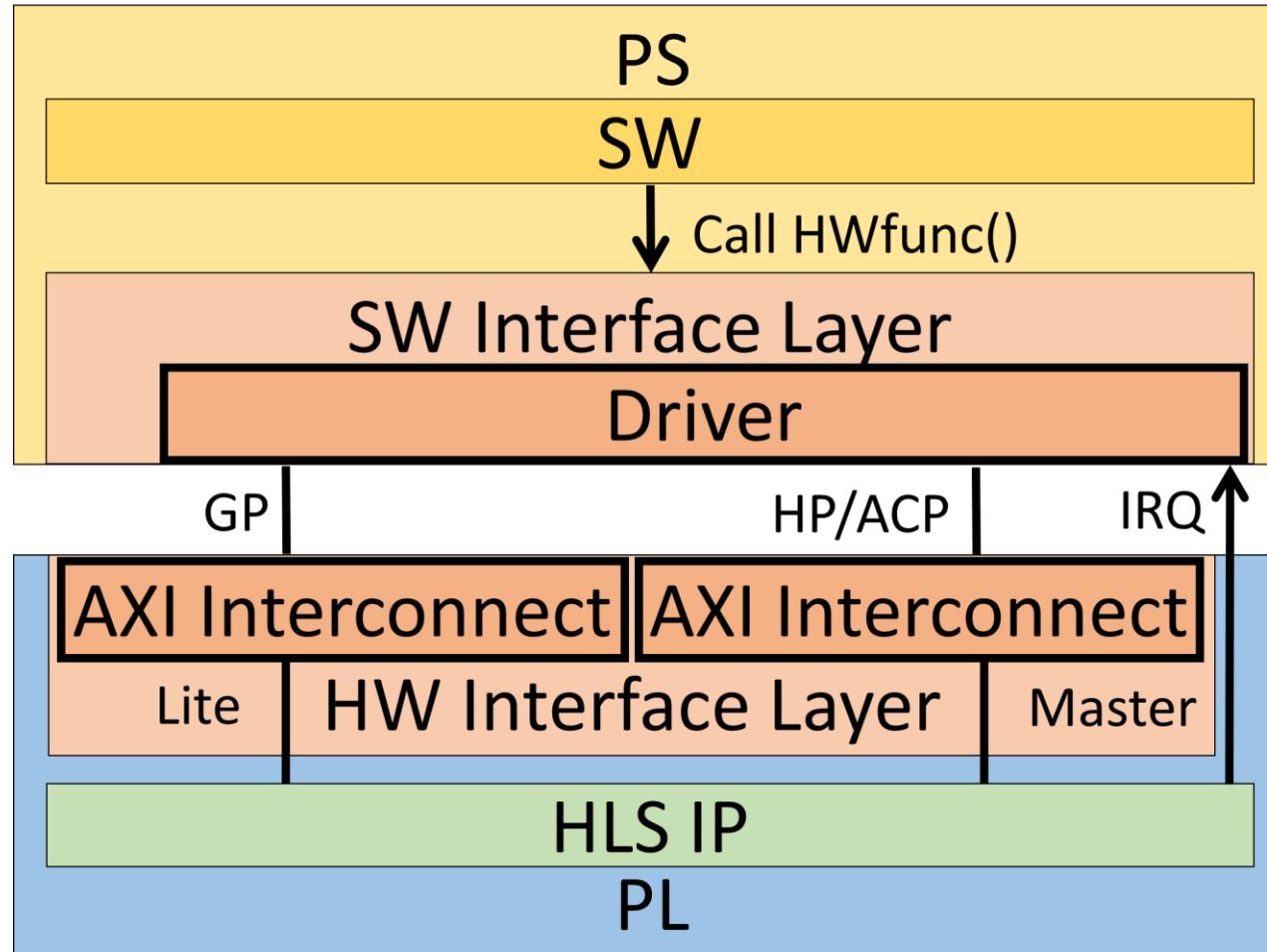
SWORDSの入力



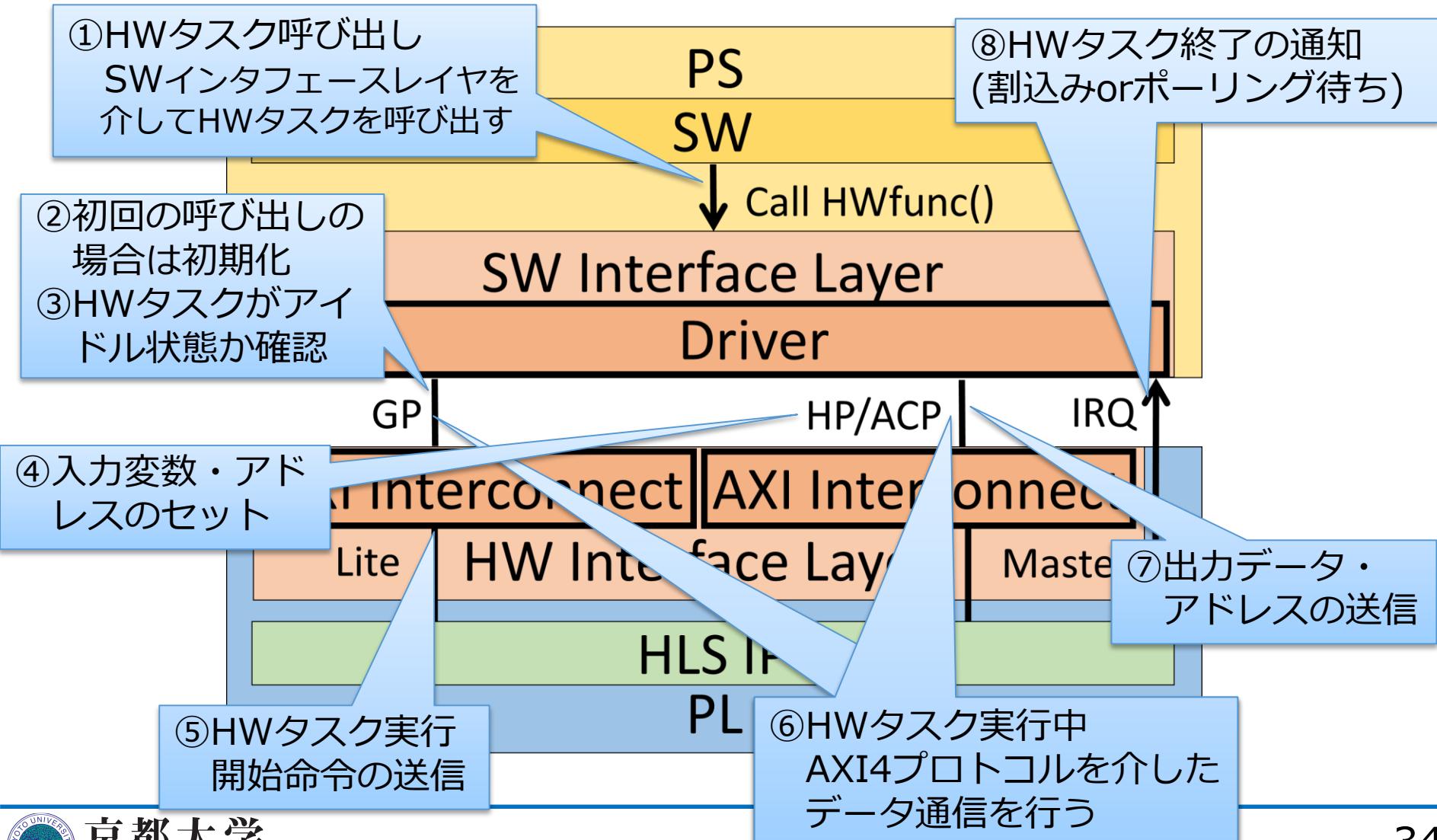
SWORDSのフロー



合成される協調システムの構造



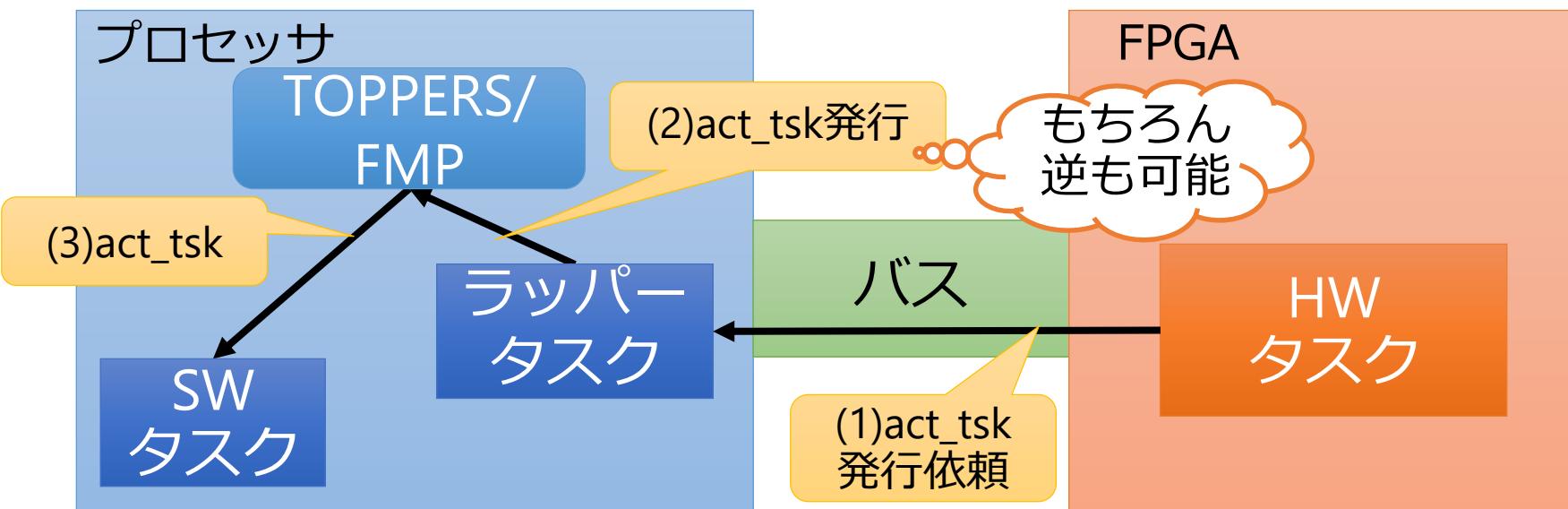
合成される協調システムの振る舞い



リアルタイムシステム設計への対応

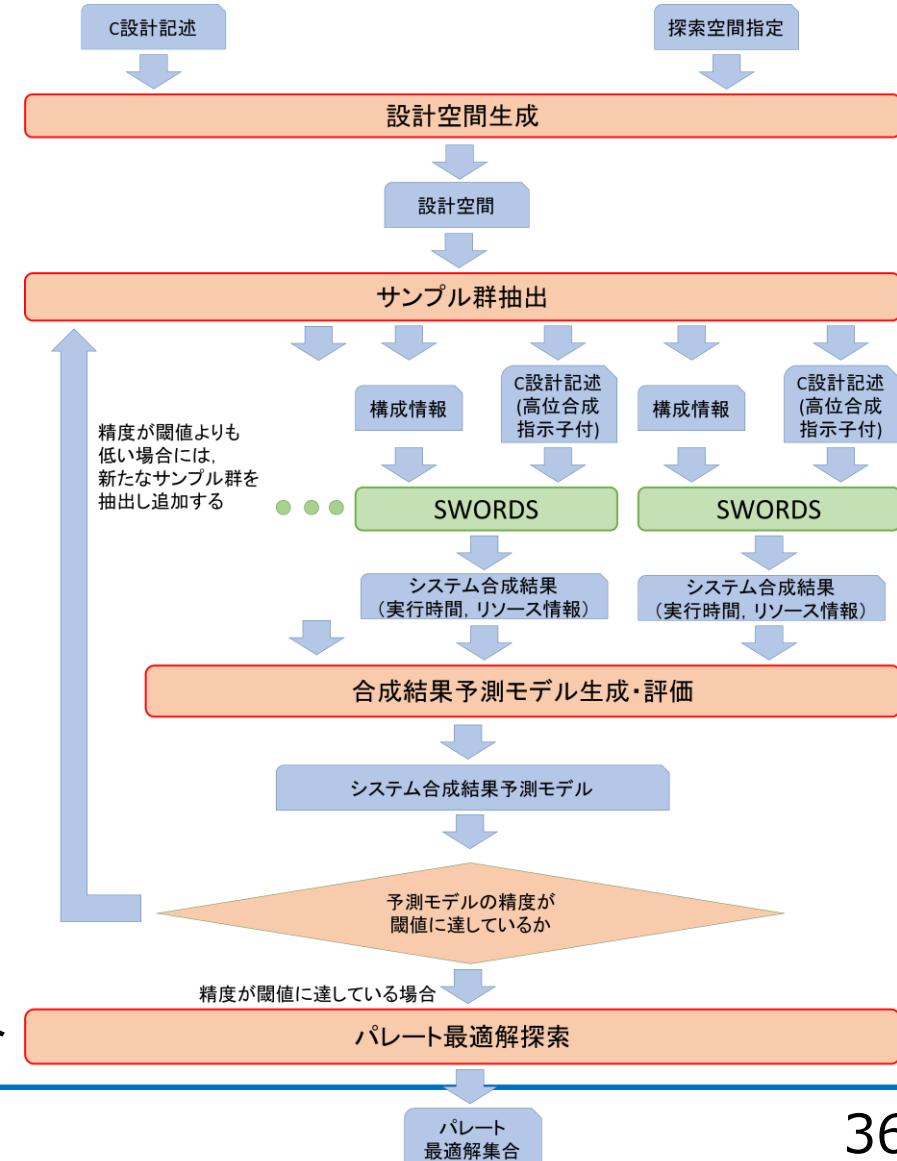
現在実装中

- TOPPERS/FMPカーネルを含めたシステム設計に対応
 - ITRONアプリの設計作法でZynq上にリアルタイムシステムを合成
- HWタスクを管理するラッパータスクを生成
 - サービスコールの処理や発行をラッパータスクが担う
 - FMPカーネルがSWタスクと同様にHWタスクも管理できる



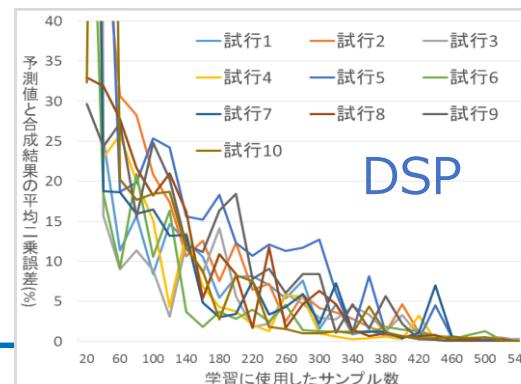
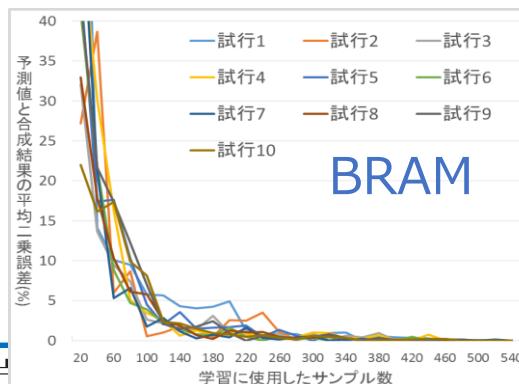
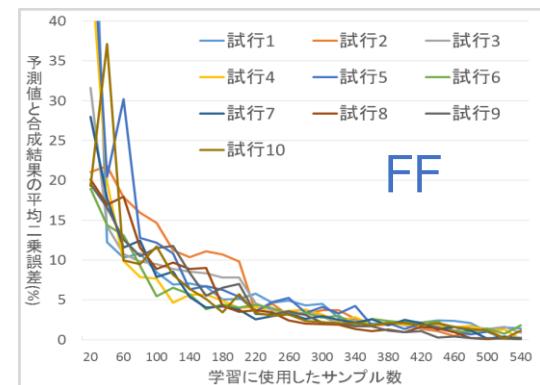
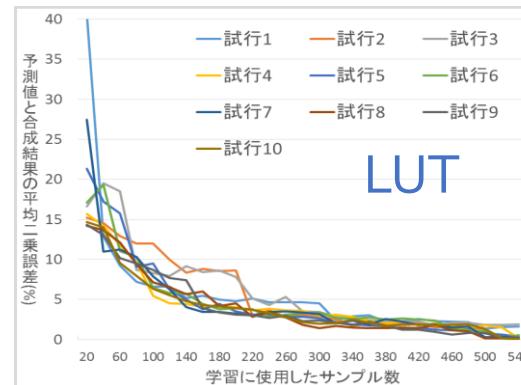
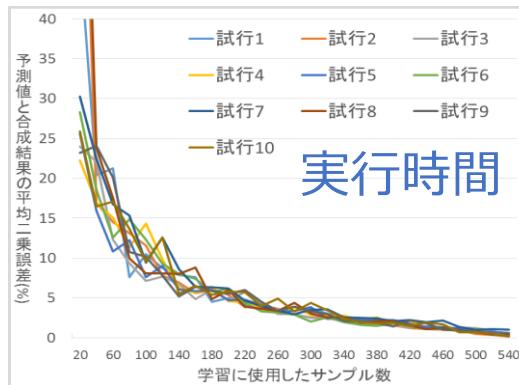
設計空間探索への応用

- ・高性能な協調システムの構成を高速に探索する
 - 選択サンプルをSWORDSでシステム合成し、その結果から予測モデルを生成する
 - 予測モデルの精度を解析し、未合成の構成の性能を見積もる
- ・入力：設計記述および探索空間の指定
 - システム構成：
SW/HWの切分け、高位合成最適化、通信IFの選択
- ・出力：パレート最適となるシステム構成の解集合



設計空間探索への応用

- Xilinx Zynq XC7Z020-1CLG484 (ZedBoard搭載)
- 適用対象：行列積演算
 - 構成候補の設計空間は計567通り（引数毎の通信方式とループ展開方法）
 - サンプル群抽出における抽出個数：20個ずつ



全567個のうちの
平均140個の合成で
見積誤差は10%程度
(DSPを除く)

セールスポイント

1. ソフトウェアの開発知識のみで協調設計が可能
 - 組込みシステム開発の設計生産性の向上が期待される
2. 様々なシステム構成の効率的な試行が可能
 - 構成情報の記述修正だけでHWタスクや通信IFが容易に選択できる
 - 高効率な設計空間探索手法の確立に繋がる
3. 多デバイスへの適用を想定した設計フロー
 - 既存のツールのフロントエンドであり、移植性がある
4. リアルタイムOSを採用したシステム開発に対応
 - 並行性設計や時間管理の実現が容易になる

関連する論文リスト

- T.B.A.

変更履歴

- 2017/09/07 : 初版作成