# リアルタイムシステムのフルハードウェア化のための Python によるハードウェア設計

志賀光 Hikaru Shiga
石浦 菜岐佐 Nagisa Ishiura
神原 弘之 Hiroyuki Kanbara
冨山 宏之 Hiroyuki Tomiyama
関西学院大学
京都高度技術研究所
立命館大学

本発表ではリアルタイムシステムのフルハードウェア化におけるハードウェア設計を Python で行う手法を提案する．組込みシステムの高機能化に伴い処理が複雑化し, リアルタイム性の実現が困難になっている．我々の研究室では RTOS を利用して構築したシステムを自動的に全てハードウェア化する手法を開発している．従来はタスクや RTOS サービス等の数が変わるたびに手動で設計を変更する必要があった．本研究では Python と Verilog HDL を生成するライブラリである Veriloggen を用いてハードウェアを設計を行い, 任意のタスク数, インスタンス数等に対応したモジュールを生成する手法を提案する．

## 1 はじめに

近年の情報通信技術の発展により, 様々なデバイスやサービスが開発されつつあるが, それに伴い組込みシステムにはより高い機能が要求されるようになっている．また, 車載機器やロボットの制御には機能性に加え高い応答性能が要求される．これらのシステムの開発と設計には, 入力イベントに対する処理を決められた時間内に完了するという時間制約を守ること (リアルタイム性) が要求される．このようなリアルタイムシステムの開発と設計はリアルタイム OS (RTOS) を用いて行われる．RTOS はリアルタイム性を実現するために様々な機能を提供する．しかし, システムの高機能化・複雑化による処理量の増加によってリアルタイム性の実現が困難になりつつある．

RTOS を使用したシステムの応答性能向上の手法として, RTOS 機能の一部もしくは大部分をハードウェア化する手法が提案されている．文献 [1], [2], [3] は RTOS のスケジューリング機能を, 文献 [4], [5] は RTOS のほとんどの機能をハードウェア化することにより, 応答性能の向上を図っている．しかし, これらの手法ではタスクおよびハンドラはソフトウェアであるため, CPU 待ちやコンテキストスイッチによるオーバヘッドが発生する．

この課題を解決する手法として, 文献 [6] は RTOS の機能に加えタスク/ハンドラの全てをハードウェア実装する手法を提案している．文献 [7] は文献 [6] のアーキテクチャにおいて, RTOS のサービス処理機能を集約することにより回路規模を削減する手法を提案している．また, 文献 [8] は文献 [7] のアーキテクチャにおいて, タスクを汎用的な高位合成ツールで合成できる制御手法を, 文献 [9] では RTOS 機能を提供する管理ハードウェアを自動生成する手法を提案している．

しかし, 文献 [9] では, 一旦ハードウェアを Verilog HDL で設計して動作確認しその後に, パラメータに応じたハードウェアを生成するスクリプトを作成していた．また, Verilog HDL をテキストベースで生成しているため, 設計の変更, サービスモジュールの追加等が発生した場合, ハードウェア設計とは別に新たにテキストベースの生成スクリプトを作成する必要がある．

本稿では Python と Verilog HDL を生成するライブラリである Veriloggen [10] を用いることにより, 必要なハードウェアの設計を Python レベルで行う手法を提案する．

## 2 RTOS 利用システムのフルハードウェア実装

### 2.1 RTOS を用いたシステムのフルハードウェア実装

文献 [6] は RTOS の機能とタスクおよびハンドラ全てをハードウェア化する手法を提案している．その概念を図 1 に示す．上図はソフトウェアで実装したシステムであり, タスク (TSK$i$) は RTOS の管理下で CPU により実行される．下図の各 (TSK$i$) は独立したモジュールとしてハードウェア化される．また RTOS の機能をハードウェア化したものがマネージャ (manager) である．このシステムではタスクの実行可能状態になれば全て並列に実行することができ, 実行制御は manager が各タスクの状態から実行と停止の信号を出力することにより行う,

ハードウェア化によりタスクは独立に並列に実行されるため, CPU 待ちとスケジューリングおよびコンテキストスイッチのオーバヘッドをなくせる．その上, タスクをハードウェア化して高速実行できるため, システムの応答性能を従来のハードウェア実装に比べ格段に向上させることができる．

### 2.2 アーキテクチャ

本稿で前提とする文献 [11] のアーキテクチャを図 2 に示す．T0, T1, T2 はタスクをハードウェア化したもの, manager は RTOS 機能をハードウェア化したモジュールである．

manager の下部にある mutex, event flag, dataqueue, shared variable, control は, RTOS のサービス機能を提供するサービスモジュールである．mutex は排他制御,

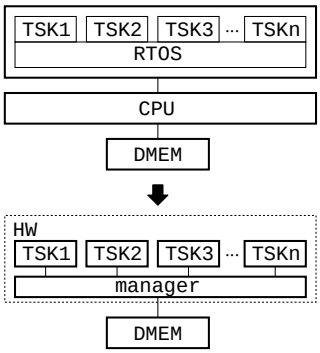


図 1　RTOS 利用システムのフルハードウェア実装 [6]

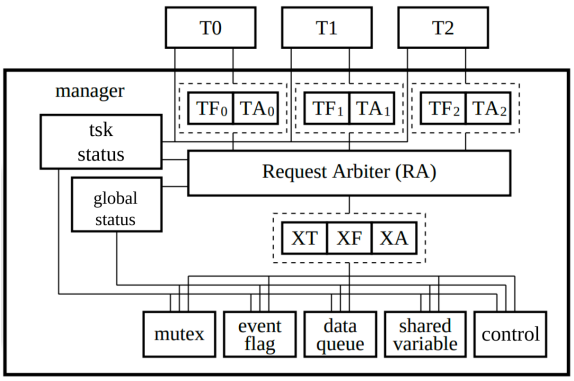


図 2　文献 [11] のハードウェアの構成

event flag はタスク間の同期, dataqueue はタスク間の通信, shared variable は共有変数の読み書きを提供するモジュールである．また, control はタスクの起動/休止や優先度の変更等のサービスを提供する．

タスクやシステムの状態は manager 内の状態レジスタである tsk status および global status に保持される．tsk status は各タスクのステータス情報を記憶しており, タスクの状態, 現在の優先度, ベース優先度, タイマー等を管理する．global status はカーネルの情報を記憶しており, 割込みロックフラグ, CPU ロックフラグ, 割込優先度マスク等を管理する．サービスモジュールは状態レジスタを参照・更新することにより処理を実行する．

タスクは並列に実行されるが, サービス間の干渉を避けるためサービスは同時に 1 つのみ実行する仕様となっている．そのため Request Arbiter (RA) は複数のタスクからサービス要求があった場合に, タスクの優先度のに基づいて要求が処理される．manager 中央にある Request Arbiter (RA) は優先度に基づいて調停を行う回路である．

タスク (T$i$) 中のサービスコールは制御レジスタ TF$i$, TA$i$ への書き込みに変換される．タスクはレジスタ TF$i$ にサービスの ID を, TA$i$ に必要な引数を書き込んでサービス処理の完了を待つ．RA はサービス要求のあったタスクの中で優先度最大のタスクの番号を求め, レジスタ XT, XF, XA にそれぞれ優先度最大のタスク番号, TF$i$ のサービスの ID, TA$i$ の引数を書き込む．サービスモジュールは要求された処理を行い, 返り値を XA レジスタに書き込む．manager が XA の値を TA$i$ にコピーしてタスクにサービスの完了を通知すると, タスクは TA$i$ から返り値を読み取って自身の処理を再開する．

## 3　Python によるハードウェア RTOS 設計

### 3.1　Veriloggen

Veriloggen [10] は,Python でハードウェアを構築するための混合パラダイムフレームワークであり, Verilog HDL の抽象構文木 (AST) の低レベルの抽象化を提供する．この AST 抽象化と Python の全機能を使用して,Verilog HDL で記述されたハードウェア設計を非常に簡潔に構築することができる．Veriloggen は, プログラマーが直接ハードウェアを設計するためのものではなく, 効率的なドメイン固有言語やツールを開発するための効果的な抽象化を提供するために設計されてる．これにより, 設計の変更や新たなサービスモジュールの追加などが発生した場合にも, 効率的に対応することが可能である．

### 3.2　Python プログラムによるハードウェア設計

本稿では Veriloggen と Python を用いて文献 [6] [7] [8] [9] [11] のアーキテクチャにおける manager モジュールの Verilog HDL を設計する手法を提案する．タスク数, サービスモジュールを使用するタスク, エラーコード, サービスの仕様等を JSON で記述すると, それらの情報を基に RTOS の管理機能, サービスモジュール等を生成する Python スクリプトにより設計を行う．この際必要な RTOS のサービス機能のみを生成することにより回路規模の削減を図る．

Verilog HDL では, 状態に基づく制御を記述する際に, ポートやワイヤーへの代入記述とレジスタへの代入記述を異なる場所に書かなければならないため, 可読性が悪い．これに対し, 本稿の手法では, 両者を同一の構文で同一の場所に記述し, Python による解析によって両者を分離した Verilog HDL を生成する．Python と Veriloggen を用いて入力値によって出力値と register の値の変化を記述する例を図 3 に示す．図 3 は入力値 `input_a`, `input_b` の値によって出力値 `output_a`, `output_b`, register の値 `reg_a`, `reg_b` を更新する記述である．図 3 の 9~14 行目のように出力ポートへの代入とレジスタへの代入が混在した記述から, 図 4 のようにこれらを分離した Verilog HDL 記述が生成される．

## 4　実装と実験

現在, 本手法により TOPPERS/ASP3 の統合仕様書 [12] をベースに mutex, dataqueue を Python と Veriloggen を用いて実装した．実装した mutex と dataqueue を Xilinx FPGA Artix-7 (xc7a100tcsg324-3) をターゲットに論理合成を行った結果を表 1, 2 に示す．mutex, dataqueue はタスク数 4, インスタンス数 1, 2, 3 であり, dataqueue は 1 つのインスタンスが $32bit \times 4$ となっている．また表 1, 2 の Python はハードウェア生成スクリプトの行数を Verilog は出力された Verilog HDL の行数を

```
import veriloggen as vg
input_a = vg.Input("input_a", 4)
input_b = vg.Input("input_b", 4)
output_a = vg.Output("output_a", 4)
output_b = vg.Output("output_b", 4)
reg_a = vg.Reg("reg_a", 4)
reg_b = vg.Reg("reg_b", 4)
vg.If(input_a == 0)(
  output_a(3),
  reg_a(1)
  vg.If(input_b == 1)(
    output_b(2),
    reg_b(0)
  )
)
```

図 3　Python による入力, 出力, register 値更新の記述例

```
always @(posedge CLK) begin
    if(input_a == 0) begin
        reg_a <= 1;
        if(input_b == 1) begin
            reg_b <= 0;
        end
    end
end

function [4-1:0] output_a_assign_func
    ;
    input [4-1:0] input_a;
    input [4-1:0] input_b;
    begin
        if(input_a == 0) begin
            output_a_assign_func = 3;
            if(input_b == 1) begin
            end
        end
    end
endfunction

function [4-1:0] output_b_assign_func
    ;
    input [4-1:0] input_a;
    input [4-1:0] input_b;
    begin
        if(input_a == 0) begin
            if(input_b == 1) begin
                output_b_assign_func
                    = 2;
            end
        end
    end
endfunction
```

図 4　図 3 から生成された Verilog HDL

表している. コンパクトな Python 記述からパラメータに応じた Verilog HDL 記述が生成できている.

表 1　mutex における Python および Verilog の行数とハードウェア生成の結果

| #instance | Python | Verilog | #LUT | #FF | delay [ns] |
|---|---|---|---|---|---|
| 1 | 247 | 942 | 47 | 8 | 5.802 |
| 2 | | 1387 | 83 | 12 | 6.029 |
| 3 | | 1832 | 123 | 16 | 5.957 |

表 2　dataqueue における Python および Verilog の行数とハードウェア生成の結果

| #instance | Python | Verilog | #LUT | #FF | delay [ns] |
|---|---|---|---|---|---|
| 1 | 214 | 787 | 79 | 10 | 5.595 |
| 2 | | 1214 | 57 | 10 | 6.011 |
| 3 | | 1649 | 121 | 31 | 5.812 |

## 5　むすび

本稿ではリアルタイムシステムのフルハードウェア化におけるハードウェア設計を Python で行う手法を提案した. 本手法では. タスク, インスタンス等の数が変わるたびに手動で設計を変更することなく, ハードウェア設計の段階で任意のタスク数, インスタンス数等に対応したモジュールを生成することができる. テストベンチの生成や割り込みハンドラの実装が今後の課題である.


### 謝辞

本研究に関して有益な御助言を頂いた元立命館大学の中谷嵩之氏, 元関西学院大学の田村真平氏に感謝致します. 本研究は一部科研費 21K19776, 24K14885 の助成による.


## 参考文献


[1] Y. Cho, S. Yoo, K. Choi, N-E Zergainoh, and A. A. Jerraya: “Scheduler implementation in MPSoC design,” in *Proc. ASP-DAC 2005*, pp. 151–156 (Jan. 2005).

[2] M. Vetromille, L. Ost, C. A. M. Marcon, C. Reif, and F. Hessel: “RTOS scheduler implementation in hardware and software for real time applications,” in *Proc. RSP ’06*, pp. 163–168 (June 2006).

[3] P. Kohout, B. Ganesh, and B. Jacob: “Hardware support for real-time operating systems,” in *Proc. CODES+ISSS ’03*, pp. 45–51 (Oct. 2003).

[4] T. Nakano, Y. Komatsudaira, A. Shiomi, and M. Imai: “Performance evaluation of STRON: A hardware implementation of a real-time OS,” in *IEICE Trans. Fundamentals*, vol. E82-A, no. 11, pp. 2375–2382 (Nov. 1999).

[5] N. Maruyama, T. Ishihara, and H. Yasuura: “An RTOS in hardware for energy efficient software-based TCP/IP processing,” in *Proc. SASP 2010*, pp. 58–63 (June 2010).

[6] Y. Oosako, N. Ishiura, H. Tomiyama, and H. Kanbara: “Synthesis of full hardware implementation of RTOS-based systems,” in *Proc. RSP 2018*, pp. 1–7 (Oct. 2018).

[7] 六車 伊織, 石浦 菜岐佐, 安堂 拓也, 冨山 宏之, 神原 弘之: “RTOS 利用システムのフルハードウェア化におけるサービス処理機能の集約,” 信学技報, VLD2020-75 (Mar. 2021).
[8] T. Ando, I. Muguruma, Y. Ishii, N. Ishiura, H. Tomiyama, and H. Kambara: “Full hardware implementation of RTOS-based systems using general high-level synthesizer,” in *Proc. SASIMI 2022*, pp. 2–7 (Oct. 2022).
[9] H. Minamiguchi, N. Ishiura, H. Tomiyama, and H. Kanbara: “Automatic generation of management module for full hardware implementation of RTOS-based systems,“ in *Proc. ITC-CSCC 2023*, pp. 473–478 (June 2023).
[10] Shinya Takamaeda-Yamazaki: “Pyverilog: A Python-based Hardware Design Processing Toolkit for Verilog HDL“ in *Proc. ARC 2015*, pp. 451–460 (April 2015).
[11] M. Nakahara and N. Ishiura: “Arrival order processing of service requests in full hardware implementation of RTOS-based systems,” in *Proc. ITC-CSCC 2023*, pp. 467–472 (June 2023).
[12] TOPPERS project: “TOPPERS 第 3 世代カーネル (ITRON 系) 統合仕様書 Release 3.6.0,” `https://www.toppers.jp/docs/tech/tgki_spec-360.pdf` (Mar. 2023).