平成 28 年度（2016 年度） 学位論文（修士）

共有メモリを用いた
分散型アーキテクチャから成る
超小型衛星に関する研究

Study on Realization of Microsatellite
Consisting of Distributed Architecture Using Shared Memory

首都大学東京大学院
システムデザイン研究科　システムデザイン専攻
航空宇宙システム工学域　博士前期課程
学修番号　15891533
氏名　堀田 周平

指導教員　佐原 宏典 教授

平成 29 年（2017 年）2 月 27 日

摘要

現在，民間企業が積極的に衛星業界に参入しており，参入障壁の低い超小型衛星によるビジネスは民間企業によって日々勢いを増しつつある．さらに超小型衛星用ロケットの開発も進み日本の超小型衛星を取り巻く環境は年々充実し，活気を帯びてきている．しかし，衛星開発において超小型衛星は短期開発・低価格を実現したように思われるが，実際にはそこまでの低価格化には至っていない．そこで当研究室では低価格化を実現するための新しいアーキテクチャの研究を行っている．

本研究では低価格化を実現する新しいアーキテクチャである「共有メモリを用いた分散型アーキテクチャ」の提案を行い，従来のアーキテクチャと待ち行列理論を用いて処理性能を比較した．その後，より実際の衛星の条件に近いマイコンを用いた簡単なシステムを構築し，同様に処理性能の比較を行った．その結果から，共有メモリを用いた分散型アーキテクチャは処理能力の面から従来のアーキテクチャと比べて優れているといえることがわかった．しかし，通信量が多いことによるバスラインの利用率による問題が浮き彫りとなった．最後に共有メモリを用いた分散型アーキテクチャの標準化に向けた設計を行い，共有メモリを用いた分散型アーキテクチャの実装方法を提案した．

目次

図目次

表目次

第 1 章　序論

1.1　研究背景

近年超小型衛星の打ち上げ数は年々増加している．図 1.1 は 2003 年以降の世界の超小型・小型衛星の打ち上げ数の推移を示しており，特に 2012 年以降は 1～10kg 級衛星の打ち上げ数が急激に伸びている．

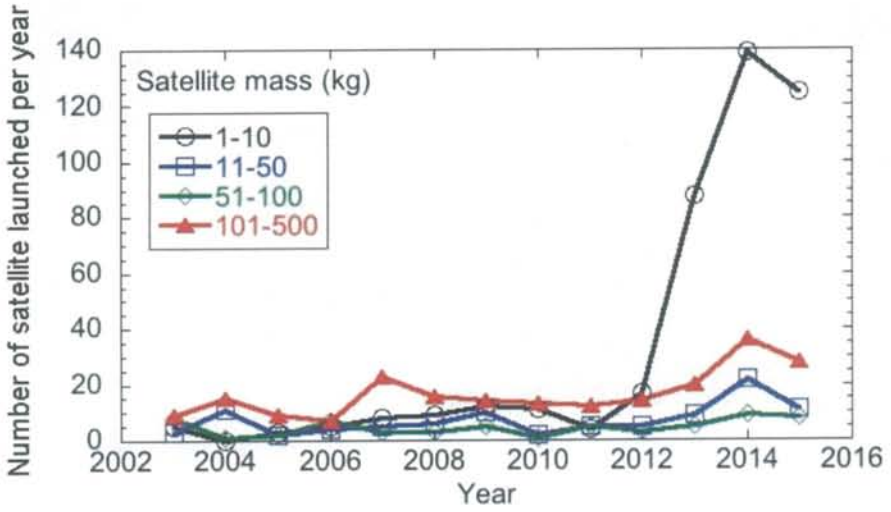


図 1.1　世界の超小型・小型衛星の打ち上げ数推移（九州工業大学調べ）[1)]

1～10kg 級の衛星開発は学生が衛星プロジェクトを進める中でシステムエンジニアリングを身に着けることができる良い教材として広がる一方，ここ数年では 50kg 級の衛星の開発も進み，サイエンスをミッションに掲げる衛星が増えている．これは超小型衛星におけるバス技術が発展した結果である．また，打ち上げ数のみならず超小型衛星業界へ新規参入する大学や企業なども増え，この分野は着実に成長をしている．

例えば，東京大学・東京工業大学での超小型衛星の開発の経験をもとに，ベンチャーとして独立した株式会社アクセルスペースは 2015 年にシリーズ A 投資ラウンドにおいて総額約 19 億円を調達し，50 機の超小型地球観測衛星によるコンステレーション計画を発表した [2)]．九州工業大学では超小型衛星試験センターを 2010 年に開設し，放射線試験以外の超小型衛星の試験がすべて集約されており，超小型衛星の試験をまとめてすることができ，超小型衛星の短期開発に貢献している [3)]．

また，超小型衛星の輸送機においても，宇宙航空研究開発機構（JAXA）は観測用小型ロケット「SS-520」を改良したロケットで 2017 年 1 月 15 日に超小型衛星を載せて打ち上げを行った．打ち上げは失敗に終わったものの小型ロケットの可能性を示した [4)]．超小型衛星の打ち上げ手段として現在は H-IIA ロケット・イプシロンロケットの相乗り衛星として打ち上げるか，もしくはロシアの ISC コスモトラス社のドニエプル・ロケットによって打ち上げるかの選択しかない状況であり，軌道も打ち上げ時期にも制限があるため，超小型衛星の開発における障害となっている．しかし，超小型衛星用のロケットが整備されることにより，それらの制限が緩和されることが予想される．さらに，ベンチャー企業であるインターステラテクノロジ社が超小型衛星用ロケットの開発を進めており [5)]，超小型衛星の輸送機側の環境も整ってきた．

以上のように日本の超小型衛星を取り巻く環境は年々充実しつつあり，民間企業の宇宙開発の台頭によ

り，さらに活気を帯びてきている．

しかし，衛星開発において超小型衛星では短期開発・低価格化を実現したように思われるが，実際にはそこまでの低価格化には貢献していない．図 1.2 は衛星の開発費を質量比で割った図である．従来，衛星質量 1kg 当たり数 1000 万円の開発費がかかっていたが，民生品利用・衛星基幹部の共通化により，1kg 当たり 500～1000 万円までコストを削減することができた，しかし仕様変更の余儀がない現状によりこれ以上の低価格化が難しくなっている．そのため，産業として自立可能といわれる 100~200 万円/kg までコスト削減を達成してはいない．

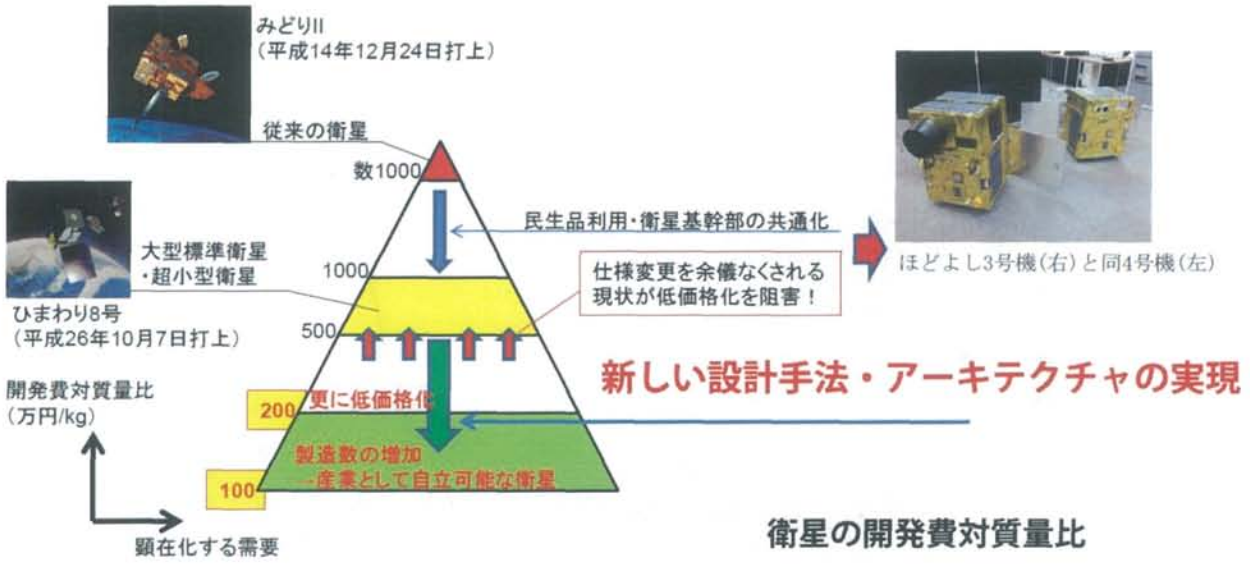


図 1.2　衛星の開発費対質量比

そこで当研究室では衛星の低価格化を実現するための新しいアーキテクチャの研究を行っている．現在，超小型衛星に適用されているアーキテクチャは中央集中型アーキテクチャと統括システムをもつ分散型アーキテクチャである．分散型アーキテクチャではワシントン大学の Bryan Palmintier, Christopher Kitts, Pascal Stang, Michael Swartwout らが研究しているが[6]，分散型アーキテクチャの標準化には至っていない．そこで分散型アーキテクチャの標準化に成功すればサブシステムの追加・交換が容易となるため開発を独立したり，サブシステムを再利用したりするなどコスト削減に大きく貢献できる．また，低コストを実現できるアーキテクチャによって衛星としての性能はどのように変わるのか，どのような特徴があるのか評価し，そしてそのアーキテクチャをどのように実装するかを述べる必要がある．

1.2　研究目的

前節より，本研究では新しいアーキテクチャの実現について述べる．そこで，本研究の目的を以下のように設定する．

・低価格化を実現できるアーキテクチャを提案する．
・提案したアーキテクチャと従来のアーキテクチャとの処理能力を比較する．
・提案したアーキテクチャの標準化に向けた設計を行い，実装方法の提案を行う．

第2章　共有メモリを用いた分散型アーキテクチャ

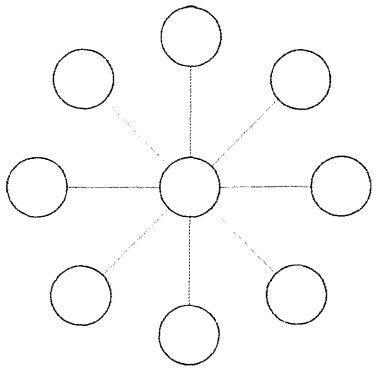
この章では最近打ち上げられた超小型衛星や現在開発中である超小型衛星のアーキテクチャを分類し，低価格化を実現できる新たなアーキテクチャを提案し，概要を説明する．

2.1　ネットワークトポロジー

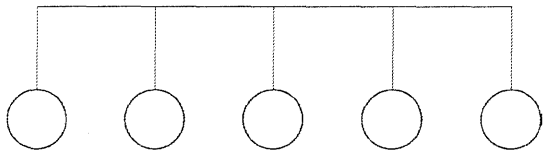
ネットワークトポロジーとはコンピュータネットワークのトポロジーのことで，ネットワーク上のノードとエッジの相関を抽象化して示したものである．ネットワークトポロジーは通信速度・通信プロトコル・ハーネス量に影響を及ぼす．通信速度・通信プロトコルが重要であることは明白であるが，一見軽視されるハーネスにおいても衛星内部ではあらゆる数の機器がそれぞれハーネスによってつながっており，ハーネスによる重量・体積は構造設計において無視できないほどである．

図 2.1 は衛星におけるネットワークトポロジーで代表的なスター型トポロジーとバス型トポロジーである．スター型トポロジーは 1 つのノードが複数のノードと個別に接続されており，ノード間の通信は中央のノードを介して行われる．バス型トポロジーはすべてのノードがバスラインと呼ばれる，同一の信号線に接続されているため，1 つのノードはバスラインに接続されているノードすべてに対して通信を行うことができる．しかし，複数のノードが同一のバスラインを用いるため，バスラインを協調して使用する必要がある．

また，ネットワークトポロジーには物理トポロジーと論理トポロジーが存在し，物理トポロジーは物理的に配線がどのように結ばれているかを表し，論理トポロジーは実際のデータの流れがどの経路を通っているかを表す．



スター型トポロジー



バス型トポロジー

図 2.1　主要なネットワークトポロジー

2.2　衛星アーキテクチャ

2.2.1　中央集中型アーキテクチャ

現在までに打ち上げられた超小型衛星において，最も代表的なアーキテクチャは中央集中型アーキテクチャである．中央集中型アーキテクチャはメイン OBC（On-Board Computer）がその他の機器に命令をし，得られた情報から全体を統括して処理するアーキテクチャである．ネットワークトポロジーはスター型で

構築される場合が多い.

図 2.2 は 2014 年 12 月に打ち上げられた，東京大学と JAXA が共同開発した超小型深宇宙探査機「PROCYON」のシステム構成図である.「PROCYON」では姿勢系機器やミッション機器が直接メイン OBC とやりとりをしている [7].

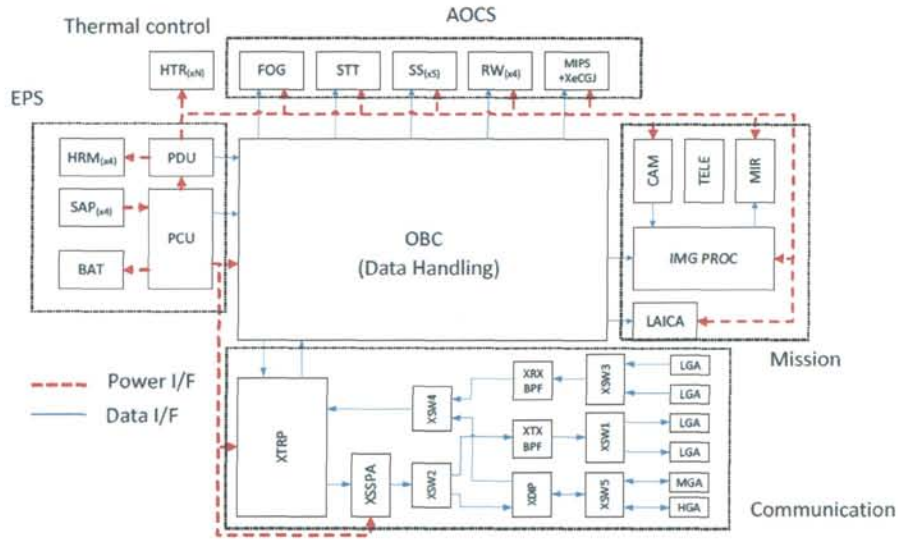


図 2.2 超小型深宇宙探査機「PROCYON」のシステム構成図 [8]

この様な中央集中型アーキテクチャでシステムを構築すると以下の長所がある.

・データフローが理解しやすい.
・全体の挙動を理解しやすい.
・コンポーネント数を最小限に留めることができる.

しかし，以下の短所が存在する.

・開発を分割したり，独立して並行に開発をしたりすることが難しい.
・メイン OBC に高性能の処理を要求する.

2.2.2 分散型アーキテクチャ

分散型アーキテクチャは中央集中型アーキテクチャでは集中していた機能を分散させたアーキテクチャである.

分散型アーキテクチャではサブシステム毎に OBC を所持し，サブシステム毎の開発を独立して行うことができる. また，処理を分散させることで中央集中型アーキテクチャと比べて高性能な OBC を必要としない. 全体の処理を行う OBC（統括システム）に各サブシステムが接続されているシステムである.

図 2.3 は日本大学宮崎・山崎研究室と日本アマチュア衛星通信協会が開発を行っている 1U サイズの通信機（リニア・トランスポンダと π/4 シフト QPSK 送信機）技術実証衛星「NEXUS」のシステム構成図である. 中央に統括システムが存在し, 他サブシステムと物理トポロジーでスター型接続となっているため, 一見すると中央集中型アーキテクチャと似ている. しかし，サブシステムは OBC を持っており，機能は分散しているため，スター型の分散型アーキテクチャといえる.

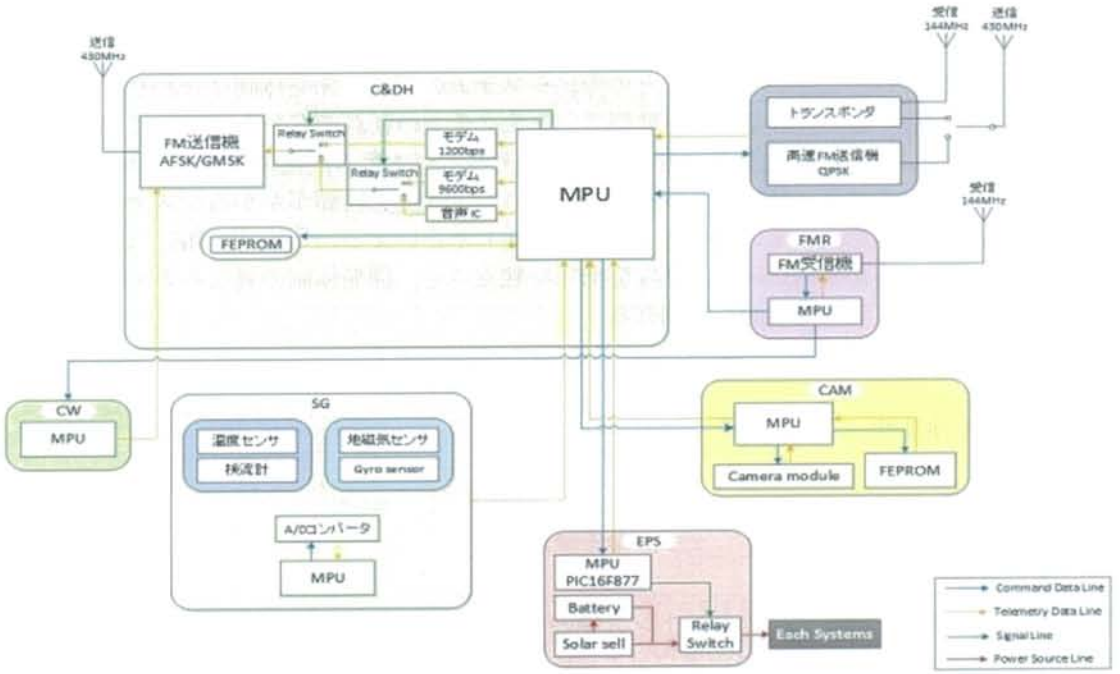


図 2.3　超小型衛星「NEXUS」のシステム構成図 [9)]

また，図 2.4 は東京工業大学松永研究室が開発し，2014 年 11 月に打ち上げられた 50kg 級超小型衛星「TSUBAME」のシステム構成図である．「TSUBAME」では X 線偏光観測装置を用いたガンマ線バーストの観測をミッションの 1 つに据えている．

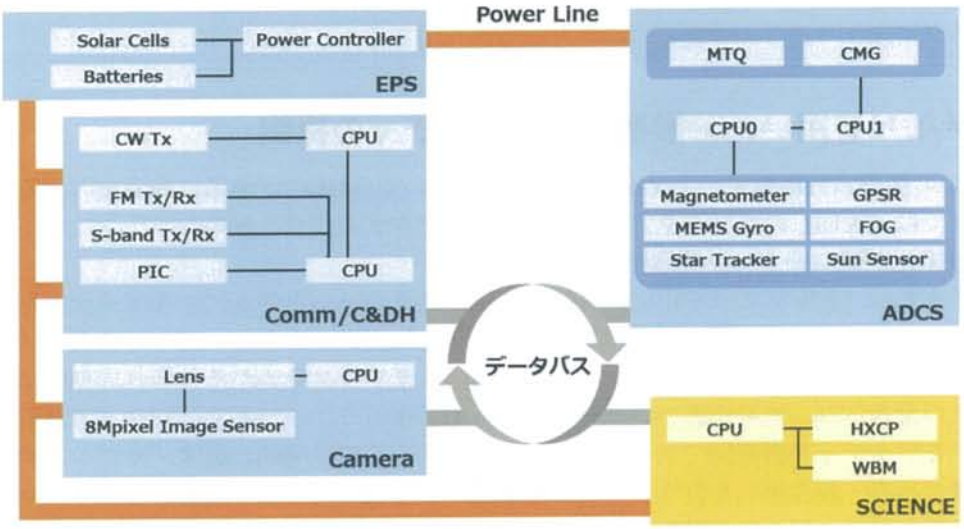


図 2.4　超小型衛星「TSUBAME」のシステム構成図 [10)]

「TSUBAME」では姿勢系（ADCS），科学系（SCIENCE），観測系（Camera），通信・C&DH 系と 4 つのサブシステムに分散させ，バス接続をしている構成となっている．そのため，バス型の分散型アーキテクチャといえる．「TSUBAME」では C&DH 系が統括システムとして，その他のサブシステムに命令を行っている．バス型トポロジーのため，通信のタイミングや通信を独占しないための処理などスター型と比べ

るとモジュール間の通信がより重要となる[11].

以上のように分散型アーキテクチャでは中央集中型アーキテクチャと比べ機能を分散することで，サブシステム毎の開発の分散を行うことができる．その際にシステムの規模・開発体制に合わせて分散のレベル・方法と柔軟に変更することができるのが分散型アーキテクチャの利点でもある．例えば「NEXUS」は1Uサイズの超小型衛星のため，機能を分散させすぎるとかえってシステムが複雑になり，コンポーネント数も増えてしまう．そのためスター型トポロジーでの接続にし，開発に時間がかかるシステムにOBCを持たせ，ほどよく機能の分散を図ったアーキテクチャを構築したといえる．「TSUBAME」では50kg級の衛星であるため，高精度の姿勢制御処理が必要な姿勢系を独立させ，開発体制の異なるカメラ・ミッション系を独立させたことがシステム構成図から伺える．

以下に分散型アーキテクチャの長所を並べる．

・再利用性
・設計時間の短縮と検証性
・柔軟さ

短所には以下があげられる．

・中央集中型アーキテクチャと比べてシステムが複雑になる．
・コンポーネント数が増える．

バス型の分散型アーキテクチャではワシントン大学のBryan Palmintier, Christopher Kitts, Pascal Stang, Michael Swartwoutらが研究しており，ハードウェアでの通信規格，通信プロトコルの標準化，さらに衛星内のコマンドやデータ処理を担うC&DHソフトウェアのインターフェイスさえ標準化してしまえば，究極的にはモジュールの追加・交換が容易に行えると述べている[12]．標準化によるメリットは大きく，コストの削減に大きく貢献する．国内の超小型衛星では現状，プロジェクト毎の作りこみ型の衛星が主流となっており，標準化を実現した超小型衛星は開発されていないため，標準化の恩恵を受ける大きな余地がある．

しかし，現状のアーキテクチャではハードウェアでの通信規格，通信プロトコルの標準化は容易に行うことができるが，C&DHを標準化させることは難しいと考える．なぜなら，C&DHを1つのOBCが担い，それぞれのサブシステムOBCに衛星内部でのコマンドやデータ要求を行う集権的なシステムである以上，コマンド仕様やサブシステムの仕様に左右されC&DH機能の一般化ができず，サブシステムが追加・変更した際にはデータ処理プログラムの変更が必要となるからである．

2.3 共有メモリを用いた分散型アーキテクチャの提案

2.2.2項において従来の分散型アーキテクチャにおける，1つのOBCが他系に命令を与える集権的なシステムの組み方ではC&DHのソフトウェア実装が一般化できず，作りこみのシステムになってしまい，C&DH機能の標準化が難しいという問題を述べた．

そこで当研究室では「共有メモリを用いた分散型アーキテクチャ」と呼ぶアーキテクチャを研究している．このアーキテクチャではバス型の分散型アーキテクチャに全系がアクセスできる共有のメモリを配置するものである．他の分散型アーキテクチャと異なる点は，サブシステムがそれぞれ周期的に共有メモリに自系のデータの書き込み，他系が書いたデータの読み出しを行うことである．図2.5は共有メモリを用いた分散型アーキテクチャの概要図である．共有メモリはあくまで受動的に動き，サブシステムの要求通りにメモリの書き込み・読み出しを行う．共有メモリはいわば掲示板のようなものであり，サブシステムは周期的に掲示板を読みに行って行動を決めるといえる．このアーキテクチャによって，それぞれのサブシステムは他系の内部構成を知らなくて済むため，インターフェイス設計では共有メモリとの通信規格・プロトコルを合わせればよい．物理トポロジーをバス型にすることで，ハーネス量・I/F量の軽減と通信規格の標準化を行うことができ，論理トポロジーをスター型とすることによって，サブシステムと共有メモリとの一対一対応を実現することができる．また，共有メモリを持つ系を共有系と呼ぶ．この共有系は独立させ，1つの系とすることもできるが，共有系での処理は比較的軽いものであるため，他のサブシステムに組み込んでモジュール数を削減することも可能である．このアーキテクチャでは統括システムは存在

しないため，以降は従来の分散型アーキテクチャを統括システムをもつ分散型アーキテクチャと呼ぶこととする．

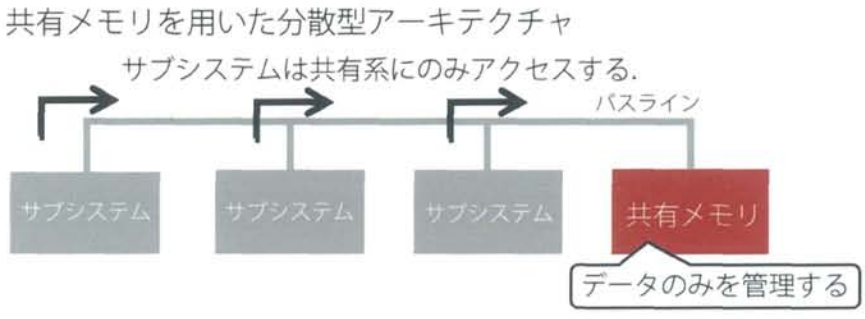


図 2.5　共有メモリを用いた分散型アーキテクチャ概要図

共有系は以下の３つの要素を管理する．

・コマンドフラグ

・衛星の現在のモード

・テレメトリ

また，共有系は地上からのコマンドを衛星内部のサブシステムへの内部コマンドに変換をする機能を実装させる．

図 2.6 はコマンド処理フローである．

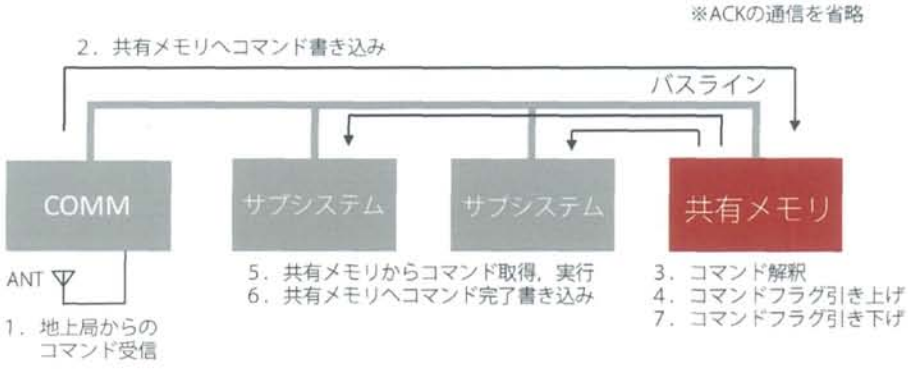


図 2.6　共有メモリを用いた分散型アーキテクチャにおけるコマンド処理フロー図

図 2.6 のようにコマンド処理においてもサブシステムと共有系との一対一の通信を行い，すべての通信処理においてサブシステム-サブシステム間は通信を行わない．

以上のように論理トポロジーをスター型にし，サブシステムと共有系との一対一の対応を徹底することと，サブシステムの周期的なアクセスにより，C&DH 機能の一般化を行うことができ，標準化が可能となると考える．C&DH 機能をソフトウェアとして実装してしまえば，より開発コストを削減できることはもちろん，アーキテクチャそのものの信頼性を上げることにつながる．

また，サブシステムと共有系との一対一対応にはまだ利点がある．サブシステムの仕様変更による影響も共有系の内部設計の仕様変更に留めることができ，それもフラグの内容の変更や追加など比較的簡単な仕様変更といえる．そしてこのアーキテクチャの利点が最も発揮されるのはシステム統合テスト時である．通常システムを統合する際には少しずつサブシステムを追加し，相互の電気的整合性，ソフトウェア的整合性をテストするが，サブシステムを追加する度に，そこまでに確認したサブシステム間の整合性もテス

トする必要が出てくる．つまり，サブシステム数を n とすると 2^n でシステム統合の作業が存在する．しかし，共有メモリを用いた分散型アーキテクチャの仕様を明確にし，最終的に標準化まで実現すれば，システム統合時にはサブシステムは共有メモリとの整合性と全体のバスライン上のソフトウェア整合性を確認すればよいため n の多項式程度の作業量で済む．

以上から，共有メモリを用いた分散型アーキテクチャによってもたらされる衛星開発のコスト削減は大きいといえる．しかし，コスト削減の面とは別に従来のアーキテクチャと比べて性能評価を行う必要がある．実用的なシステムであれば，標準化のためのアーキテクチャ詳細設計についての議論に進むことができる．

第 3 章　処理能力によるアーキテクチャ比較

前章では共有メモリを用いた分散型アーキテクチャの提案を行った．そこで共有メモリを用いた分散型アーキテクチャが現在主流のアーキテクチャである統括システムをもつ分散型アーキテクチャと比べて処理能力が実用的であるといえるのか，待ち行列理論を用いて考察する．その後，複数の CPU を通信させる実験によって，より実際に近い条件で処理能力を比較する．

3.1　待ち行列理論によるアーキテクチャ比較

3.1.1　待ち行列理論

待ち行列理論は電話網の設計のために，デンマークの電話会社設計技師であるアーランが行った研究に始まる，確率的な現象である待ち行列を数理的に調べる手段である．待ち行列の特徴を抽象化して表現するにはモデル化が必要となる [13) 14) 15) 16)]．

待ち行列では待ち行列に関する用語を以下の図 3.1 に示すように定義している．

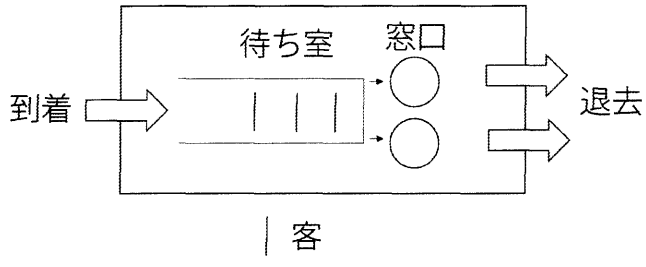


図 3.1　待ち行列基本モデルと用語

待ち行列理論では客の到着を到着過程でモデル化する．サービス時間は確率変数とみなす．待ち行列モデルは以下の要素から特徴づけられる．

- 客の到着過程
- サービス時間分布
- 窓口の数
- 待ち室容量
- 呼源数（到着する可能性のある顧客数）
- サービス規律（サービスの順番に関する規則）

待ち行列理論を用いることでシステムに具体的な処理時間や通信速度を考えずにアーキテクチャ同士の比較をすることができる．

今回比較するモデルではサブシステムからデータを集め，それをもとに最後のサブシステムが実行するというモデルを考える．

3.1.2　統括システムをもつ分散型アーキテクチャ

統括システムをもつ分散型アーキテクチャでは，統括システムがそれぞれの系に命令しデータを集めるモデルを考える．また，それぞれのサブシステムのサービス率はμ[1/s]と仮定する．

システムに入ってくるタスクを待ち行列理論での客とし，サブシステムでの処理をサービスとする．以上の条件でモデル化を行うと待ち行列システムは図 3.2 で表され，以下のようなモデルとなる．

1. 客はポアソン過程に従い，統括システムに到着する．
2. 統括システムから出た客はサブシステムへ入り，サブシステムを出た客は次のサブシステムへ向かう．
3. 通信時間，系内での処理時間を含めサービス時間とする．

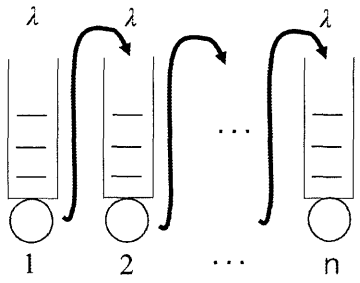


図 3.2　統括システムをもつ分散型アーキテクチャ

サブシステムから次のサブシステムへの出力は，バークの定理により，到着率λのポアソン過程になり，これが次にサブシステムへの入力になる．このことからこの待ち行列システムは M/M/1 モデルが直列に並んだものとしてモデル化できる．

待ち行列システムに流れ込む仕事量（呼量）を$a = \lambda/\mu$ とする．

M/M/1 モデルの定常分布の要素が有限個（ j 個）の場合には

$$\pi_j = (1-\rho)\rho^j,\ j = 0, 1, \ldots$$

であるから，サブシステムが n 個存在する場合には

それぞれのサブシステムに対応する待ち人数をL_1，L_2，…とすれば，$\rho = a = \lambda/\mu < 1$のとき，

$$P(L_i = k) = (1-a)\,a^k \qquad (i = 1, 2, \ldots, n) \tag{3.1}$$

であり，それぞれのサブシステム客数の期待値は

$$E[L_i] = \frac{a}{1-a} \tag{3.2}$$

期待値の公式から期待値の和は和の期待値であるため，

$$E[L] = \sum_{i=1}^{n} E[L_i] = \frac{na}{1-a} = \frac{n\lambda}{\mu-\lambda} \tag{3.3}$$

となる．サブシステム数が 1～10 の場合のグラフを図 3.3 に示す．

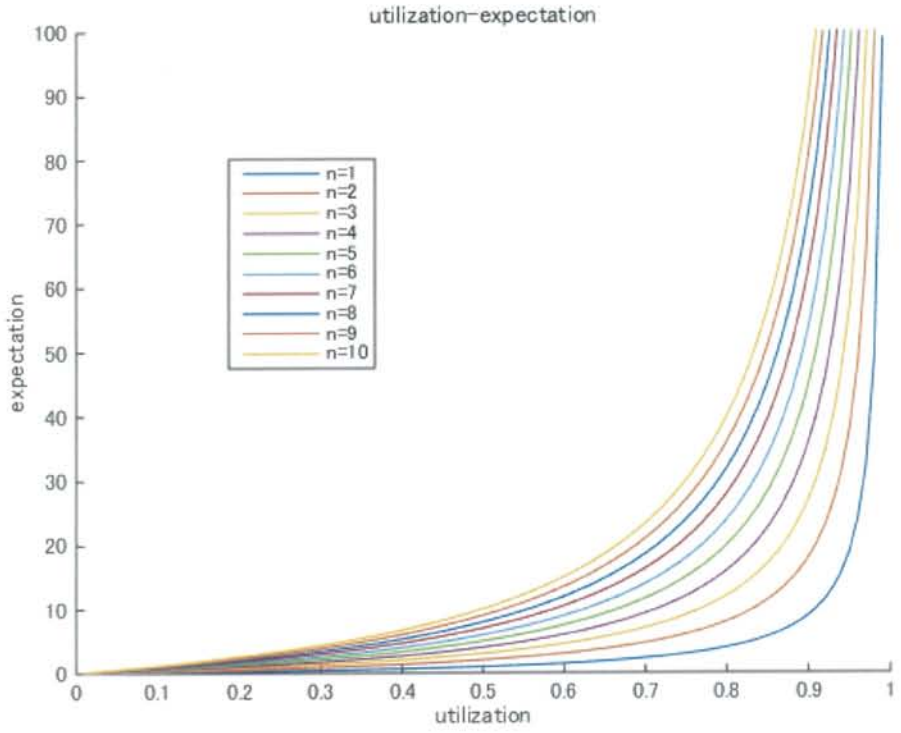


図 3.3　統括システムをもつ分散型アーキテクチャにおける利用率と期待値のグラフ

実際には設計者がプログラムしたタスクが順次発生するため，システムにタスクがポアソン過程でやってくることはないが,以上のように待ち行列理論を用いることによって,受付とそのネットワークにより，処理構造そのものを評価することができる．図 3.3 ではサブシステム数が増えるほどタスクの指数関数爆発が早く起きるということが示された.

3. 1. 3　共有メモリを用いた分散型アーキテクチャ

共有メモリを用いた分散型アーキテクチャではそれぞれの系がデータを共有メモリに書き込み，最後の系がデータを読み込み実行するモデルを考える．そのため，実行するサブシステムの M/M/1 モデルと考えることができる．各サブシステムの共有メモリアクセス周期を$1/\mu'$とする，そのため最後の系は各サブシステムのデータを見て実行するため，サービス率はμ'とする．ここではほかサブシステムは周期的に書き込み読み込みを行っており，待ち行列のように確率的な要素がなく，待ちが発生しないものとする．そのため，最後の系のみの待ち行列モデルとなる．以上の条件でモデル化を行うと待ち行列システムは以下のようになる.

1. 客はポアソン過程に従い共有メモリへ到着する.
2. それぞれのサブシステムのデータの書き込みを周期$1/\mu'$とする.

図 3.4　共有メモリを用いた分散型アーキテクチャ

この待ち行列モデルでは窓口がただ 1 つの単純な M/M/1 モデルと考えることができ，サブシステム数によらない結果が出てくる．しかし，サブシステムの書き込み読み込み周期によって待ちが発生してしまう．そこで書き込み読み込み周期を変えられるように$\mu' = \mu/r$とする．

$$E[L] = \frac{a}{1-a} = \frac{\lambda}{\mu' - \lambda} \tag{3.4}$$

上記の式から$\mu' = \mu/r$において r を 1~1.5 で変化させた結果，図 3.5 のグラフを得た．

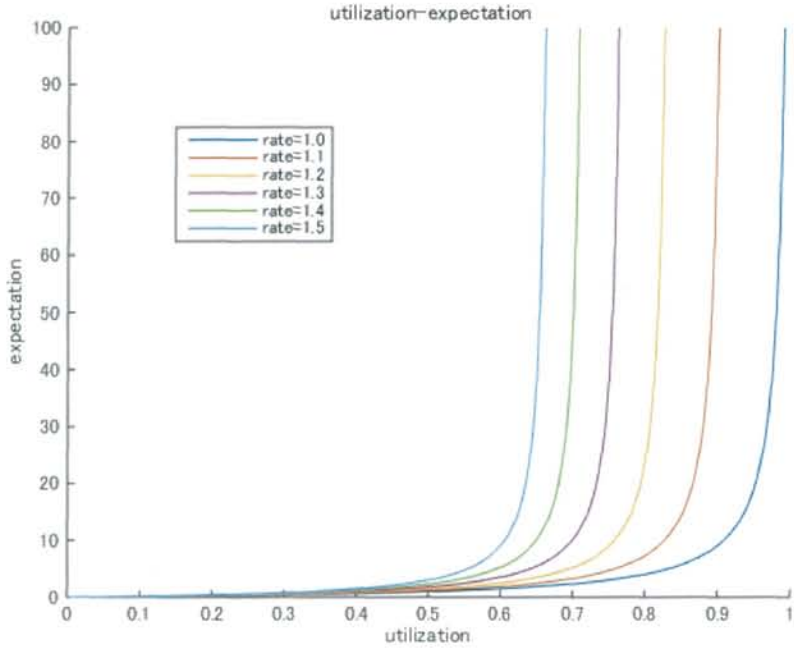


図 3.5　共有メモリを用いた分散型アーキテクチャにおける利用率と期待値のグラフ

3.1.4 考察

上記の2つのアーキテクチャの待ち行列モデルから得られた結果から1つのグラフに重ねると図 3.6 のようになる.

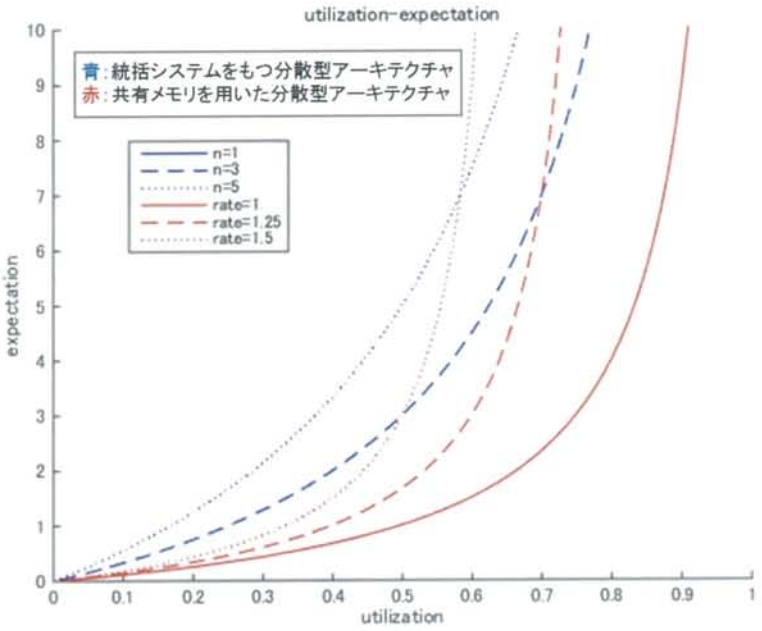


図 3.6　利用率と期待値のグラフ

共有メモリを用いた分散型アーキテクチャでは書き込み読み込み周期が $r=1$ の場合，すなわち，統括システムにおけるサブシステムのサービス周期と同じ場合には一般的に統括システムをもつ分散型アーキテクチャのサブシステム数は3〜5ほどであるため,共有メモリを用いた分散型アーキテクチャの方が優れているといえる．しかし，$r>1.5$ 程度の周期でサブシステムが共有メモリへアクセスするとなると利用率 $\rho = 2/3$ で無限大に発散してしまう．利用率が 1 まで達しないのは統括システムでの客の到着過程と同条件にして，サービス率だけ変更しているためである．統括システムをもつ分散型アーキテクチャと比較して，$r<1.25$ ほどで使用すれば同様の処理性能を発揮できるといえる．

3.2 アーキテクチャ比較実験

3.2.1 概要

この節では統括システムをもつ分散型アーキテクチャと共有メモリを用いた分散型アーキテクチャにおいて複数の CPU と通信を行うことによって，CPU 処理時間・通信時間が存在するより実際に近い条件で処理能力の比較実験を行う．

実験には 3 つのマイコンを用意し，それぞれのマイコンをサブシステムとして姿勢決定系を ADS（Attitude Determination System），姿勢制御系を ACS（Attitude Control System）と名付け，統括システムにはC&DH,共有メモリにはMEMORY と名付ける.2つのサブシステムはCANによってバス接続をさせる．アーキテクチャモデルそれぞれにおいて，ADS から疑似センサデータを集め，ACS に疑似コマンドを行わせるモデルを考える．処理時間は PC に出力されたターミナルソフトのタイムスタンプから取得し，処理開始の出力と処理終了の出力をもとに処理時間の算出を行う．

3.2.2 実験装置

サブシステム間通信にはCAN通信を用いる[17]．CANとは，「Controller Area Network」の略で，BOSCH社が開発した車載ネットワークのことである．特徴としては，長距離の伝送が可能なこと，通信レートが高いこと，差動通信路のためコモンモードノイズに強い，エラー検出機能が充実していてリカバリが容易などが挙げられる．信頼性の高さや優れた故障検出機能により，オートメーション機器の制御などにも利用されている．

表 3.1 はCAN通信での設定項目である．

表 3.1　CAN通信諸元

項目	諸元
通信速度[kbps]	125
ノード数	3

次に使用した電子部品の諸元を表 3.2 に示す．

表 3.2　電子部品諸元

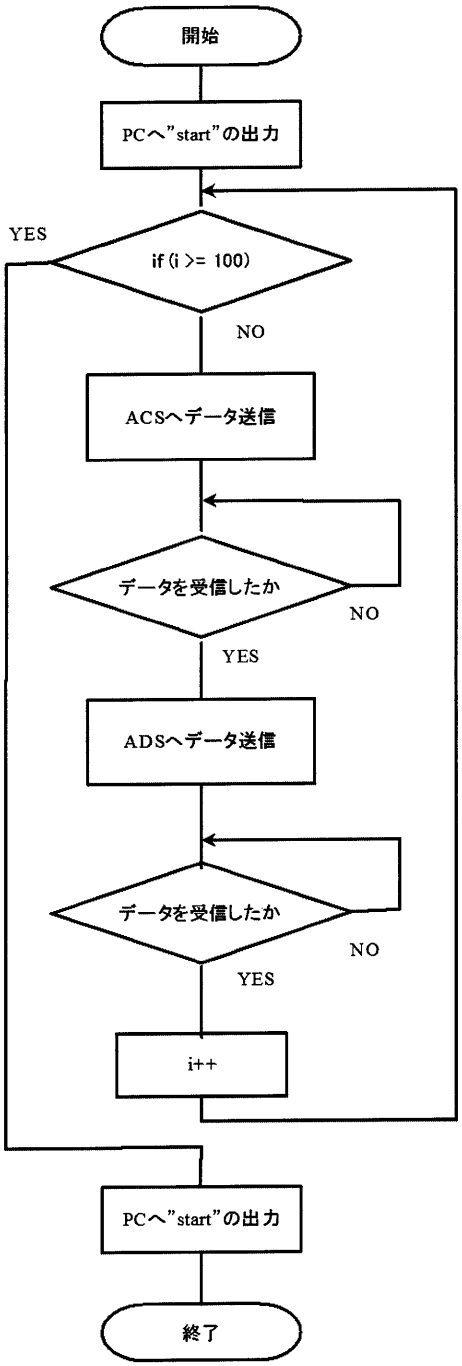
項目	諸元
CPU	SH7145F
CANコントローラ	MCP2515
CANトランシーバ	SN65HVD232D
デバッガ	E10A
直流安定化電源	LXO18-2B

試験環境を図 3.7 に示す．

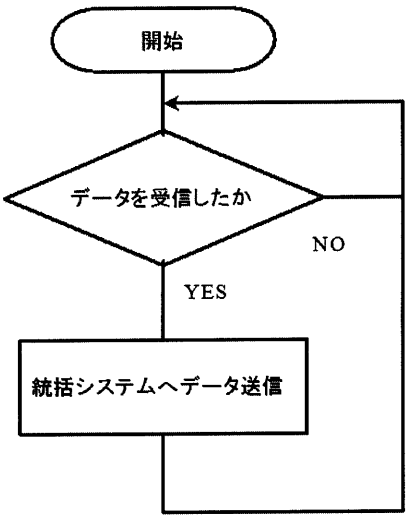
図 3.7　試験環境概要

3.2.3 統括システムをもつ分散型アーキテクチャ実験

統括システムをもつ分散型 アーキテクチャでの各サブシステムでの処理のフローチャートを図 3.8 に示す．



統括システムのフローチャート



サブシステムのフローチャート

図 3.8　統括システムをもつ分散型アーキテクチャでの処理フローチャート

実験結果を表 3.3 に示す.

表 3.3　統括システムをもつ分散型アーキテクチャの実験結果

平均値[s]	標準偏差[s]
13.015	0.35

3.2.4 共有メモリを用いた分散型アーキテクチャ実験

共有メモリを用いた分散型アーキテクチャの実験ではサブシステムの共有メモリアクセス周期を 50ms と 20ms 周期とで変化させて実験を行った．共有メモリを用いた分散型アーキテクチャでの各サブシステムの処理のフローチャートを図 3.9 に示す．

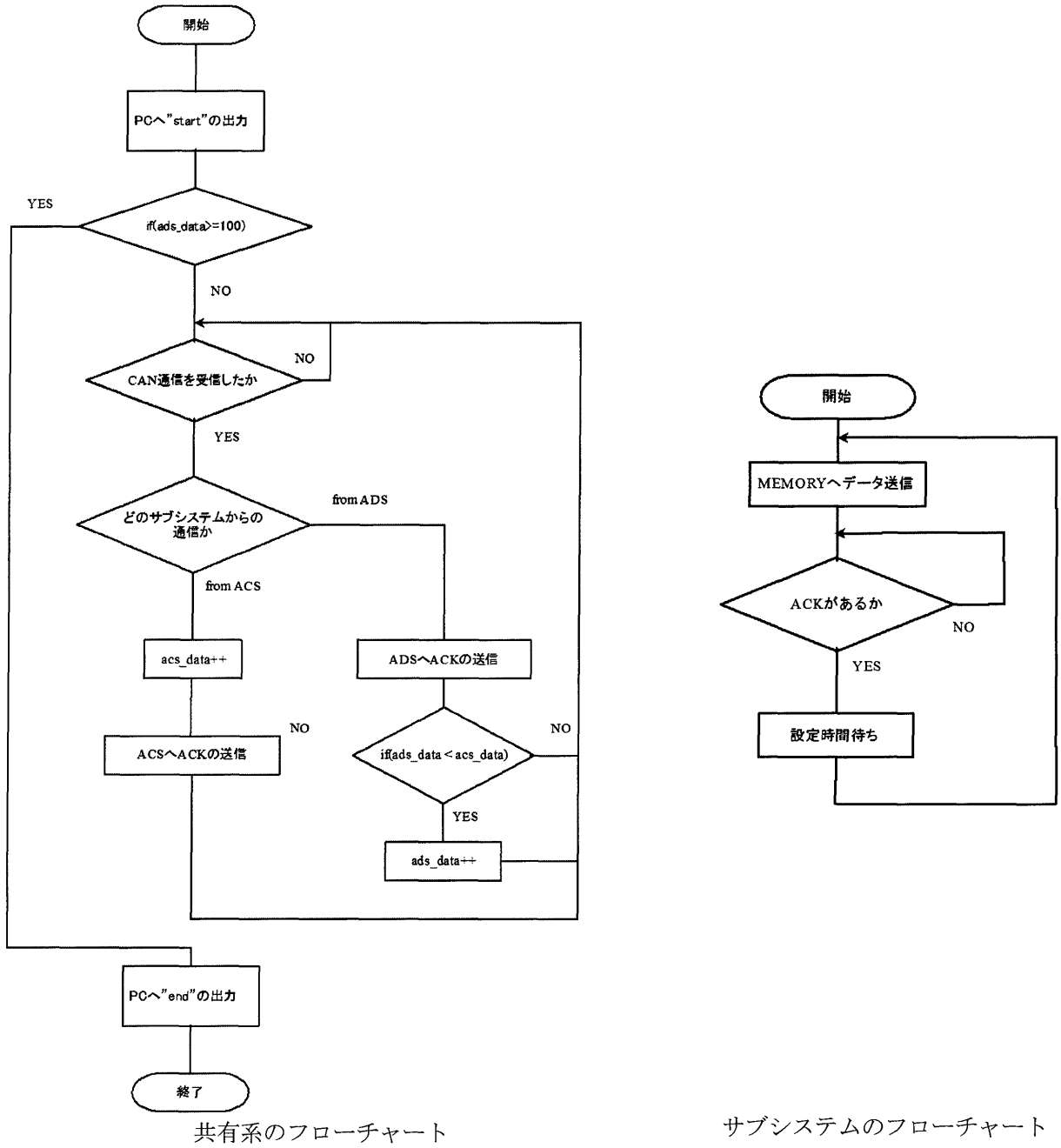


図 3.9　共有メモリを用いた分散型アーキテクチャのフローチャート

実験結果を表 3.4 に示す．実験を行った際，通信途中で CAN コントローラのモードが変更し，通信不能となることが多々あった．通信がうまくいった際はうまくバグに陥らなかったものと考えられ，そのデータは特別サブシステム間の処理が早くなったものではない，そのため，通信が最後の処理まで完了した

際のデータのみを抽出している．

表 3.4　共有メモリを用いた分散型アーキテクチャの実験結果

共有メモリアクセス周期[ms]	平均値[s]	標準偏差[s]
50	15.011	0.73
20	11.914	0.54

3.3　実験考察

統括システムをもつ分散型アーキテクチャと共有メモリを用いた分散型アーキテクチャでは共有メモリへのアクセス周期を 20ms としたときにより目的を達成することが早いという結果となった．

統括システムをもつ分散型アーキテクチャでの平均値からサービス時間が 0.064s となり，共有メモリを用いたバス型アーキテクチャはサービス時間と比べて以下の表 3.5 のようになった．

表 3.5　共有メモリアクセス周期とサービス時間比

共有メモリアクセス周期[ms]	50	20
サービス時間比 r	7.81	3.13

待ち行列モデルにおいて $r < 1.25$ ほどで統括システムを用いた分散型アーキテクチャと同程度の処理性能が発揮できると示した．しかし，実験では $r = 3.125$ 程度で同等以上の処理性能を示すことができた．これは待ち行列モデルでは考慮していない，CAN 通信での再送処理や，送信を待っているなどの処理による時間がかかっているものと考えられる．また，待ち行列モデルでは OBC での処理時間と通信処理時間を含めてサービス時間としたのに対して，実験では OBC のタイマー機能による待ち時間をサービス時間としたため，実際にはタイマーによる待ち時間と CPU の処理時間，通信における処理時間が存在していたと考えられる．

また，アクセス周期を 10ms と設定すると通信が成功せず途中で止まってしまう結果となった．以下に 2 バイトのデータを送信する際の CAN の回線利用率とサブシステム数を表 3.6 に示す．

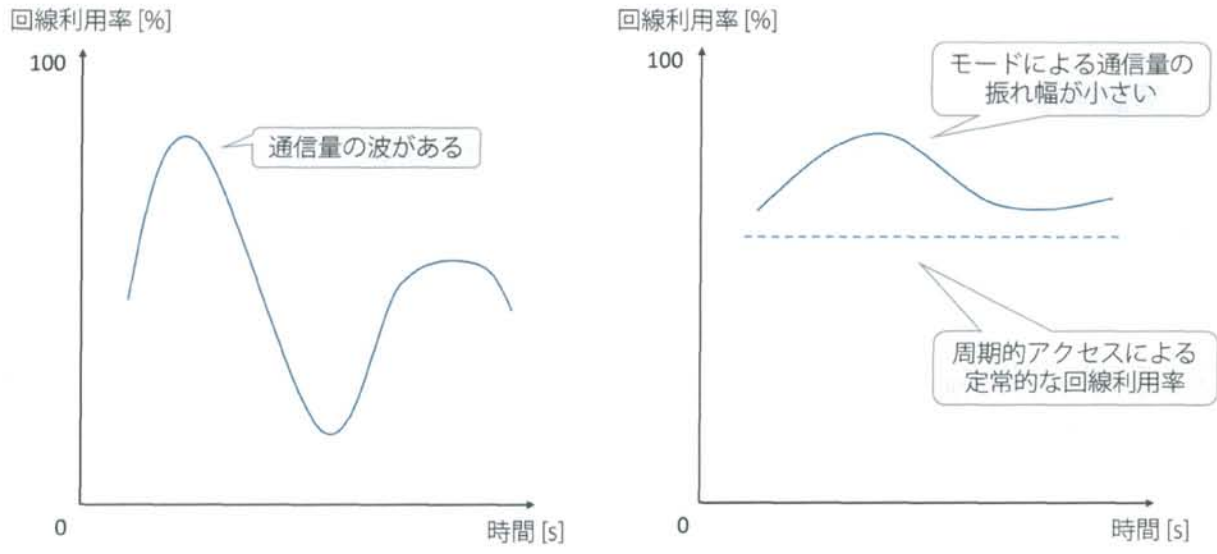
表 3.6　共有メモリによるバス回線利用率

共有メモリアクセスによる回線利用率[%]		サブシステム数			
		2	3	4	5
アクセス周期[ms]	10	19.8	29.8	39.7	49.6
	20	9.92	14.9	19.8	24.8
	30	6.61	9.92	13.2	16.5
	40	4.96	7.44	9.92	12.4
	50	3.97	5.95	7.94	9.92

今回の実験では共有メモリを用いた分散型アーキテクチャはサブシステム数 2 の場合で 20ms 周期書き込みが上限となった．そのため，CAN の回線利用率から考えると，9.92～19.8%の間で CAN 回線利用率の上限値が存在していると考えられる．そこで，分散型アーキテクチャで主流である，サブシステム数 4 の場合は 40ms の書き込み周期が好ましいと考えられる．

また，CAN の回線利用率の上限値は，フロー制御を追加するなどのプログラム次第で上げられると考える．しかし，CAN を回線利用率 100%近くで使用した場合，通信の再送や宇宙環境での予期せぬ通信速度の低下などで通信不可となる蓋然性が非常に高く，避けるべきである．従来の統括システムをもつ分散型アーキテクチャでは図 3.10 のようにモードによって通信量の波が存在する．しかし，統括システムが完全に通信を管理するため，バスラインを使用するノードは常に管理され，通信を最大限利用できる利点があ

る．しかし，共有メモリを用いた分散型アーキテクチャでは常時共有メモリへアクセスするため，図 3.10 のように一定の回線利用率を共有メモリへのアクセスで占めてしまう，よって，共有メモリへのアクセス周期は処理速度と信頼性のトレードオフとなり，最適なアクセス周期を探すことが共有メモリを用いた分散型アーキテクチャにおいて重要であるといえる．しかしながら今回の実験では改善の余地があるが，現状のプログラムでのアクセス周期の指標を得ることができた．



統括システムをもつ分散型アーキテクチャ　　共有メモリを用いた分散型アーキテクチャ

図 3.10　アーキテクチャ毎のバスライン利用率と時間のグラフ

第 4 章　共有メモリを用いた分散型アーキテクチャの標準化のための提案

前章では共有メモリを用いた分散型アーキテクチャが処理能力において従来の統括システムをもつ分散型アーキテクチャと比較し，有用であることを示した．そこでこの章では共有メモリを用いた分散型アーキテクチャの標準化に向けた設計を行う．

4.1　通信規格の選択

共有メモリを用いた分散型アーキテクチャでは従来のアーキテクチャと異なり，周期的にサブシステムが共有メモリへアクセスするため，通信回数が多くなる．そのため，通信規格の選択が重要となる．

通信規格においては規格を自作することや，組み込みシステムで一般的な通信規格である I^2C，車載ネットワークのみならず国際標準規格となっている CAN，衛星ネットワークでの標準規格である SpaceWire などが選択肢として存在する．

通信規格を自作する場合は，自由にプロトコルを設定できるため余計な機能を排除して通信効率をあげることができるが，民生品ほどの信頼性を確保することは難しい．しかし，既存の通信規格を用いる場合はその通信規格のプロトコルに加えて共有メモリにアクセスするための上位のプロトコルを用意する必要があるため，通信規格を自作する場合には 1 つのプロトコルにまとめられるメリットがある．

SpaceWire は現在大型衛星において標準的な通信規格である．超小型衛星におけるミッション機器は大型衛星の設計を流用することが多く，SpaceWire 規格での仕様となっていることが多い．SpaceWire を用いるとルータによって通信先を指定することができ，図 4.1 のように冗長な接続を設けて信頼性を向上させることも可能である．

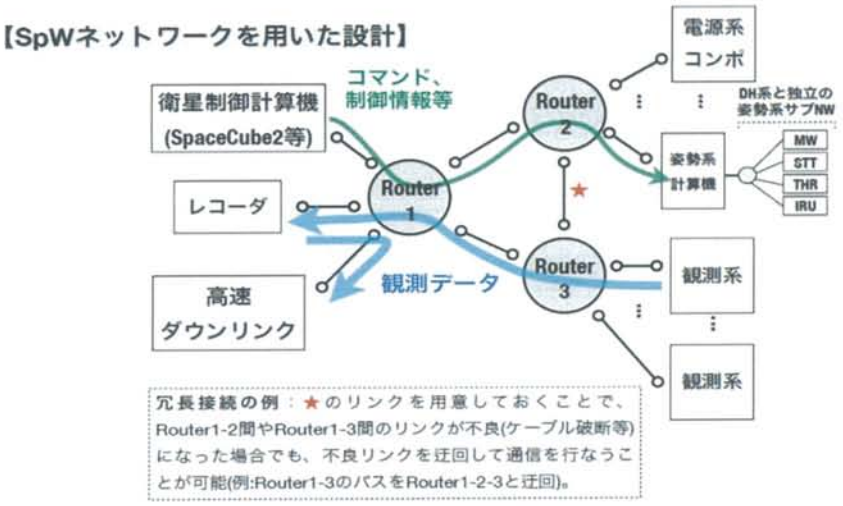


図 4.1　SpaceWire ネットワークによる衛星設計の概念図 [18)]

また，SpaceWire には以下の主な特徴がある．

- LVDS と DS リンクによる peer-to-peer の高速シリアルリンク
- 幅広い転送速度（2～400Mbps）
- 自由なパケットサイズ

・簡単なプロトコル
・ルータを用いたネットワークによる自由なネットワークトポロジー

以上のように SpaceWire は高性能で様々な利点があるが，超小型衛星においては，民生品利用ができないこと，SpaceWire 用のハーネスが高価であることなどから現状適していないといえる．

次に産業用の通信規格である，I^2C と CAN との比較となるが，両社ともバス通信が可能であり，民生品も広く普及されているため，超小型衛星の内部ネットワークに最適である．I^2C の場合はプロトコルがある程度短い分，誤り検出などの機能を有していないため，通信の信頼性を確保するためには上位のプロトコルを作る際に誤り検出を組み込む必要が出てくる．そのため，通信速度は劣るが信頼性の高い CAN を用いることとする．通信規格の比較を表 4.1 にまとめる．

表 4.1　通信規格の比較表

通信規格	自作	SpaceWire	I2C	CAN
価格	◎	×	○	○
民生品利用	△	×	○	○
通信速度	○	◎	○	△
信頼性	×	◎	○	◎

4.2　C&DH の仕様

C&DH の主な機能は以下のものである．
・コマンド信号の入力，解読，分配
・テレメトリデータの収集，構築
・モードによる状態管理
これらの機能を共有メモリとアーキテクチャに標準機能として持たせるとする．

そこで，2.3 節で述べた共有メモリを用いた分散型アーキテクチャの概念に従い，仕様を策定していく．

4.2.1　コマンド信号の入力，解読，分配の仕様

コマンドとは地球から衛星に向かって伝送される信号で，搭載機器の動作状態や動作モードを設定・移行させる信号および搭載機器のパラメータ設定・データアップロード信号のことである．コマンドの形式とタイミングには表 4.2 のような種類が存在する [19]．

表 4.2　コマンド種類

		実行タイミング	
		リアルタイムコマンド	タイムラインコマンド
実行形式	シングルコマンド	シングルコマンドかつ リアルタイムコマンド	シングルコマンドかつ タイムラインコマンド
	ブロックコマンド	ブロックコマンドかつ リアルコマンド	ブロックコマンドかつ タイムラインコマンド

ブロックコマンドはシングルコマンドを集めたものであり，よく使うコマンドシーケンスを 1 度に送信でき，情報量を少なくするメリットがある．また，1 つの機器を再起動させたい場合，機器を OFF するシングルコマンドを送り，その後機器を ON するシングルコマンドを送るより，ブロックコマンドで機器を OFF し ON させるという一連のコマンドを送った方が，通信が途絶える場合を考慮すると信頼性が高い．

さらにシングルコマンドには以下に示す 2 種類の形式が存在する．
・ディスクリートコマンド(DC: Discrete Command)

パラメータを含まないコマンドであり，主として衛星搭載機器の ON/OFF を行うために使用する．

・シリアルマグニチュードコマンド(SMC: Serial Magnitude Command)

パラメータを含んだコマンドであり，衛星搭載機器の状態設定を行うために使用する．

コンセプトから詳細なコマンド処理フローの仕様を決定すると図 4.2 のようになる．

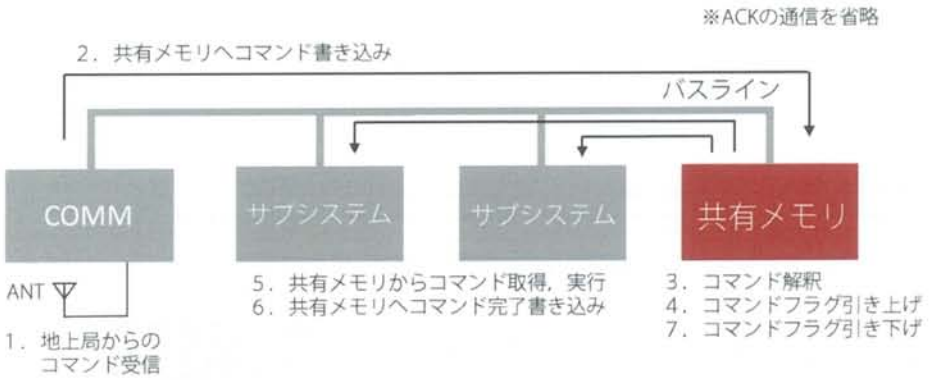


図 4.2　共有メモリを用いた分散型アーキテクチャにおけるコマンド処理フロー図

1. 地上局からのコマンド受信について

コマンドの入力は通信系の地上局コマンドの受信データを共有メモリに送信することで始まる．地上局からの通信の復調は通信系が行うものとする．

2. 共有メモリへのコマンド書き込みについて

通信系はサブシステムの周期的書き込み読み込みのタイミングで共有メモリへと送信するのではなく，地上局からのコマンド受信後割り込み処理を行い，受信後に即時共有メモリに送信を行う．

3. コマンド解釈について

サブシステムは共有メモリへアクセスし，コマンドを取得するため，共有メモリにはサブシステム毎の最小限の機能をコマンドフラグとして持たせる必要がある．また，よく用いるブロックコマンドのフラグを持たせればよい．

4. コマンドフラグ引き上げについて

シングルコマンドの際はそのフラグを立てればよい．

ブロックコマンドの場合はサブシステムをまたぐ場合と 1 つのサブシステムで完結する場合とに分かれる．1 つのサブシステムで完結する場合にはシングルコマンドと同様に扱えばよい．ブロックコマンドがサブシステムをまたぐ場合にはここまで完了したという報告を共有メモリへ送信し，コマンドを引き継ぐ必要がある．そのため，ブロックコマンドの場合にはフラグの状態がサブシステムをまたぐ分増えることとなる．

また，タイムラインコマンドの場合は共有メモリにおいて，コマンド実行時間まで待ち，コマンドフラグを挙げればよい．

5. 共有メモリからのコマンド取得，実行について

各サブシステムは共有メモリへの周期的書き込み読み込みの際に，共有メモリからシングルコマンドまたはブロックコマンドを送信される．

6. 共有メモリへコマンド完了書き込みについて

各サブシステムはコマンドを完了した際には周期的書き込み読み込みとは別に完了報告を共有メモリに送信する．

7. コマンドフラグ引き下げについて

シングルコマンドや 1 つの系で完結するブロックコマンドの場合はコマンドフラグを引き下げることによって一連のコマンドハンドリングが完了する．系をまたぐコマンドフラグの場合にはフラグの状態を次のフラグへと変更し，4～7 のシーケンスを繰り返す．

4.2.2 テレメトリデータの収集，送信の仕様

テレメトリとは衛星から地球に向かって伝送される信号の総称で，衛星搭載機器の動作状態およびモード遷移を確認するための信号（ハウスキーピングデータ），およびミッションデータのことを示す[20]．

テレメトリデータはサブシステムの周期的な書き込みによって得られたものを集めてテレメトリデータとして地上に送信する．テレメトリデータのフロー図を図 4.3 に示す．共有メモリはサブシステムの詳細仕様を把握しなくても済むように，サブシステムが自発的に共有メモリへデータを送信し，共有メモリは集めたデータをテレメトリデータとする仕様とする．

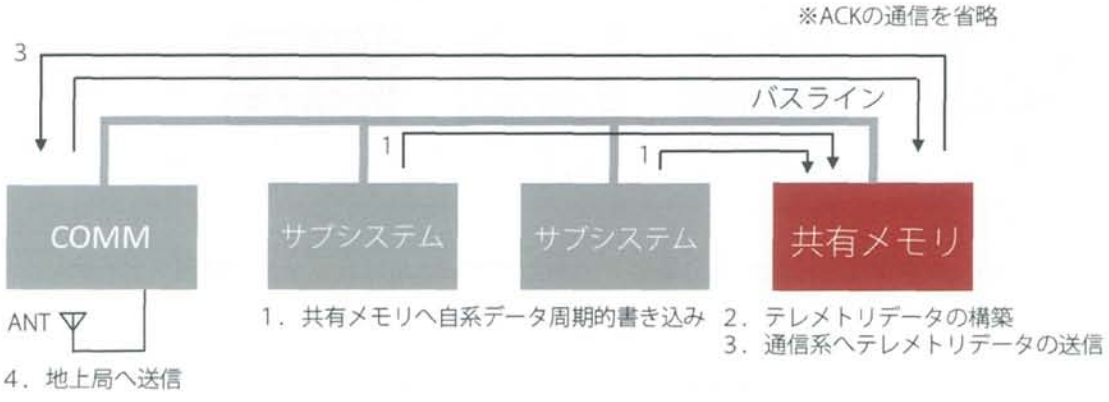


図 4.3　テレメトリデータフロー図

1. 共有メモリへの自系データ周期的書き込みについて

共有メモリへの周期的アクセスの際にテレメトリデータも付随して送る．

2. テレメトリデータの整理

サブシステムからテレメトリデータの一部が送られた際にテレメトリデータのフォーマットに合わせて構築する．また，サブシステムから送られてきたデータの平均値をテレメトリデータとする場合は計算を行い，テレメトリデータのサンプリングレートが周期的な書き込み量より少ない場合はデータを間引くといった処理を行う．

3. 通信系へテレメトリデータの送信について

地上局との通信モードまたは通信シーケンスに入った際に通信系へテレメトリデータの送信を行う．

4. 地上局へ送信

通信系は共有メモリから受信したテレメトリデータを変調し，地上局へ送信する．

4.2.3 モードによる状態管理の仕様

衛星はリアルタイムで常時監視・命令できないため，衛星を運用していくうえで衛星の状態を管理し，臨んだ状態に変更・維持させることが必須となる．一般的にそのような衛星の状態をモードと呼び，あるモードからあるモードへ移行することをモード遷移と呼ぶ．モード遷移にはトリガと呼ばれる条件がありモードとトリガは合わせて検討される．従来のアーキテクチャでは通常モード遷移は統括システムが行い，また，モードの情報は統括システムで完結するが，共有メモリを用いた分散型アーキテクチャではモードの決定を共有系が行い，モードの情報を分配する形となる．

ここでは当研究室で開発を進めている超小型 X 線観測衛星 ORBIS におけるモード遷移図草案を例として図 4.4 に示す．

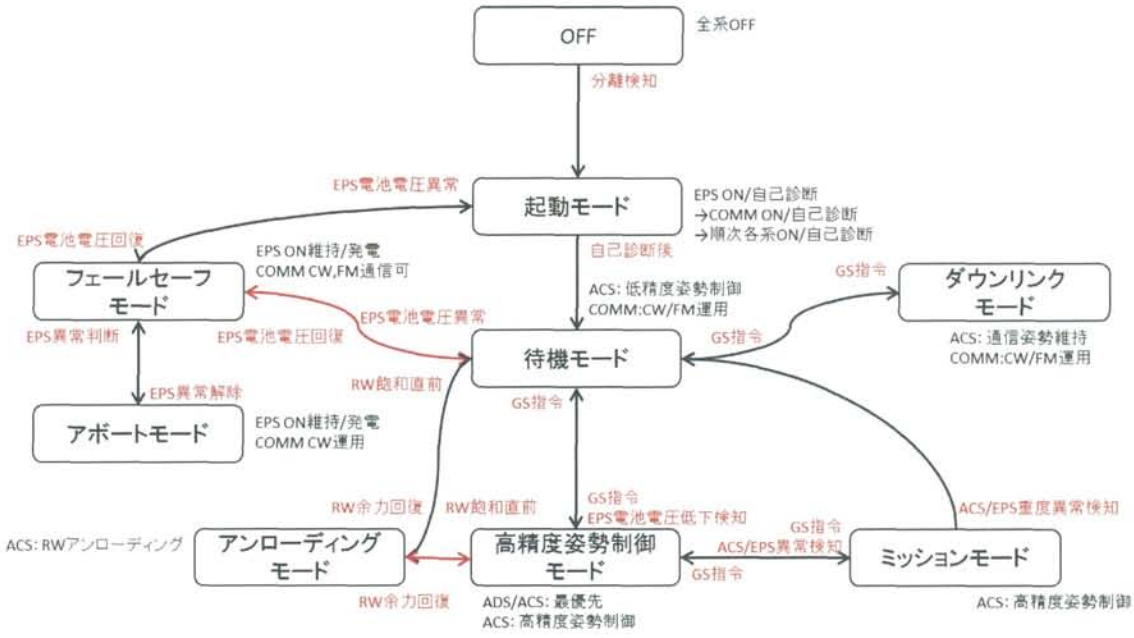


図 4.4　ORBIS モード遷移図

トリガには以下のものが考えられる.

・地上局（GS）からのコマンド

・サブシステムからのデータが閾値を上回る，または，下回る.

・指定された時間になる.

図 4.5 は共有メモリを用いた分散型アーキテクチャにおけるモード遷移フロー図である.

共有メモリは各モードのフラグを持っているものとし，各サブシステムからの周期的書き込みによるデータがモードのトリガ条件に当てはまった場合モード遷移を行う．モード遷移を行う際に共有系はブロードキャスト通信を行うことで，サブシステム毎にモードが違うといったずれをなくすことができる．また，ブロードキャスト通信を行う場合にはサブシステムの詳細を共有メモリは知らなくて済むためモジュールの変更にも影響はしない．例外としてバッテリ電池残量が足りないなどの電源系の異常事態が起きた際には電源系から直接ブロードキャストによってセーフモードへ入るものとする.

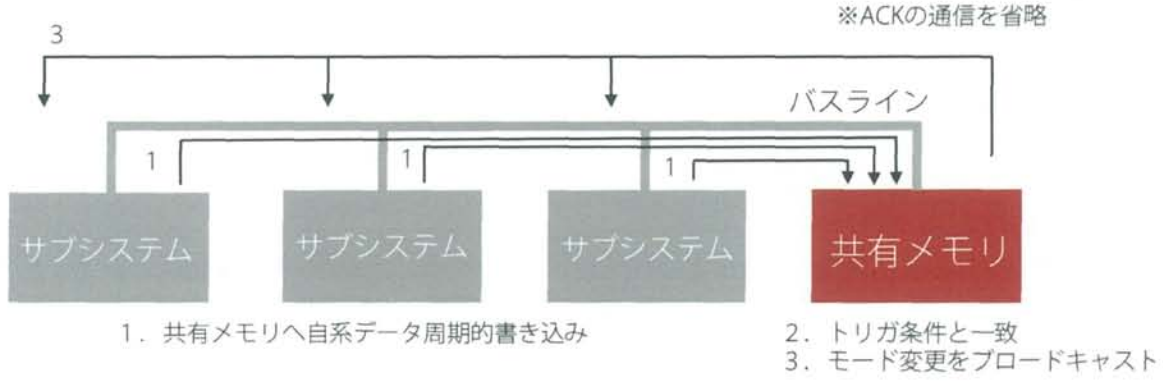


図 4.5　モード遷移フロー図

4.3 C&DH 通信プロトコルの設計

4.1，4.2 節により共有メモリと通信するデータ内容の仕様が決定した．そこで本節ではサブシステムが共有メモリにアクセスする際のプロトコルと共有メモリがサブシステムに返信をする際やブロードキャストする際のプロトコルを設計する．

共有メモリを用いた分散型アーキテクチャでの内部バスの通信プロトコルを OSI 基本参照モデルを参考に階層に分けると図 4.6 のようになる．

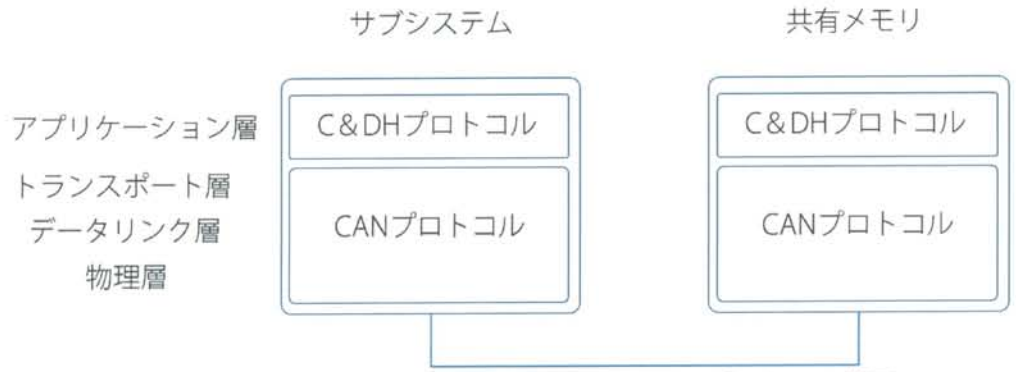


図 4.6 衛星内ネットワークと OSI 基本参照モデルとの比較図

C&DH プロトコルでは送るデータがどのような内容なのか，また，どのサブシステムから送られたデータなのかなどを示す必要がある．そのため，データ内容を判別するためのメタデータを示す ID 部と，送るデータそのものを格納するデータ部とに分ける．

また，サブシステムは共有メモリとのみ通信をし，共有メモリもまた，サブシステムとのみ通信を行うため，通信はサブシステムから共有メモリに対して行う場合と，共有メモリからサブシステムに対して行う場合と 2 つに大別できる．そのためそれぞれに対してプロトコルを設けることができる．以上から C&DH プロトコルの策定を行うと ID 部には以下の要求が存在する．

サブシステムから共有メモリへの通信における要求．

- IDA：ACK(確認応答)の通信か
  Yes/NO の 2 択のため，1bit.
- IDB：通信系からの地上局のコマンドデータか．
  Yes/NO の 2 択のため，1bit.
- IDC：ブロードキャストのデータか共有メモリへのユニキャストか．
  2 択のため，1bit.
- IDD：どの系から共有メモリに通信しているか．
  サブシステム数は通常 4〜8 程度であるので，8 種類とすると 3bit.
- IDE：コマンドの完了報告か自系データの送信か．
  2 択のため，1bit.
- IDF：IDE でコマンド完了報告の場合はどのコマンドに関してか，自系データの場合はどのデータか．
  ORBIS でのサブシステムのコマンドの最大数 136，自系データ種類最大数 123 を参考に 8bit.

以上よりサブシステムから共有メモリへの通信プロトコルは表 4.3 のようになる．

表 4.3 サブシステムから共有メモリへの C&DH 通信プロトコル

データ種類	IDA	IDB	IDC	IDD	IDE	IDF	データ部
ビット数	1	1	1	3	1	8	49

共有系からのサブシステムへの通信における要求.

・IDA：ACK の通信か

Yes/NO の 2 択のため，1bit.

・IDB：ブロードキャストのデータであるか.

Yes/NO の 2 択のため，1bit.

・IDC：コマンド送信の場合にはどのコマンドか

ORBIS でのサブシステムのコマンドの最大数 136 を参考に 8bit.

以上より共有メモリからサブシステムへの通信は表 4.4 のようになる.

表 4.4　共有メモリからサブシステムへの C&DH 通信プロトコル

データ種類	IDA	IDB	IDC	データ部
ビット数	1	1	8	54

4.4　アプリケーション設計

この節では共有メモリを用いた分散型アーキテクチャでのサブシステムの OBC に持たせるアプリケーションと共有メモリの OBC に持たせるアプリケーションをまとめる.

4.4.1　サブシステムの C&DH アプリケーション

サブシステムの OBC には以下のアプリケーションが必要となる.

・C&DH 通信プロトコル制御関数

CAN 通信によって受信した C&DH プロトコルの ID をチェックし，コマンドフラグテーブルを呼び出し，フラグを渡す．データを送信する際には CAN 通信制御関数に呼び出されデータに ID 部を付加させ C&DH プロトコルに構築する．通信に ACK が必要な際には CAN 通信制御関数を呼び出し，ACK を行う.

・コマンドテーブル

C&DH 通信プロトコル制御関数によって呼び出される．渡されたコマンドデータをもとにコマンドフラグを立てる．OBC のメインルーチンではコマンドフラグテーブルを監視することでコマンドフラグに変化があった際にコマンドを実行し，完了した際には CAN 通信制御関数を呼び出し，共有メモリへコマンド完了報告を行う.

以上のサブシステムの C&DH アプリケーションの関係を図にまとめると以下の図 4.7 のようになる.

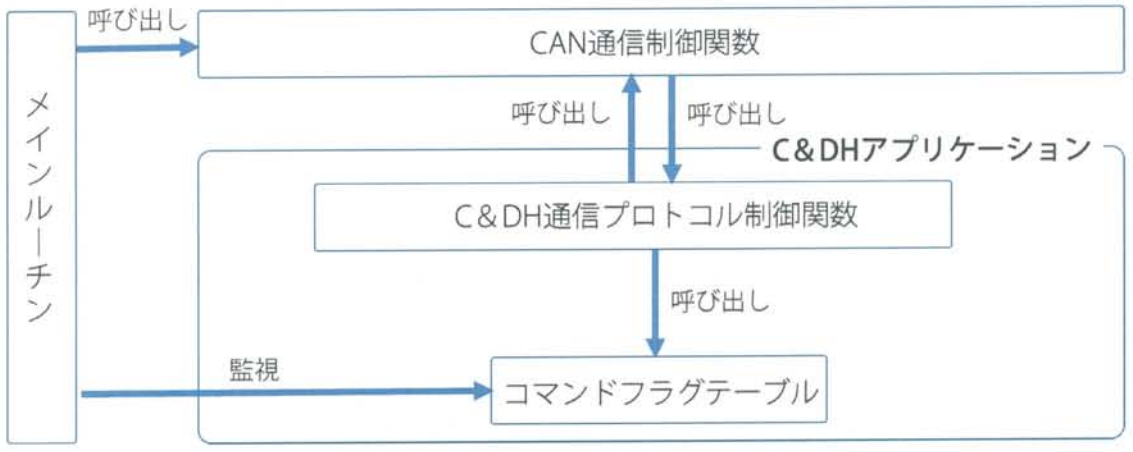


図 4.7　サブシステムの C&DH アプリケーション

4.4.2 共有メモリの C&DH アプリケーション

共有メモリをもつサブシステムの OBC には以下のアプリケーションが必要となる.

・C&DH 通信プロトコル制御関数

CAN 通信によって受信した C&DH プロトコルの ID をチェックし，適当な関数を呼び出し，値を渡す．データを送信する際には CAN 通信制御関数に呼び出されデータに ID 部を付加させ C&DH プロトコルに構築する．通信に ACK が必要な際には CAN 通信制御関数を呼び出し，ACK を行う.

・モード変更条件関数

C&DH 通信プロトコル制御関数に呼び出される．サブシステムからの自系データがモード変更条件のトリガに当てはまるかチェックを行う.条件に当てはまった場合モードフラグテーブルを呼び出す.

・モードフラグテーブル

モード変更条件関数によって呼び出される．該当するモードフラグの状態を変更する．OBC のメインルーチンではモードフラグテーブルを監視することでモードフラグに変化があった際にモード遷移を実行し，CAN 通信制御関数を呼び出し，全サブシステムへモード遷移の命令をブロードキャストする.

・テレメトリデータ生成関数

C&DH 通信プロトコル制御関数に呼び出される．サブシステムからの自系データをテレメトリのフォーマットに合うよう，並び替え，平均値取得，サンプリング数の調整を行う.

・コマンドフラグテーブル

C&DH 通信プロトコル制御関数に呼び出される．通信系から受信した地上局からのコマンドを受け取る，または，コマンド完了報告を受信した際にコマンドフラグの状態を変更する.

以上の C&DH アプリケーションの関係を図にまとめると以下の図 4.8 のようになる.

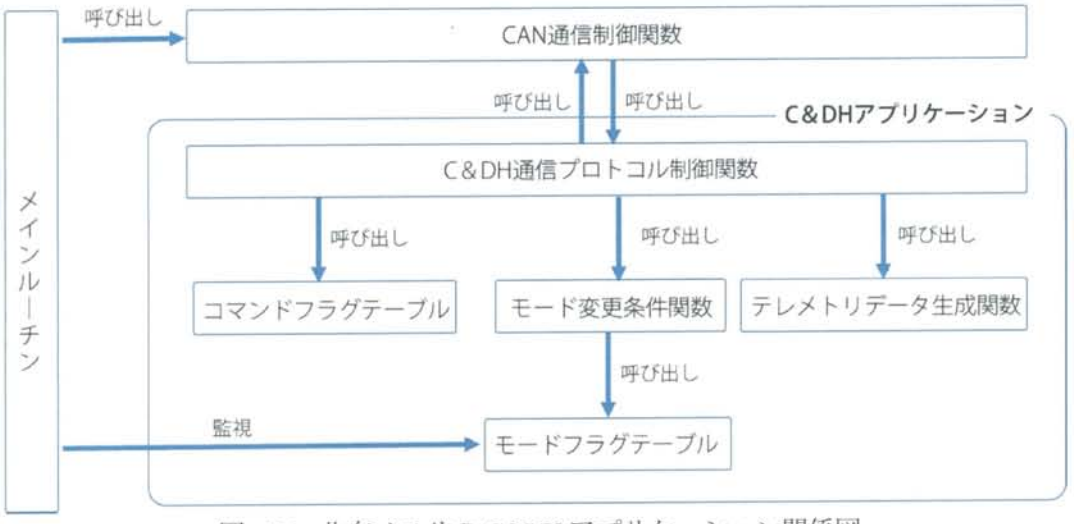


図 4.8　共有メモリの C&DH アプリケーション関係図

4.5 考察

前節よりコンセプトレベルから共有メモリを用いた分散型アーキテクチャの仕様を決め，より詳細な設計を行った．設計を進める上で以下の検討点が浮上した.

・1 回の通信でサブシステムは共有メモリへ 49bit のデータしか送信できない.

・50kg 級の超小型衛星に必須の機能であるといわれているリプログラミングの実装が難しい.

1 回の通信量について，CAN 通信は 1~8 バイトのデータを送信可能であり，データ部以外に 47bit を CRC や ID に使用する．そのため，1 回の通信で最大限データを送っても CAN 通信プロトコル，C&DH プロト

コルを合わせると送信される 104bit の内，実質のデータは 49bit となってしまうことがわかった．バスライン上では基本的にフラグのデータや各サブシステムのテレメトリのための小さなデータが送られるため問題はないが，大容量のデータが送られる場合には工夫が必要となる．ORBIS での現状のテレメトリデータが 1MB ほどであるため，共有メモリから COMM への通信には 163266 回の通信が必要となり，それには CAN の通信速度 125kbps では 125 秒間通信に費やされる．この問題にはいくつかの対応策がある，パスの時間までに分割してサブシステムの共有メモリへの周期アクセスの合間を縫って，ゆっくり送信をする対応や，共有メモリを COMM の基板に乗せて共有メモリと COMM 間の通信を無くしてしまうという対応である．しかし，サブシステム-共有メモリ間で，即時大容量のデータ送信が必要となるミッションには現状のアーキテクチャ実装方法では対応できていない．そのため，大容量のデータが必要となる通信ラインには別途通信ラインを設けるなどで対応する必要が出てくる．

リプログラミングについて，リプログラムとは軌道上で OBC のプログラムを地上から送信されたプログラムに変更することである．超小型衛星は打ち上げ時期の制約等により，地上で十分な長期テストを行えず，ソフトウェアのバグが取り切れずに打ち上げられる場合がある．また，姿勢制御において地上にて無重力環境下のテストが行えないため，軌道上で制御アルゴリズムの合わせ込みが必要となる．その際にリプログラミングを実装していれば衛星の信頼性が向上するため，重要な技術である．リプログラミングは失敗した場合に復帰できなくなる可能性があるため，リプログラムデータには高い信頼性が求められる．マイコンの実行ファイルは 200KB ほどであるため共有メモリを用いた分散型アーキテクチャでは分割送信が必要となる．その際に COMM→共有メモリ→サブシステムと分割送信，統合，分割送信，統合，実行のプロセスが必要となり，信頼性が低いといえる．対応としては COMM から直接サブシステムに送信することだが，やはり分割送信の必要があり，信頼性は比較的低い．そのため，リプログラミングを実装する際には他の機能より十分なテストを地上で行う必要がある．

2 つの検討点においてバスラインに大容量データを送る際の対応策としてデータを分割しパケット通信を行うことを挙げた．しかし，これを実装するにはバスラインを占有してまとめて送信する方法とバスラインを占有せず何番目のデータかわかるように ID を付けてゆっくり送信する方法と 2 つが考えられる．1 つ目の方法では他のサブシステムに通信を中断させる機能を持たせる必要があり，誤って実行しないようインヒビット機能の追加など C&DH 機能を複雑にしてしまう．2 つ目の方法ではパケットに ID を振り分ける必要があり，先述の ORBIS のテレメトリデータでは 163266 個の ID 振り分けが必要となるため，$2^{17}+1=131072<$ 163266 $<2^{18}+1=262144$ より，18bit を ID 振り分けに使用する必要がある．それをプロトコルへ追加すると実質的なデータ部がさらに小さくなり，通信回数が増える負のフィードバックに陥ってしまう．そのため，現状のアーキテクチャではバスライン上での大容量データの送信は対策としては可能であるが，柔軟なアーキテクチャとして標準化を目指すうえでは大容量データの送信の問題は根本から検討しなければならない．考えられる手段としてはバスラインをもう 1 つ用意し，大容量データのみを流す経路を別途設けることである．その場合また新たに C&DH の仕様を検討する必要がある．

最後に超小型衛星に共有メモリを用いた分散型アーキテクチャを適用する例と利点を述べる．超小型衛星の開発団体では，初号機では技術実証を行い，次号機以降はその技術を生かし，次の技術実証，または高性能を必要とするミッションを行う場合が多い．例として初号機には高精度姿勢制御技術実証，次号機にその技術を生かした天体観測ミッションを行う衛星での開発に共有メモリを用いた分散型アーキテクチャを適用した場合について述べる．

初号機の開発ではサブシステムを COMM・ADCS・共有系と切り分け開発を行う，開発はそれぞれの系で独立し，並行開発を行う，その際にサブシステム間のハードウェア・ソフトウェアの I/F が規定されているため，他系の開発に依存せず開発を進めることができる．共有系の開発にはテレメトリデータ・コマンドの設計，実装を行う．その際にすでに C&DH アプリケーションが用意されているため，実装時に行う作業が少なくて済む．アーキテクチャの I/F 規定に沿って開発されているため，システム統合時にサブシステム間が整合し，また，論理トポロジーがスター型であるためシステム統合時の作業量が少なくて済む．

次号機の開発では初号機で技術実証に成功したサブシステムは再利用することができ，不具合が出たサブシステムは交換を行い，再度開発を行う．そして，バスラインにミッション系を加えるなどサブシステ

ムの追加を行う．この際に仕様変更の影響はC&DHアプリケーションの仕様変更に留まり，変更点も少ないため，仕様変更に伴う手戻りを極力抑え込むことが可能である．

以上のように設計を進めたことで，アーキテクチャ特有，またアーキテクチャの実装手段による特徴が出てきた．通信量が多い点と大容量データの送信時の工夫が必要な点のデメリットは存在するが，対応する手段は存在するため．コスト削減のメリットがより大きいといえる．実装手段も特別複雑なものとはなっておらず，超小型衛星のアーキテクチャとして適当である．

第5章　結論

5.1　結論

本研究では低価格化を実現することができる超小型衛星用の新たなアーキテクチャを提案し，その性能の評価と実装に向けた設計を行った．

第2章では現在主流であるアーキテクチャを示し，低コストを実現するには標準化されたアーキテクチャが必要であり共有メモリを用いた分散型アーキテクチャならばシステム統合時の作業量が少なく，C&DH機能をアーキテクチャに組み込めるため，よりコスト削減に貢献できることを導いた．

第3章では待ち行列理論を用いて衛星アーキテクチャをモデル化し，アーキテクチャの抽象的なモデルでの性能を示した．また，実験を行うことによってCPUの処理や通信の諸処理を含めた具体的なモデルにおいても共有メモリを用いた分散型アーキテクチャの優位性を示すことができた．また，共有メモリを用いた分散型アーキテクチャを用いる際にはバスラインの利用率，すなわちサブシステム数とサブシステムの共有メモリへのアクセス頻度について信頼性を考慮して設定しなければならないことが分かった．

第4章では共有メモリを用いた分散型アーキテクチャを実装するために，コンセプトレベルから使用を策定し，より具体的アーキテクチャの設計を行い，アーキテクチャ実装方法を示した．そのうえで大容量データの送信という新たな検討点が浮かび上がってきた．

以上に1.2節で示した研究目的に対して，以下の結論を延べる．

- 低価格化を実現できる共有メモリを用いた分散型アーキテクチャを示した．
- 共有メモリを用いた分散型アーキテクチャと統括システムをもつ分散型アーキテクチャとを比較し，共有メモリを用いた分散型アーキテクチャの優位性を示した．
- 共有メモリを用いた分散型アーキテクチャの標準化に向けた設計を行い，アーキテクチャの実装方法を提示した．

5.2　今後の課題

第3章において，共有メモリを用いた分散型アーキテクチャにおいて，周期的なメモリアクセスによるCAN回線利用率がかなり低い結果となった．これにはまだまだ改善の余地があり，今回の研究では原因がハードウェアによるものなのかソフトウェアによるものなのかを特定することができなかった．そのため，今後CANの性能を最大限に引き出すハードウェア・ソフトウェアの構築が必要となる．

また，第4章では標準化に向けた設計までしか行っておらず，実装に至っていないため，アーキテクチャの実装を行い，実際の超小型衛星に適用することが課題である．

参考文献

1) Mengu Cho, Hirokazu Masui : 超小型衛星信頼性向上の Best Practice，第 60 回宇宙科学技術連合講演会講演集，3G09，2016
2) 中村友哉 : 超小型衛星ビジネスの現状とこれから，第 60 回宇宙科学技術連合講演会講演集，S03，2016
3) 趙孟佑，九州工業大学宇宙環境技術ラボラトリー : 宇宙環境技術研究センター～10 年間の成果とこれからの展望～，第 15 回宇宙環境技術交流会講演資料，2015
4) JAXA，SS-520-4 号機のミッション概要
5) インターステラテクノロジ社，http://www.istellartech.com/，（2017 年 1 月 26 日参照）
6) Julie Townsend, Bryan Palmintier, and Eric Allison. : Effects of a Distributed Computing Architecture on the Emerald Nanosatellite Development Process，14th AIAA/USU Conference on Small Satellites，2000
7) 船瀬流，PROCYON 開発チーム : 超小型衛星深宇宙探査機 PROCYON（プロキオン），第 15 回宇宙環境技術交流会講演資料，2015
8) 船瀬龍，川勝康弘，PROCYON プロジェクトチーム : はやぶさ 2 相乗り超小型深宇宙探査機 PROCYON の開発状況，第 58 回宇宙科学技術連合講演会講演集，2H12，2014
9) Nihon University CubeSat Project Official Web Site -NEXUS-, http://sat.aero.cst.nihon-u.ac.jp/nexus/system/index.html，（2017 年 1 月 26 日参照）
10) 太田佳，古賀将哉，鈴木聡太，谷津陽一，松永三郎 : 超小型衛星 TSUBAME の C&DH 系の軌道上評価と技術課題，第 59 回宇宙科学技術連合講演会講演集，2I06，2015
11) 金相均 : Study on Link-Bus Hybrid Information Network and Its optimization for small satellites，2009 年度東京大学大学院工学研究科航空宇宙工学専攻博士論文，2009
12) Bryan Palmintier, Christopher Kitts, Pascal Stang and Michael Swartwout. : A Distributed Computing Architecture for Small Satellite and Multi-Spacecraft Missions，16th Annual AIAA/USU Conference on Small Satellites，SSC02-IV-6，2014
13) 宮沢政清 : 待ち行列の数理とその応用，牧野書店，2006
14) 北岡正敏 : 例題でわかる待ち行列理論入門，日本理工出版会，2010
15) 稲井寛 : 基礎から学ぶトラヒック理論，森北出版株式会社，2014
16) 川島幸之助，塩田茂雄，河西憲一，豊泉洋，会田雅樹 : 待ち行列理論の基礎と応用，2014
17) 中尾司 : 動かして学ぶ CAN 通信，CQ 出版社，2010
18) 高橋忠幸，能町正治，福田盛介，高島健，湯浅孝行 : SpaceWire にもとづく衛星アーキテクチャ，第 57 回宇宙科学技術連合講演会講演集，1E01，2013
19) 滝澤潤一 : 再利用性と軌道上再構成能力に優れた衛星ソフトウェア・アーキテクチャに関する研究，2014 年度東京大学大学院工学研究科航空宇宙工学専攻博士論文，2014
20) 茂原正道，鳥山芳夫 : 衛星設計入門，培風館，2002

付録 A　回路図

ここでは第 3 章で用いた実験基板の OBC と CAN コントローラ, CAN トランシーバの回路図を掲載する.

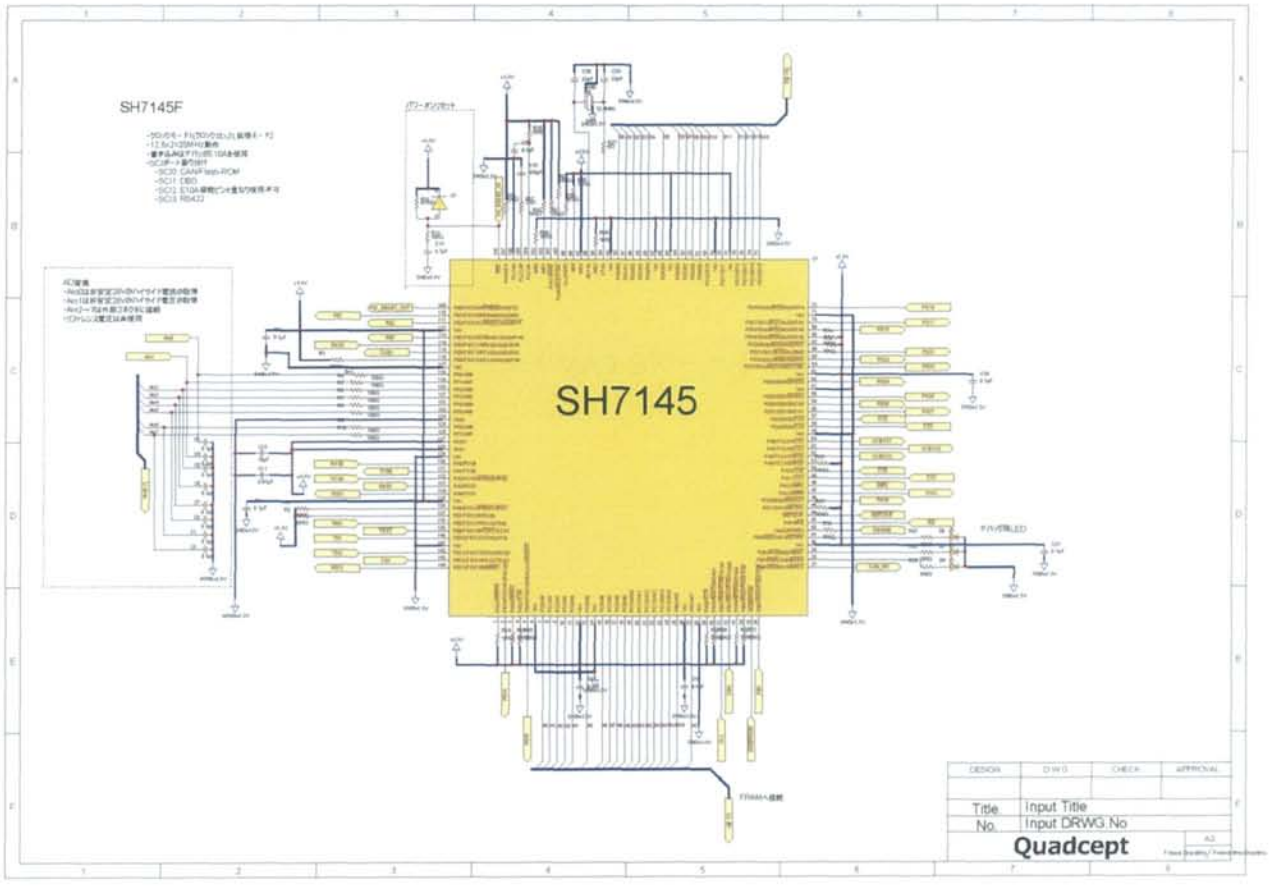


図 A.1　SH7145F 回路図

CANコントローラとトランシーバ回路

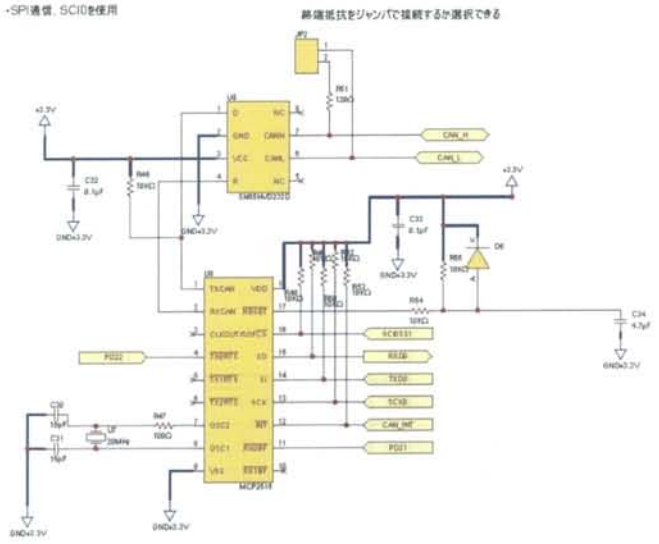


図 A.2　CAN コントローラと CAN トランシーバ回路図