Vol.2015-SLDM-170 No.7
Vol.2015-EMB-36 No.7
2015/3/6

FA コントローラのタスク処理方式による性能改善

羽田野一貴[†1]　上村哲生[†2]　塩見彰睦[†3]

株式会社エヌエスティーと共同開発の FA(Factory Automation)コントローラ NX2000 には複数のユニットで協調して制御プログラムを実行するマルチユニット動作と PC のユーザアプリと連携する機能を有している．制御フローや通信コマンド処理の中には入出力装置のハードウェアや通信の性質により，実行に必要なデータの取得に待ち時間が発生し，処理時間が長くなる．処理時間の長い処理が CPU を専有してしまうと，他の制御フローや通信コマンドの応答時間が低下する問題が発生する．

本研究では，制御フローだけでなく通信コマンドの，データ取得待ち時間が発生する処理に対して，データ要求後実行を中断し他の制御フローや通信コマンドの実行に CPU を明け渡すタスク管理を適用した．これにより，データ取得待ち時間による処理時間の長期化を防ぐことで制御フロー及び通信コマンドの応答時間の改善を行った．

制御フロー及び通信コマンドの応答時間を計測したところ，提案手法を用いることで応答時間が改善されたことを確認した．

The Performance Improvement by Task Processing Method of FA Controller

KAZUKI HATANO[†1]　TETUO UEMURA[†2]
AKICHIKA SHIOMI[†3]

1. はじめに

本章では，株式会社エヌエスティーと共同開発の FA（Factory Automation）コントローラ NX2000 について説明を交えながら本研究の背景，先行研究の概要と問題点を述べて，本研究の目的を示す．

1.1 研究の背景

NX2000 は，先行研究での複数のユニットで協調して制御プログラムを実行するマルチユニット動作の実現，そして新しく PC のユーザアプリと連携する機能を有している．ユニットとは NX2000 シリーズに対して使用される構成単位のことであり，ひとつのユニットは通信マスターボード 1 枚と拡張ボード（7 種，最大接続数 8 枚）から構成される．そして，NX2000 の制御プログラムは複数の制御フローで構成される．制御フローとはユーザがフローチャート言語で記述する一連の制御処理のことである．制御フローは処理を示すステップと，遷移を表す線により構成される．

マルチユニット動作やユーザアプリとの連携機能により，機械制御用の制御フローを実行中の NX2000 に対して，他ユニットや PC からデータの取得要求や格納処理のコマンドが送信されるようになった．そのため，NX2000 は制御フローの実行と並行してコマンドの実行を行う．送られてきたコマンドの処理時間が長い場合，制御フローの応答時間が遅延する問題が発生するようになった．以後 PC から送信されるコマンドを通信コマンドと呼び，マルチユニット動作時に他ユニットから送信されるコマンドをイベントコマンドと呼ぶ．

通信コマンドとイベントコマンド処理の中には，処理に必要なデータを取得するのにユニット間通信（IEEE1394）やネットワーク通信（Ethernet）を使用する処理が存在する．それらの処理はデータの取得に通信が伴うため，取得完了までの処理時間が長くなってしまう．処理時間の長い処理が CPU を専有してしまうと，制御フローの応答時間が遅延する問題が発生する．

制御フローや通信コマンドの応答時間が遅延する場合，NX2000 が制御する機器の制御精度が低下し，品質が低下するのみならず事故の原因となる可能性が高くなる．

以上のことから，FA コントローラ NX2000 において，処理時間の長い通信コマンド，イベントコマンドが実行される場合に制御フローの応答性時間が遅延する問題の解決が求められている．

1.2 先行研究の概要と課題

先行研究として NX2000 を対象に行われた独立型汎用 FA コントローラの設計開発[1]がある．畠山の研究では，制御の実行中は PC を切り離して動作可能とするためのスタンドアロン機能の開発を行った．これにより，システムの動作環境におけるロバストネス性能が向上した．さらに，実行する制御フローのステップがネットワーク通信

†1 静岡大学大学院情報学研究科情報学専攻
Shizuoka University
†2 株式会社エヌエスティー
NST(Nippon System Technology)
†3 静岡大学大学院情報学研究科
Shizuoka University

Vol.2015-SLDM-170 No.7
Vol.2015-EMB-36 No.7
2015/3/6

(Ethernet)を使用する処理である場合，制御フローの実行権をとりあげ他の制御フローに実行を移すタスク管理手法を実現していた．このタスク管理手法により制御フローの処理が他の制御フローの応答時間遅延を改善した．

次に，先行研究として分散協調型 FA コントローラの開発[2]がある．宮澤の研究では，システムの大規模化に対応するため，スタンドアロン機能を備えた複数の NX2000 が協調して制御フローを実行するマルチユニット動作の開発を行った．これにより，システムの大規模化に対応できるようになった．そしてマルチユニット動作において，制御フローのステップがユニット間通信（IEEE1394）を使用して他のユニットからデータを取得する処理である場合，データ要求後に制御フローの実行権をとりあげ他の制御フローに実行を移すタスク管理手法を実現していた．

2 つの先行研究は個々の状況に対して制御フローの応答時間遅延の発生を防ぐことに成功していた．しかし，図 1 で示す処理を行った場合，制御フローの応答時間が遅延が生じてしまう．図 1 の処理は，マルチユニット状態においてユニット 2 がタッチパネルのデータを取得すために，ユニット 1 に対してデータ取得イベントコマンドを送信する処理である．

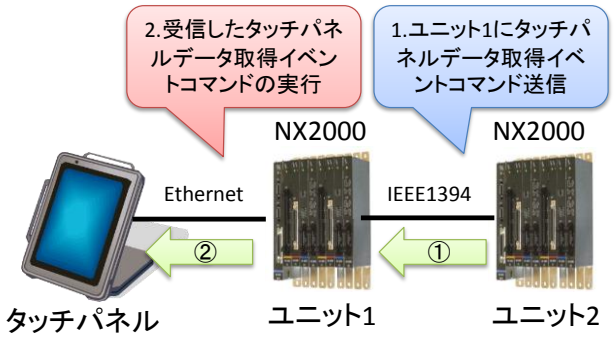

図 1 制御フローの応答時間遅延が発生する処理内容

先行研究では制御フローの処理時間長期化による他の制御フローの応答時間遅延に対する対策は行われたが，通信コマンド，イベントコマンドの処理時間長期化による制御フローの応答時間遅延問題が未解決であった．1.1 節で記述したとおり，制御フローの応答時間遅延は FA コントローラにおいて重大な問題となり得る．そのため，本研究では図 1 の処理を実行する場合でも制御フローの応答時間が遅延する問題の発生を防止する応答時間の改善を目指す．

1.3 本研究の目的

本研究の目的は，FA コントローラ NX2000 に対して応答時間改善システムを適用し，制御フロー実行中に送信されてきた処理時間の長い通信コマンドとイベントコマンドが実行される場合でも，制御フローの応答時間が“5ms 以内”となることを実現することである．

2. 応答時間改善手法の検討

本章では，まず 2.1 節で応答時間改善を行う手法について検討する．そして，2.2 節と 2.3 節では検討した手法を NX2000 に適用するために必要な条件やシステムの適用範囲について記述する．

2.1 応答時間改善手法の検討

NX2000 では，現状最大 100 個の制御フロータスクを記述し同時に起動及び実行を行うことができる．制御フロータスクとは，NX2000 で実行するメインルーチンに対応する制御フローのことである．個々の制御フロータスクに対して RTOS（Real Time Operating System）を 割り当てた場合，タスク数が増えるたび RTOS のオーバーヘッドが膨大になると考えられる．さらに，拡張ボードの中には排他制御できないものもあり，排他制御できない処理に関しては途中で実行権の剥奪が発生しないようにする必要がある．そのため，排他制御できない処理に対して実行割り当て時間が大きくなる．タスク毎の割り当て時間がばらついてしまうと応答時間確保が難しくなる．そのため，RTOS を用いるのは NX2000 の応答時間改善には適切で無いと考える．

次に，ノンプリエンプティブマルチタスク方式を適用した場合について検討する．ノンプリエンプティブ方式とは，OS を用いずにアプリケーションが自主的に CPU を解放することで，複数アプリケーション間で制御を切り替えるマルチタスク方式である．この方式の利点として，RTOS を使用しないため OS のオーバーヘッドが無く高速にタスクを切り替えることが可能であることが挙げられる．

そして，ファームウェア上でタスクの切り替え条件を記述することができる．そのため，フロータスクのステップ完了後や通信コマンドの実行後といった処理の区切りでタスクを切り替えることが可能となり，制御対象によって排他制御ができない問題を回避することができる．さらに，データ取得待ちの発生を検出もしくは判定ができれば，データ取得待ちとなったタスクの実行を中断し次のタスクへ切り替えることができる．データ取得待ち時間を回避できれば制御フロータスク及び通信コマンドの実行開始が遅延する問題を解決できると考える．

以上の理由から，本研究で開発するシステムにはノンプリエンプティブマルチタスク方式を採用し，処理時間が長くなる処理に対して実行を一時中断することで応答性改善を行う．

2.2 タスク実行中断の基準選定

実行を中断するのは，処理時間の長い制御フローや通信コマンド，イベントコマンドを対象とする．処理時間の長

い処理は，主に処理に必要なデータを取得するのにユニット間通信（IEEE1394）やネットワーク通信（Ethernet）を使用する処理である．NX2000 には Ethernet ケーブルで接続されたタッチパネルと IEEE1394 ケーブルで接続された他ユニットからデータ取得を行う処理がある．

以上の理由から，本研究ではこの 2 つの処理に対して実行の中断を行い，データ取得が完了したら再開するシステムを適用する．

2.3 タスク切り替えのタイミングと実行再開方法の検討

実行権を剥奪しタスク切り替えを行うタイミングは，データ取得要求を対象デバイスに送信した後に行うのが適切であると考える．なぜならば，データ取得要求を対象デバイスに送信した後は要求データが返送されるまで実行する処理が無いためである．

中断されたタスクは要求したデータ取得が完了したことを確認したら実行を再開する．

3. 実現方法

本章では 3.1 節にて本研究で開発する応答時間改善システムの概要を解説し，3.2 節から 3.7 節にかけて個々の機能ついて解説する．

3.1 応答時間の改善方針

本研究では，制御フローのみならず通信コマンド，イベントコマンドに対しても，データ取得待ち時間の発生する処理はデータ要求後に実行を中断し他の制御フローや通信コマンドの実行に CPU を明け渡すタスク管理機能を適用した．このタスク管理機能を実現するために，実行中断処理の判定機能や取得対象値の一時保存変数の作成機能，中断したタスクの再開機能を通信コマンドとイベントコマンドに対して実装した．

これにより，データ取得待ちによる処理時間の長い処理が CPU を専有することを防ぎ, 制御フローの応答時間が遅延する問題の解決を行った．

3.2 タスクの中断と再開機能

制御フローのステップ実行は図 2 の状態遷移図の処理を用いて行っている．制御フローの中断と再開機能は制御ステップ実行を行うクラスを使用するフロータスクコントローラクラスで実現されている．

ステップ実行クラスは，フェッチ，ロード，ストア，一時停止の 4 状態を持つ．デバッガモードでステップにブレークポイントを設定されていない場合，次の処理を行う．フェッチ状態ではステップを読込み，処理に必要な変数や値を後述する入出力パッケージに対して要求しロード状態に遷移する．ロード状態では，入出力パッケージに要求した値がそろうまで待つ．要求した値の取得が完了するとステップを実行する．値を格納するのであれば入出力パッケージに格納要求を行い，ストア状態に遷移する．ストア状態では，結果が格納されるのを待つ．格納されれば，フェッチ状態に遷移する．

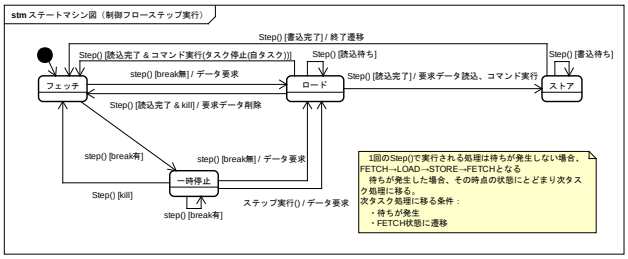

図 2 制御フローのステップ実行状態遷移図

一回のステップ実行につき，フェッチ，ロード，ストアを一通り行う．ただし，入出力パッケージによりデータ取得待ち処理であると判定されると，ロード状態で実行を中断し，フロータスクコントローラクラスに実行権を渡す．実行権を渡されたフロータスクコントローラクラスは次の制御フロータスクや他の処理を行う．そして，再び実行順序が回ってきたところで要求したデータが揃っているかどうかを判定し，揃っていればロード状態から実行を再開する．

3.3 実行権剥奪処理の判定機能

データ取得待ち時間の発生する処理に対して，データ要求後に実行を中断し他の制御フローや通信コマンド，イベントコマンドの実行に CPU を明け渡すタスク管理を適用するためには，実行する処理がデータ取得待ち時間の発生する処理であるか否かを判定する必要がある．データ取得待ち時間の発生有無の判定には，NX2000 ファームウェアにおける入出力に関する処理を実行するクラス群をまとめた入出力パッケージを用いる．制御フローや通信コマンド，イベントコマンドの実行において，いずれもはじめに入出力パッケージに対して実行に必要なデータの取得を行う．入出力パッケージを用いてデータの取得を行う場合，統一されたインターフェースを通じてアクセスを行う．データ取得対象リソースの指定方法はユニット ID，ボード ID，開始変数 ID，ID 数の 4 値により行う．ユニット ID によりシステム内のユニットを指定し，ボード ID によりユニットに接続されたボードまたはデバイス(通信マスタボード，拡張ボード，タッチパネル) を指定する．変数 ID は各ボードの所有するリソースに対して割り振られた番号である．ID 数は，開始変数 ID から連番で何回アクセスを行うかを指定することにより，複数リソースに対するアクセスを行うことができる．

2.2 節で記述したとおり，処理時間が長い処理は，主に

処理に必要なデータを取得するのにユニット間通信（IEEE1394）やネットワーク通信（Ethernet）を使用する処理である．そのため，ユニット ID とボード ID が他ユニットやタッチパネルを示す値であるかどうかを判定する．それらの ID が他ユニットやタッチパネルを示す場合，データ取得要求処理を行った後タスクの実行権を剥奪する．

3.4 取得対象値の一時保存変数作成機能

データ取得時に待ち時間が発生する場合，制御フローは要求した値がすべて取得完了するまで実行を中断したままである．その間，取得完了した値はどこかに記憶しておかなければならない．

取得した値の一時記憶を制御フローが行う場合，使用されうる最大量の記憶領域をタスク毎に用意する必要がある．しかし，すべてのタスクが最大量の一時記憶領域を同時に使うことはかなり稀である．そのため，この方法での実装はメモリ資源の無駄となる．メモリリークやヒープ空き領域の断片化が発生する恐れもある為，動的確保は行わない．代わりとして，一時記憶領域の提供を行うクラスを実装する．データ取得要求時にこのクラスを用いて一時記憶領域を獲得する．そして，データ要求先に一時記憶領域のアドレスを渡し，格納させる．データ要求を行ったタスクは実行順序が回ってくる度に要求データの格納が完了したか否かを確認する．データの格納完了が確認できたら実行を再開する．

3.5 NX2000 通信処理機能の拡張開発

通信コマンドとイベントコマンドの解析・実行処理において，開始時までに受信したコマンドは全てまとめて実行する．この時，中断したコマンドの後に別のコマンドが来ている場合，そのコマンドを実行する．

通信コマンドの実行によって制御フロータスクの応答性が大きく損なわれないように，PC は NX2000 に対して送信したコマンドのレスポンスが送り返されるまで次のコマンドを送信しない．マルチユニット動作においても，IEEE1394 を用いたデイジーチェイン接続であるため，ほぼ同じタイミングで受信するコマンド数はあまり多くない．そして，コマンドの中には軸の同時起動のようにできるだけ早く実行されることが望まれるものも存在する．そのため，まれに起きる複数の通信コマンド実行による制御フローの応答時間の遅延よりも受信コマンドの応答性を優先した．

PC との通信で使用する Ethernet 通信処理では，今後の通信端末の増加に対応しやすくし，コマンドのメモリコピーによる処理量の増加を減らすため，通信端点毎の受信データの先頭アドレスを登録・参照するリングバッファを用いた設計を行った．処理の流れは図 3 に示す通りである．

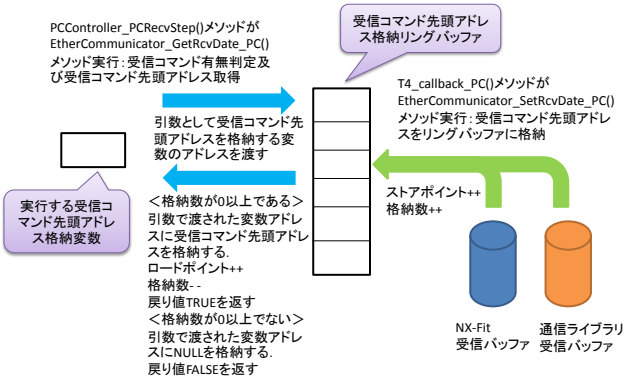

図 3 NX2000 受信データ格納処理の概要図

3.6 PC から送信される通信コマンドの実行タスクの実現

通信コマンドは NX2000 専用プログラム開発ツール NX-Fit によるデバッガモードでの実行機能を提供している．そのため，通信コマンドの実行は制御フローと同じステップ実行クラスを用いて行う．よって，実行権剥奪の判定とコマンド実行中断と再開は制御フローと同様の方法を用いている．

しかし，通信コマンドの実行は一度きりであり，実行が完了すると破棄される．通信コマンドの実行時には受信バッファからコマンドデータの取得と初期化処理を行う．そのため，通信コマンド実行時に実行再開コマンドと通常実行コマンドとを区別し，実行再開時にはコマンドデータ取得と初期化処理を行わないようにする必要がある．

通信コマンド実行時に実行再開コマンドと通常実行コマンドとを区別するためタスクの状態を「WAIT,EXECUTE, PAUSE」の 3 つとした．WAIT 状態は実行コマンドが無い状態であり，通信コマンドの受信があると．EXECUTE 状態へ遷移する．そして，EXECUTE 状態で実行権剥奪が起きた場合，PAUSE 状態へと遷移する．PAUSE 状態から実行を再開する場合はコマンド実行初期化処理を行わないようにした．タスク状態の遷移図は〜に示す通りである．

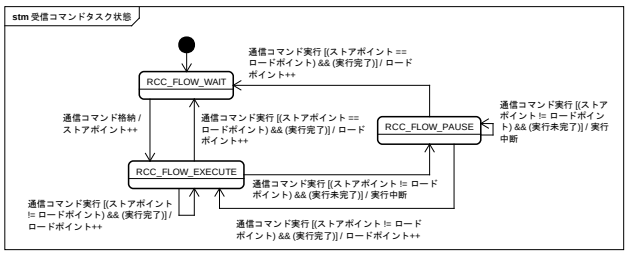

図 4 通信コマンド実行タスク状態遷移図

3.7 マルチユニット間イベントコマンドの実行タスクの実現

宮澤の先行研究より，マルチユニット時の協調動作においてユニット間でやり取りするコマンドは PC とは異なり，イベントコマンドとして処理している．受信データ解析処

理までは通信コマンドと共通であるが，実行は別の形態をとっている．

本研究では，イベントコマンド実行タスクにも実行中断の判定機能を用いてデータ取得処理に時間がかかるイベントコマンドは中断し，次の処理に CPU を明け渡すようにした．さらに，中断したコマンドを退避させる方法として，FIFO のリングバッファを用いた．これは複数のイベントコマンドが来た場合に対応するためである．PC コマンドの実行タスクでは中断した通信コマンドで実行が止まってしまう．複数のユニットからコマンドが送られてくる場合があるイベントコマンドでは実行を中断した時，後に待機しているコマンドの実行されない可能性がある．そのため，イベントコマンドでデータ取得待ちが発生した場合はイベントコマンドの各種データをリングバッファに格納し，後に待機しているコマンドを先に実行する．そして，リングバッファに格納されたイベントコマンドは後にデータ取得が完了したか否かを確認し，完了した場合は実行を再開する．

4. 実験・評価

単一ユニット動作時やマルチユニット動作時，PC との連携動作時でユニットの構成を変え，制御フロータスクの応答時間に対する通信コマンドの影響について計測する．

次に，応答時間改善システムの追加前と後で制御フロータスクの応答時間の評価を行う．そして，実験結果の比較を行い，制御フローの応答時間が”5ms 以内”となることを実現できたか否かを評価する．

4.1 使用器材

本研究の評価実験には 2 台の NX2000 を用いた．これは，マルチユニット動作での評価実験を行うためである．2 台の NX2000 を IEEE1394 ケーブルで接続することでマルチユニット状態を構築する．PC とタッチパネルは Ethernet ケーブルを用い，ハブを介して接続する．基本的は実験機材の構成は以下の図 5 の通りである．

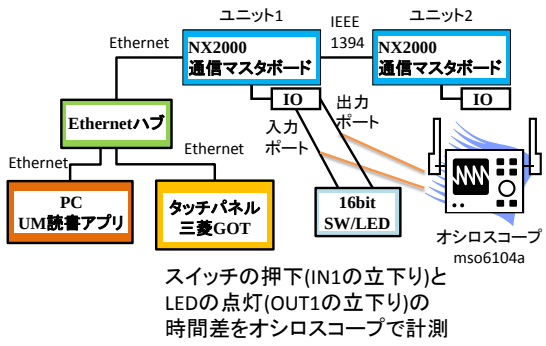

図 5 実験環境

4.2 実験・評価方法

IO ボードを使用し，応答時間評価用の制御フローを作成した．具体的な制御フローは図 6 に示す通りである．

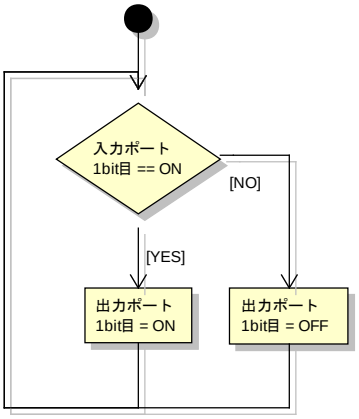

図 6 応答時間評価用制御フロー

この評価用制御フローは，IO ボードの入力ポート 1bit 目に入力があれば，出力ポート 1bit 目に出力する処理である．入力信号を与えてから出力信号が変化するまでの時間を制御フロータスクの応答時間とし，オシロスコープにより計測する．そして，応答性改善システム追加前と後の NX2000 に対して，以下の動作状況でユニットの構成を変え計測を行う．さらに，同時に起動するタスク数を加減し，制御フロータスクの応答時間変化の評価を行う．

① 単一ユニット動作
② 単一ユニットが PC からコマンドを受信しての動作
③ マルチユニット動作にてマスタがサブからデータ取得要求（UM 対象）を受ける動作
④ マルチユニット動作にてマスタがサブからデータ取得要求（タッチパネル対象）を受ける動作
⑤ マルチユニット動作にてマスタがサブからデータ取得要求（UM 対象）と PC からコマンドを受ける動作
⑥ マルチユニット動作にてマスタがサブからデータ取得要求（タッチパネル対象）と PC からコマンドを受ける動作

ただし，NX2000 の IO ボードは 1ms ごとにポーリングを行う設計になっており，さらに信号が同じレベルを連続で 2 回読込むまで，入力を認めない．そのため，信号の入力から最小 1ms，最大 2ms，平均 1.5ms の遅延があることも考慮して計測する．計測は，入力 1000 回に対する最小時間，最大時間を計測する．最小時間，最大時間を計測するのは，信号入力のタイミングによって IO ボードの入力値反映タイミングが変化することがある．さらに，通信コマンドの受信タイミングによっても応答時間が変化すること，そして通信コマンドの送信タイミングと受信側ユニットの処理タイミングを指定して固定することができないためである．以上の理由から変化する応答時間を取得できるよう最小時間，最大時間を計測する．

Vol.2015-SLDM-170 No.7
Vol.2015-EMB-36 No.7
2015/3/6

4.3 実験・評価結果

応答性改善システム追加前の NX2000 で実験を行った結果，以下の表 1 の結果が得られた．表中の実験番号は 4.2 節で記述した動作環境と対応している．

表 1 応答性改善システム追加前の実験結果

実験番号	各タスク数における最大応答時間と最小応答時間(ms)							
	タスク数 1		タスク数 10		タスク数 30		タスク数 50	
	min	max	min	max	min	max	min	max
①	1.19	2.23	1.47	3.15	2.34	5.34	3.16	7.88
②	1.15	2.26	1.46	3.11	2.32	5.50	3.22	8.04
③	1.17	2.29	1.45	3.23	2.30	5.60	3.16	7.88
④	1.20	34.80	34.00	100.80	28.90	100.80	34.00	100.00
⑤	1.19	2.29	1.46	3.13	2.46	5.44	3.40	7.84
⑥	1.24	34.80	34.80	100.40	34.80	99.60	35.20	100.80

次に，応答性改善システム追加後の NX2000 で実験を行った結果，以下の表 2 の結果が得られた．

表 2 応答性改善システム追加後の実験結果

実験番号	各タスク数における最大応答時間と最小応答時間(ms)							
	タスク数 1		タスク数 10		タスク数 30		タスク数 50	
	min	max	min	max	min	max	min	max
①	1.16	2.25	1.45	3.11	2.26	5.44	3.14	7.92
②	1.17	2.28	1.50	3.18	2.32	5.06	3.14	7.96
③	1.16	2.26	1.50	3.18	2.28	5.56	3.14	7.96
④	1.19	2.26	1.52	3.20	2.28	5.58	3.20	8.60
⑤	1.16	2.24	1.74	3.24	2.26	5.58	3.18	7.54
⑥	1.19	2.25	1.52	3.24	2.32	5.56	3.20	8.12

4.4 考察

表 1 より，NX2000 単体で応答時間を計測した場合と比較し，別ユニットから調査ユニットに対してタッチパネルへのデータ取得要求がある実験環境 4, 6 の応答時間が大幅に遅延する瞬間があることがわかる．そして，その他の実験環境は NX2000 単体と大きな差は無いことが確認できた．以上の結果から，システム追加前では別ユニットがタッチパネルのデータ取得要求通信によって NX2000 の応答性が損なわれることが判明した．

次に表 2 より，各実験環境で計測を行い追加前の結果と比較したところ，システム追加前で大きく応答時間が遅延していた実験環境 4, 6 に対して，応答時間の改善がなされたことが確認できた．さらに，システム追加前と後で各タスク数の最大応答時間を比較した結果，最大応答時間を 92.68ms 短くすることができた．

表 2 の結果から，タスク数 1，10 の場合では目標であった制御フローの応答時間 5ms 以内を達成することができた．しかし，タスク数が増加するに従って最大応答時間が 5ms を超えてしまう結果となった．

以上のことから，本研究のシステムを追加したことでイベントコマンドによって制御フローの応答時間が遅延する問題を改善することができた．しかし，タスク数の増加応じて応答時間は増大していき，タスク数 30 で目標の 5ms を超えてしまった．このことから，ユニット間通信(IEEE1394)やネットワーク通信(Ethernet)に関する処理だけでなく，他の制御処理や NX2000 ファームウェアに対しても応答時間改善の手法を実現する必要があると考える．

5. まとめと今後の課題

本研究では，処理時間の長いイベントコマンドが実行される場合でも制御フローの応答時間が 5ms 以内を実現するため，制御フローのみならず通信コマンドに対しても応答時間改善システムを NX2000 に適用した．その結果，処理時間の長いイベントコマンドが実行される場合でも制御フローのタスク数が 1，10 の場合，応答時間を 5ms 以内にすることができた．

今後の課題として，さらなる応答時間改善の実現と実行権剥奪の判定基準を再定義する必要があると考える．

実験の結果，タスク数が増加した場合に目標の応答時間 5ms 以内を満たせなくなっていった．そのため，さらなる応答時間改善の手法を検討, 実現する必要があると考える．

次に本研究では，データ取得対象がタッチパネルもしくは他ユニットである場合に限定して実行権剥奪を行った．つまり，データ取得対象デバイスを判定基準としたのである．しかし，この方法では今後ネットワークを介して接続されるデバイスの種類が増加する場合，デバイスの増加毎に処理を追加する必要が出てしまう．デバイスの増加毎に実行権剥奪処理を追加していては，ソフトウェアの拡張性の面から不適切である．さらに，条件分岐増加による処理量増加が発生すると思われる．そのため，ネットワークアクセスを伴うデータ取得処理を行う場合に実行権剥奪を行うなど，判定基準を再定義しソフトウェアを改修する必要があると思われる．

謝辞　本研究に際して、enPit-Emb の取り組みを通して様々なご指導ご協力を頂きました皆様に，謹んで感謝の意を表します．

参考文献

[1] 畠山　祥維，独立型汎用 FA コントローラの設計開発，静岡大学情報学部 修士論文（2012）．
[2] 宮澤　彰人，分散協調型 FA コントローラの開発，静岡大学情報学部 修士論文(2013)