リアルタイム性とスループット向上を両立する完全仮想化の研究

特定チーム（課題番号：167201）
研究期間：平成 28 年 7 月 28 日～平成 29 年 3 月 31 日
研究代表者：請園智玲　研究員：荒木光一

1．研究背景

リアルタイム処理が要求されるコンピュータ組込み機器は数多くある．その中でも自動車や航空機の制御や，医療機器はリアルタイム性能が直接的に人命にかかわるためリアルタイム性（ハードリアルタイム）の保証に注意深く設計されなければならない．この場合の開発工数の大半は検証のために使われ，それが開発期間の支配的要因となる．一方，近年の製品のリリースサイクルはますます早まり，組込み向けコンピュータシステムの効率的な開発が必要とされている．

組込みシステムの開発において，製品のリリースサイクルを早めるための組込み仮想化の技術が研究されている．組込み仮想化は既存システムのソフトウェアの下位レベル（OSなど）から上位レベル（アプリケーション）までのすべてを異なるハードウェアに移植する技術である．仮想化はクラウドなどに用いられ，エンタープライズシステムなどにすでに適用されているが，コンピュータの処理性能が低い組み込みシステムに適用するためには，VMM（Virtual Machine Monitor）と呼ばれる仮想化の制御ソフトウェアのソフトウェアオーバヘッドがリアルタイム性を阻害する要因となり，ハードリアルタイム性が要求される組込みアプリケーションへの適用が難しい．

このような背景から，本研究はVMMを完全にハードウェアで実装し，ソフトウェアオーバヘッドを仮想化から排除することで，ハードリアルタイムシステムへの仮想化の適用を可能にし，ハードリアルタイムシステムの開発工数の削減を目的とする．

これまでの研究で，組込み仮想化を実現するための，各種ハードウェアコンポーネントを提案してきた．本研究では，リアルタイムタスクスケジューラのハードウェア化に着目し，ハードウェアリアルタイムタスクスケジューラをVMMのCPUリソース管理機能として応用することで，資源管理のハードウェアコンポーネントを構築する．これを本研究ではハードウェアドメインスケジューラと呼ぶ．

2．布線論理タスクスケジューラ

本研究は布線論理でタスクスケジューラを実現する際の回路規模に着目する．通常，汎用システムのタスクスケジューラはシステム上で動作するタスク数に比例して，ワーキングセットサイズが変動する．主記憶資源を利用できない布線論理でタスクスケジューラを実装するためには，この可変サイズのワーキングセットを固定サイズで実現する必要がある．このため，布線論理のタスクスケジューラを用いると，システム上で実行可能なタスク数に上限ができる．この制限は，汎用システムにおいては受け入れがたい制約となる．一方，組込みシステムは，製品の詳細設計が確定した時点でシステムが生涯実行するタスクセットが静的に決まる特性を持つ．このため，設計したタスクセットが与えられた布線論理のタスクスケジューラの容量で実行可能か否かを事前に知ることが可能となる．この組込みシステムの特性から，布線論理のタスクスケジューラは大きなシステム上の制約とはなりえない．

比較的大規模な組込みシステムを設計する上で，布線論理を用いてタスクスケジューラを実現する場合，タスクスケジューラが扱うワーキングセットを保持するために，チップ内に大量の記憶素子が必要とされる．もし，現在の半導体の集積度において，現実的な回路規模で実現可能であるならば，布線論理で実現されたタスクスケジューラは組込み向けSoCの重要な機能部品となりえる．本稿は比較的単純なタスクスケジューラを持つ組込みOSであるμITRON仕様[1]のμT-Kernel[2]のスケ

ジューリングアルゴリズムとデータ構造を対象とする．そして，近年の高集積化による恩恵で，どの程度のタスク数に対応するタスクスケジューラが布線論理によって実装可能かを，FPGA を対象とした論理合成の結果から考察することを目的としている．

3．μITRON タスクスケジューラ

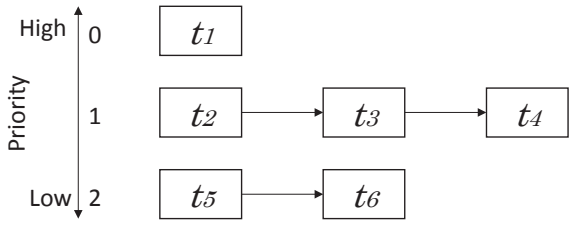

図１　タスクスケジューラのデータ構造

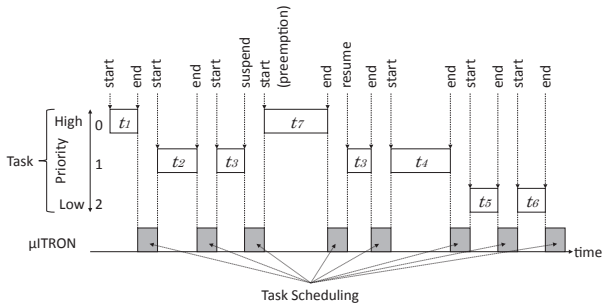

図２　タスクスケジューラの動作

μITRON は TRON プロジェクトが策定している組込みOSの仕様である．μITRON に準拠した組込みOSは家電製品や工業制御機器など様々な組込みシステムで利用されている．

タスクスケジューラは組込みOSの重要な一機能であり，タスクの実行順序を決定する．μITRON ではプリエンプティブな優先度ベーススケジューリング方式を採用している．この方式はレディーキューと呼ばれる実行可能なタスクを扱うキューを用いて優先度の高いタスクから順に実行する．また，実行中のタスクより高い優先度のタスクがレディーキューに追加された場合，実行中のタスクを中断して追加されたタスクを実行させる．この実行権横取りはプリエンプションと呼ばれる．

例として，図１に，３つの優先度で構成するレディーキューのデータ構造を示す．t はタスク，その添字はタスクIDである．各優先度のタスクはリストとして管理されている．新規タスクはその優先度のリストの末尾に追加される．

図２に，μITRON タスクスケジューラよるタスクの実行例を示す．レディーキューの初期状態は図１とし，t_3の実行中に優先度０のt_7がレディーキューに追加されたときの実行例である．灰色の四角はμITRON 仕様の組込みOSによるタスクスケジューリングの処理を示す．タスクスケジューリングは，各タスクがタスクの追加や完了などμITRON のサービスコールが呼び出されたときに行われる．まず，優先度０のt_1が実行される．t_1完了後，μITRON タスクスケジューラはレディーキューのt_1を削除し，リスケジューリングする．このとき，優先度０にはタスクがないため優先度１のt_2が実行され，その完了後にt_2の削除とリスケジューリングが行われる．その後に実行されるt_3は，サービスコールで優先度０のt_7をレディーキューに追加する処理を行う．t_7の追加後，リスケジューリングが行われ，t_3を中断してt_7を開始する（プリエンプション）．t_7完了後，t_3を再開する．以降，t_4→t_5→t_6の順に各タスクが完了す

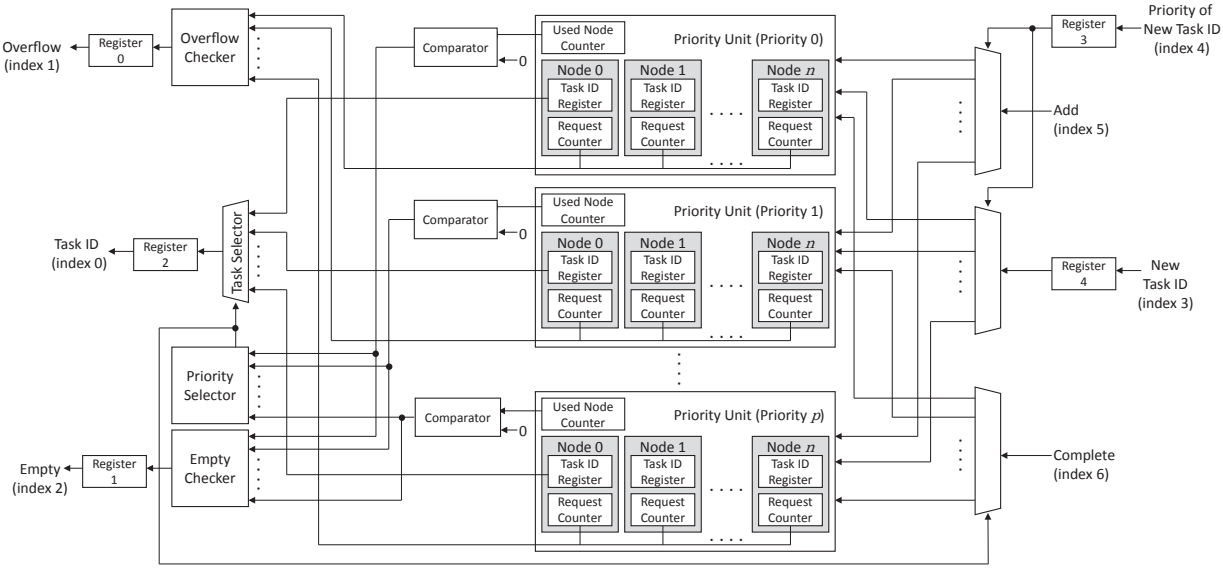

図３　提案手法のハードウェア実装図

る度にリスケジューリングしながら，それらを処理する．

μITRONのタスクスケジューリングはタスクの追加と完了の度に行われる．また，その処理時間は基本的に実行可能なタスク数により変動する．そのため，タスクスケジューリングの安定しないオーバヘッドがリアルタイム性能に影響を与える．特に，ハードリアルタイムシステムではデッドラインミスを起こす原因ともなりえる．本稿では，このμITRONタスクスケジューリングを布線論理で行った場合の実現可能性について議論する．

4．布線論理によるタスクスケジューラの実装

図3に，布線論理によるμITRONタスクスケジューラを示す．入出力名の下に記載されているindex番号はメモリマップドI/Oのベースアドレスからのオフセットを意味している．Priority Unit内のpは優先度の最大値（Priority 0 が最も優先度が高い）を，Node内のnはNodeの最大インデックスを示す．したがって，各Priority UnitのNode数はn+1である．μITRONタスクスケジューラの入力は新規タスクID，その優先度，タスク追加信号とタスク完了信号である．出力はオーバーフロー信号，エンプティ信号と実行すべきタスクのIDである．Register 0〜4, Used Node Counter，Task ID RegisterとRequest Counter以外は組み合わせ回路として実装されている．組込みOSからタスク追加の通知，タスク完了の通知が1になってから1クロック後にμITRONタスクスケジューラは次に実行すべきタスクIDを出力する．

Priority UnitとNodeは図1のレディーキューのデータ構造に相当する．Priority Unitは各優先度のタスクをFCFSで管理する機構であり，Used Node Counterとn+1個のNodeで構成されている．n+1個のNodeは各優先度のリストに相当し，Priority Unit内の各Nodeはデータを隣のNodeに転送できるように連結されている．Node0のタスクIDは各優先度において優先的に実行するタスクであるため，次に実行するタスクIDを選択するTask Selectorと接続されている．Used Node CounterはタスクIDを保存しているNode数をカウントし，タスクIDを保存しているNodeの末尾と，その優先度のタスクの有無を確認するために利用される．Node内のTask ID RegisterにはタスクIDが保存される．Request CounterはTask ID Registerに保存されているタスクIDの起動要求回数をカウントする．例えば，既に保存されているタスクIDを新規タスクIDとして追加する場合，他のNodeに新規タスクIDを保存せず，新規タスクIDと同じタスクIDが保存されているNodeのRequest Counterをインクリメントする．

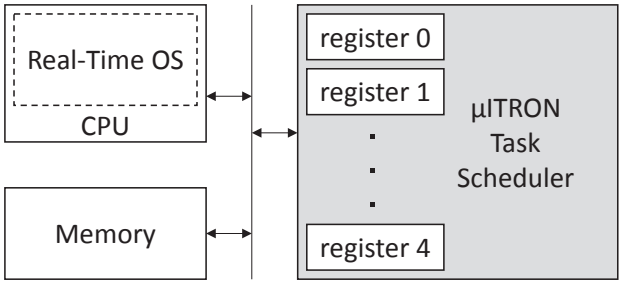

図4　提案手法を含むシステムモデル

両サイドのRegister0からRegister4は図4に示す提案手法のシステムモデルのレジスタに対応しており，組込みOSがアクセスするレジスタである．提案手法はメモリバスに接続され，それぞれのレジスタがメモリマップドI/Oでソフトウェアに管理される．ComparatorはUsed Node Counterの値と0を比較してタスクが存在する場合は1となる．Priority Selectorは各Comparatorの出力を元に，次に実行する優先度を選択する．Empty Checkerも各Comparatorの出力からタスクIDの有無を確認する．スケジューラ内に1つでもタスクIDが存在する場合，Empty Checkerは1を出力する．Overflow Checkerは全Request Counterのオーバーフローを確認する．1つでもRequest Counterがオーバーフローした場合，出力は1になる．

μITRONタスクスケジューラ全体のNode数は，組込みシステムの詳細設計で決定されたタスクセットがシステムに許容されるか否かを決定づける1つの指標となる．ある優先度のタスクの数が当該優先度のNode数を超えた場合，スケジューリングできない状況になるため，タスクの設計者は必ずそのNode数を知る必要がある．逆に言えば，Node数を超えなければ，必ずスケジューリングができることを保証できる．これはRequest Counterを導入したために発生した特性である．Request Counterの導入で，厳密にはFCFSを守ることはできなくなる．しかしながら，タスクスケジューア化されたタスクスケジューラの恩恵を得ることができる．

μITRONタスクスケジューラにタスクを追加する場合，組込みOSはそのタスクIDと優先度をそれぞれRegister 3とRegister 4に保存した後に，Addを1クロック間だけ1にする．優先度からタスクIDを保存するPriority Unitを決定し，Addの信号とタスクIDをそのPriority Unitへ送る．このとき，追加するタスクIDがNodeに保存されていなければ，Used Node Counterの値と同じインデックスのNodeにタスクIDを保存する．同時に，Used Node Counterをインクリメントする．一方，既にタスクIDがNode保存されている場合は同じタスクIDが保存されているNodeのRequest Counterをインクリメントする．この場合，Used Node Counterの

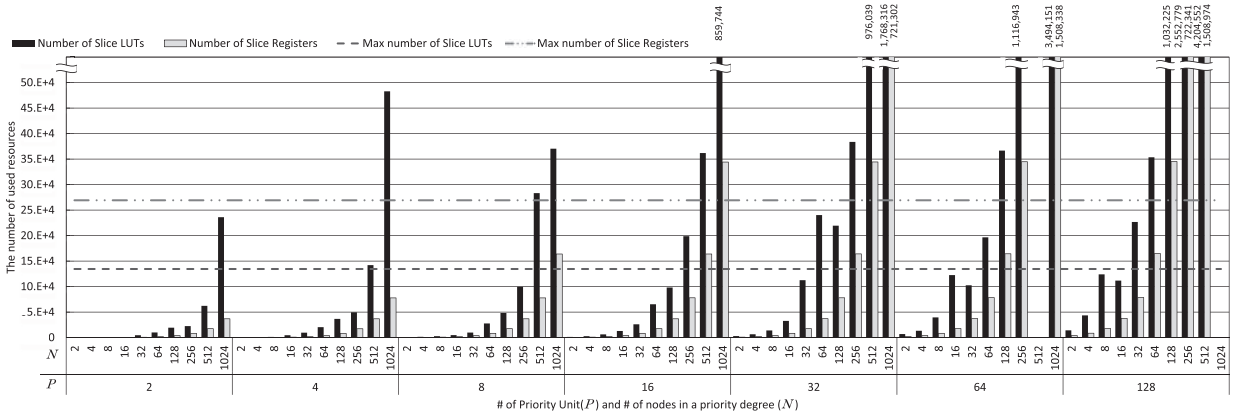

図５　提案手法のハードウェアサイズ

インクリメントは行わない.

各優先度のNode 0と接続されているTask Selectorは，Priority Selectorからの出力を元に次に実行すべきタスクIDを選択し，Register 0にタスクIDを保存する．プリエンプションが発生する場合，Priority Selectorは常にComparatorの出力を監視しているため，即座に新規タスクの優先度を出力でき，Register 0に新規タスクIDを保存できる.

実行していたタスクが完了して，組込みOSがCompleteを１クロック間だけ１にした場合，Priority Selectorからの優先度を元にタスクIDを削除するPriority Unitを選択し，そのPriority UnitへCompleteの信号を送る．μITRONタスクスケジューラから削除するタスクIDのRequest Counterが１ならば，Node 0以外のNodeは隣のNodeへデータを移動する（Node 0はNode １の内容に上書きされ，Node 0に存在したデータが消える）．同時に，Used Node Counterをデクリメントする．削除するタスクIDを持つRequest Counterが２以上ならば，Request Counterをデクリメントして，Used Node CounterをPriority Unit内のNodeのインデックスとして使用し，その位置にデータを移動する．そして，他のNodeは隣のNodeへデータを移動する．これは，ソフトウェア実装において，リストの末尾に繋ぎ直す動作をハードウェアで実現するための仕組みである. この場合，使用しているNode数は変わらないため，Used Node Counterはデンクリメントされない.

Used Node Counterのデクリメントにより，そのカウンタが０になればComparatorの出力は０に変化するため，Priority Selectorが出力する優先度も変化する．一方，Used Node Counterのカウンタ値が１以上の場合，そのPriority Unitにはまだタスクが存在するため，Priority Selectorの出力は変化しない．Task Selectorは，タスク追加の処理と同様に，Priority Selectorからの優先度を元に次に実行すべきタスクIDを選択し，Register 0にタスクIDを保存する.

５．使用リソース及び動作周波数の評価

FPGA専用の論理合成ツールで，布線論理によるμITRONタスクスケジューラの使用リソース数と動作周波数を取得する．本節では，それらの結果から布線論理によるμITRONタスクスケジューラの実現可能性について考察する．μITRONタスクスケジューラはVerilog HDLで実装し, 論理合成はLinuxマシン（CPU: Intel Core i7 3829 3.6GHz，メモリ：64GB，OS：CentOS 6.6）で行った．ターゲットFPGAはXilinx社Artix-7のXC7A200T[3]とした．論理合成ツールとしてXilinx社ISE14.4 WebPackのXST14.4を用い，論理合成のオプションはDesign GoalをBalanced，StrategyをXilinx Defaultとした．様々な規模の使用リソース数と動作周波数を取得するために，Priority Unit数P（=p+1）は２から128までの２の累乗数とした．各Pにおける Priority Unit内のNode数N（=n+1）は２から1024までの２の累乗数とした．以降，便宜上μITRONタスクスケジューラの規模を{P, N}と定義する．各規模のμITRONタスクスケジューラで管理できるタスク数はN×Pである．Request Counter内のレジスタは5bit幅に固定し，管理できる実行要求回数を31回までとした.

図５に，μITRONタスクスケジューラの使用Slice LUT数と使用Slice Register数を示す．{64, 512}と{128, 1024}は論理合成中にLinuxマシン上でスタックオーバーフローが発生したため，使用リソース数を取得できなかった．点線はXC7A200Tが搭載しているSlice LUT数とSlice Register数を示す.

布線論理のμITRONタスクスケジューラはSlice RegisterよりSlice LUTを多く利用する．これは，μITRONタスクスケジューラ内はTask SelectorやPriority SelectorなどSlice LUTを利用する組み合わせ回路の機構の実装面積が支配的であることを示している．XC7A200Tのリソース数（点線）と使用リソース数の比較をすると，XC7A200TにμITRONタスクスケジューラを実装する場合，使用Slice LUT数で実装可能

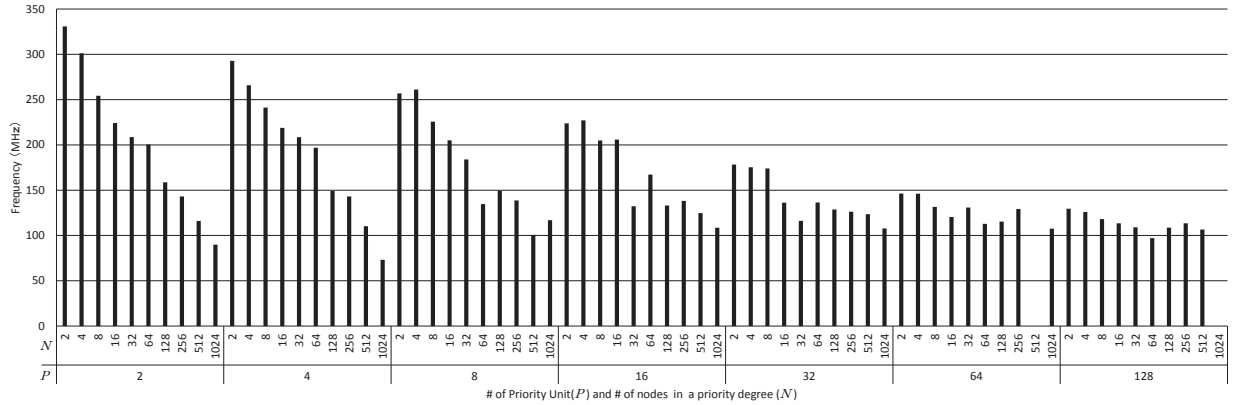

図６　提案手法の動作周波数

な規模が決まることが分かる．

各Ｐの使用Slice Register数はＮの増加に伴って増加する．μITRONタスクスケジューラ全体のNode数が同じ場合（タスク数Ｎ×Ｐが同じ），使用Slice Register数はほぼ同等となった．これは，Nodeの総数が記憶素子の実装面積の大きさを決定づけている．

各Ｐの使用Slice LUT数は，一部を除きＮの増加に伴って増加する．{8, 512}と{8, 1024}間の使用Slice LUT数の増加率は，XST14.4のLow Level Synthesisでの最適化により，他のＰより低くなった．{32, 128}，{64, 32}と{128, 16}は，それぞれ{32, 64}，{64, 16}と{128, 8}と比較して，Ｎは２倍となっているが使用Slice LUT数は減少している．これらもLow Level Synthesisでの最適化の影響である．同Node数での比較では，優先度設定が多い（Ｐが大きい）方がSlice LUTの使用数が少ない傾向がある（e.g. {2, 512}，{4, 256}）．これはPriority Unit内のノード間接続を制御する組み合わせ回路がSlice LUTを消費する大きな要因であることを示しており，同一優先度のタスクが多くなれば，回路規模が大きくなることを示している．

図６に，μITRONタスクスケジューラの動作周波数を示す．{64, 512}と{128, 1024}の動作周波数は，使用リソース計測と同様のスタックオーバーフローで取得できなかった．Ｐ=２とＰ=４の規模では，それらの動作周波数はＮが増加するにつれて低下する．Ｐ=８においては，XST14.4のルーティングの影響により，大きな規模の動作周波数がそれより小さい規模より高くなる場合がある（e.g. {8, 512}と{8, 1024}）．

Ｐが増加するにつれて，動作周波数の最大と最小の差が小さくなる傾向がある．P=2での最大と最小の差は240.95MHzであるが，Ｐ=16では118.42MHz，Ｐ=128では32.47MHzとなる．Ｐが増加すると，Priority Unitに接続されている機構は大規模化し，動作周波数は低下する．そして，Ｐが大きい場合，Ｎの増加による動作周波数の変化は小さい．この結果，動作周波数の最大と最小の差は小さくなる．

このことから，回路構成において，Priority Unit間の接続部がクリティカルパスになりやすく，優先度設定が多過ぎると（Ｐを大きくし過ぎると），全体のNode数が少ないのにも関わらず，十分な性能が得られない可能性があることがわかる．

本計測では，ほとんどの構成で100MHz以上の動作周波数を達成している．現在のFPGAの現実的な動作周波数から考えると，本稿のタスクスケジューラは十分な性能が達成できる可能性をもつ極めて単純な回路構造であるといえる．

６．結論

本稿では，布線論理によるμITRONタスクスケジューラの実現可能性についてFPGAを対象とした論理合成結果から議論した．実行可能状態のタスクを管理するμITRONタスクスケジューラの場合，その回路規模は組込み向けSoCの機能部品として提供するために現実的な規模であることを述べた．また，その性能は機能部品として提供可能なレベルである可能性をもつことを示した．加えて，スケジューリングの遅延が一定となることで，実行時間予測が容易になることも述べた．今後は，μITRONの全タスク状態の遷移に対応するとともに，ソフトウェア実装時のオーバヘッドを計測し提案手法の性能を相対的に検証していく．

参考文献

[1] 坂村健 監修，μITRON4.0 標準ガイドブック，社団法人トロン協会（編），パーソナルメディア株式会社，東京，Nov. 2001.

[2] T-Engineフォーラム，http://www.t-engine.org/ja/what-is-t-kernel/mt-kernel, 参照 May 29, 2015.

[3] Xilinx Inc.,“Xilinx 7 Series FPGAs Overview, Product Specification,”http://www.xilinx.com/support/documentartion/data_sheets/ds180_7Series_O

verview.pdf, May 27, 2015.

研究業績

現在数本の論文を執筆中である.