W-CDMA向けSOC開発環境試行実験へのZIPC適用

日本電気(株) NECエレクトロンデバイス システムLSI事業本部 マイクロコンピュータ事業部 システム部
水瀬 晴美

W-CDMA向けSOC開発環境試行実験※1の一環として、設計/検証対象となるCPUソフトウェア※2の開発にZIPCを適用しました。ZIPCがMMI(Man-Machine Interface)の設計に適しているのは既に知られているので、ここでは専用ハードウェアに依存したドライバ部の設計に使用しています。これにより、SOC開発環境に適用する際の有効性や課題について述べたいと思います。

1．目的

ZIPCを適用した目的は次の３点です。

(A) 設計手法の有効性

(B) 早期検証の有効性

(C) 自動生成の有効性

2．システム概要

本試行実験におけるシステム構成を図 1 に示します。ベースバンド受信部は、ハードウェアブロック的な意味から BBIC(ベースバンド集積回路部)とも呼びます。本来W-CDMA基地局が生成したI/Qデータを入力として、止まり木チャネルデータの検出・受信を行います。「セルサーチ部」、「パスサーチ部」、「フィンガー部」及び「Rake 処理部」の４つのハードウェアブロックから構成されます。

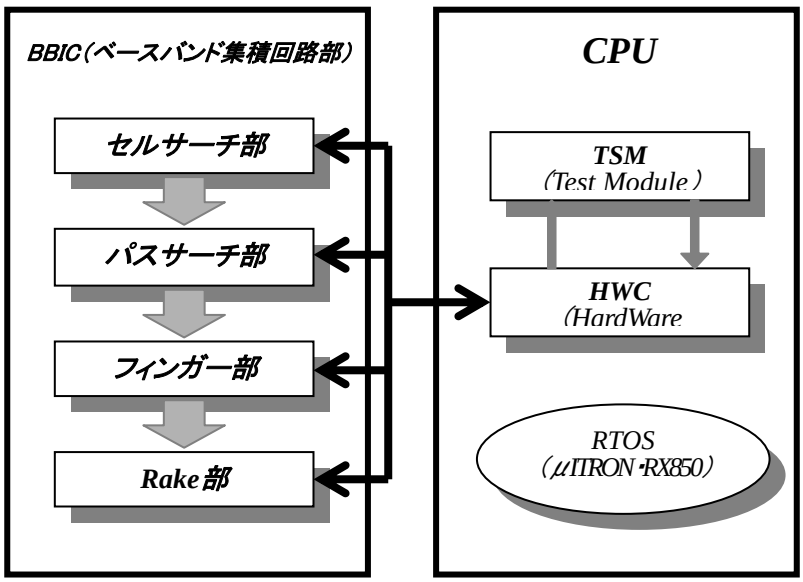


図１ システム概要

※1 本件で定義されるソフトウェア、ハードウェアは、試行実験を目的とするものであり、実システムへの流用等は考慮していません。よって、機能構成等は実システム(Program Interface Description for W-CDMA Mobile Station – Experimental System(Phase2)-:Ver.0.2 DoCoMo 発行)を参考にはしていますが、基本的に試行実験専用の独自仕様として位置付けます。
※2 ターゲットCPUはV850

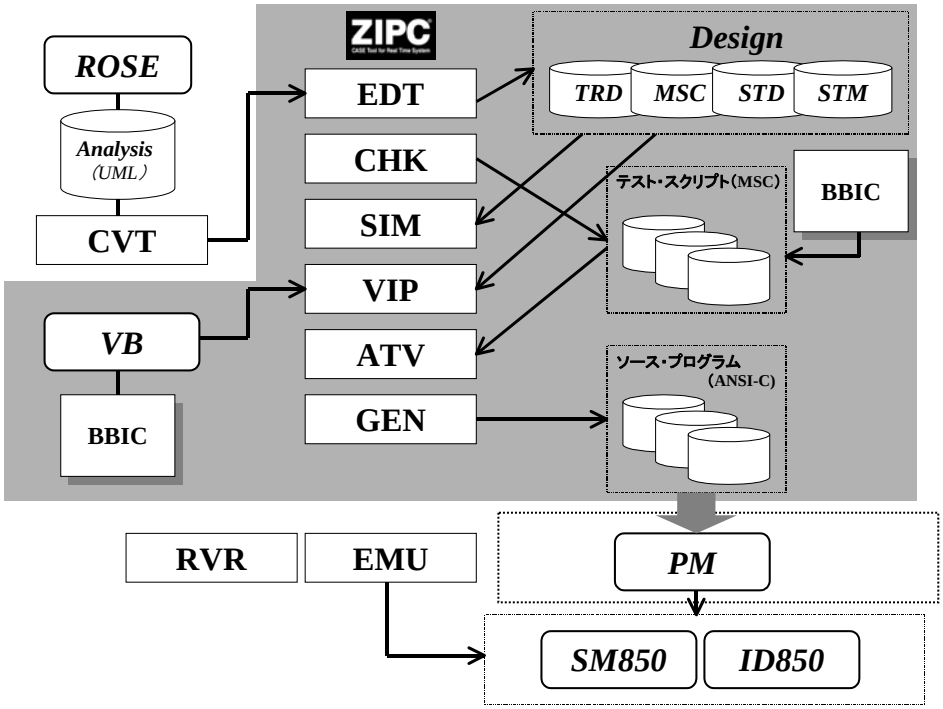


図 2　ZIPC 機能概要

CPUはV850 です。ソフトウェアは、RTOS(μITRON:RX850)上に構築される二つのタスク「TSM:Test Module」と「HWC:HardWare Control」からなります。「TSM」は「HWC」をテストするためのタスクですが、本稿では触れません。

3．環境概要

ZIPC には幾つかの機能が存在します。本稿で取り上げる機能は、図 2 で示す囲まれた内部のものです。枠外の上流ツール(UML※3 コンバート)や下流ツール(システム・シミュレータ／統合ディバッガなど)との連携機能については触れません。「EDT」は設計書エディタであり、タスク関連図(TRD: Task Relationship Diagram)、メッセージシーケンスチャート(MSC)、状態遷移表(STM: State Transition Matrix)や状態遷移図(STD: State Transition Diagram)等の表記法を用いて設計することができます。「CHK」はチェッカであり、前述のドキュメント群の静的な構文解析、整合性をチェックします。「SIM」はシミュレータであり、STM により記述されたステートマシンモデルをμITRON のタスクとしてシミュレートします。「ATV」(Auto Test and Verification)は自動試験／自動評価機能です。スクリプトを用意することで自動的にイベントをシミュレータ上の被タスクに対して発行し、その結果のログと検証スクリプトとを比較するものです。試験・検証スクリプトは MSC や TC(Timing Chart)で記述します。「VIP」(Visual Interface Prototyper)は Microsoft・ Visual Basic (以下 VB と記す) で作成した仮想ターゲットを「SIM」と連動させるものです。「GEN」とはジェネレータで、STM から ANSI-C のプログラム

※3 Unified Modeling Language

コードを自動生成します。

4．設計

ここでは、実際に設計した手順を示します。本手順は Trial 適用に限ったものでなく、実際の開発にも参考となるでしょう。ZIPC ではこのような手順に関して厳密なルールや手法がありませんので、ある程度の経験が必要と思われます。

［設計の手順］

(1) 要求仕様からシステムレベルの MSC を記述し、システムレベルのビヘイビアを掴む。（添付資料 A）
(2) システムレベルの MSC からシステムレベルの STD を記述し、より深くシステムレベルのビヘイビア、例えば並列状態等を洗い出す。（添付資料 B）
(3) システムレベルの STD を STM にコンバートし、システムレベルでの漏れ抜けを防止する。（添付資料 C）
(4) システムレベルの動作が明確になった時点で、HWC と BBIC に着目し、タスクレベルの MSC を設計する。（添付資料 D）
(5) タスクレベルの MSC からタスクレベルの STM を設計する。この段階で事象の型を定義する。（添付資料 E）

以上で HWC の設計が終了です。HWC がどのように動作するかが MSC と STD でまとめられ、矛盾や漏れ無く設計されているかが STM で表記されています。ZIPC ではこのようなドキュメントをツリー上で管理することができます。

5．検証

ZIPC で HWC を検証するには、様々な選択肢があります。選択肢は大きく３つのグループに分けられます。

1) 仮想ターゲット
2) STM レベル
3) 自動試験・検証

実際の BBIC の完成を待たずに CPU ソフトウ

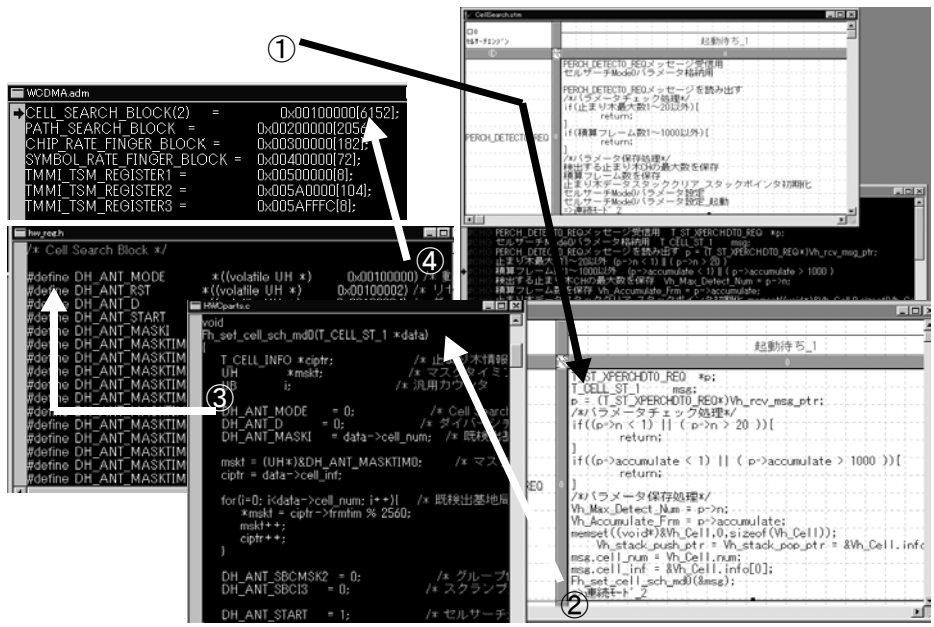


図３ SIM 定義

ェア側を設計書レベルで早期に検証できることは、次の点において有効です。

- BBIC と CPU ソフトウェア側の I/F を設計段階で検証可能
- CPU ソフトウェア側の論理設計が設計段階で完了
- CPU ソフトウェア側のタスク分割方針や BBIC の処理性能を考慮した設計が可能
- 既存関数を取り入れた形での検証が可能

以下、検証における手順を選択肢グループに分けて説明します。

5-1.仮想ターゲット

仮想ターゲットとは、CPU ソフトウェアと外部事象の関係を記述するものです。ここでは、BBIC を仮想ターゲットとします。ZIPC では、仮想ターゲットを表現する方法が二つあります。一つは BBIC を VB で記述する方法、もう一つは MSC スクリプトで BBIC の挙動を記述する方法です。それぞれ、VIP または ATV と SIM を接続して挙動をシミュレーションすることができます。本稿では VB による VIP 接続を選択します。

[VB の手順]

1. VB の Form 上に BBIC を描く。(添付資料 F 参考)
2. VB の Form 上に SIM とコミュニケーションをとるための OCX 部品(ZIPC により供給される)を配置する。
3. BBIC のビヘイビアを VB で記述する。

本項ではもう少し詳細に、SIM-VIP-VB の連携部分について説明します。図 3 に VB と接続するために SIM 側で必要なことは、以下の 4 点です。

[SIM 側の手順]

1. 日本語で記述された STM 内のアクションを置換するための定義を行う。
2. BBIC をアクセスする関数を C 言語で用意する。
3. BBIC レジスタマッピングを定義する。
4. SIM に BBIC レジスタ領域を知らせるためにアドレスマップを定義する。

[VIP 側の手順] (図 4)

VIP 側では VB で配置したそれぞれの BBIC のレジスタ・オブジェクトを選択します。それぞれのレジスタ・オブジェクトのポート名称にアドレスをマッピングします。これにより、SIM と VIP がリンクされることになります。つまり SIM 側で BBIC の IO レジスタ空間としてして定

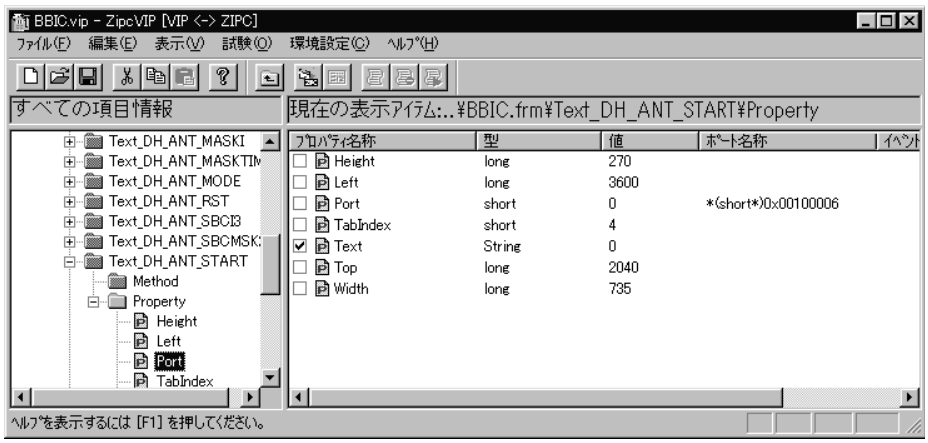


図 4　VIP 定義

義された領域にアクセスが行われると VIP を呼び出すことができます。(リスト 1)

[VB 側の手順]

ZIPC の OCX により ZvipComm1_GetVipEvent()を使用し、VIP からのイベントを受け取ることができます。VIP は VIP 上で定義されたオブジェクト名称を VB に受け渡します。

CPU ソフトウェアで BBIC のレジスタへ書き込まれた際の BBIC の挙動を VB でモデリングします。ライトの一定時間（ここでは 100μSec）後に CPU に割り込みを発生させる場合は、SIM にタイムアウトイベントを VB から要求し、一定時間経過後、SIM からタイムアウトの通知を VB が受けると、VIP に割り込み発生を要求し、SIM に VIP から割り込みを発生させます。BBIC から CPU 方向へのアクセスは ZVipComm1.SetVipEvent を使用します。(リスト 2)

5-2.STM レベル

ZIPC では設計書記述レベルをあるルールに基づいた自然言語（日本語）レベルと ANSI-C レベルの記述を選択することができます。本適用では、BBIC にアクセスする個所が既にライブラリ部品として存在したので ANSI-C レベルで行うことにしました。この際、必要とする情報ファイルは、5-1 で述べたように自然言語を C 言語に置換するための置換情報ファイルです。直接 STM に C 言語を記述しても良いですが、今回は置換して日本語の STM を見ながらシミ

```
Private Sub ZVipComm1_GetVipEvent(ByVal strEvtName As String, ByVal varEvtData As Variant)
    Dim MyString

    If strEvtName = "Text_DH_ANT_MODE.Port" Then
            MyString = CStr(varEvtData)         ' MyString には、DH_AND_MODE
            Text_DH_ANT_MODE.Text = MyString
            Line_DH_ANT_MODE.BorderColor = &HFF              ' &H80000008
    ElseIf strEvtName = "Text_DH_ANT_START.Port" Then
            MyString = CStr(varEvtData)
            Text_DH_ANT_START.Text = MyString
            Line_DH_ANT_START.BorderColor = &HFF
            '割込み発生待ち
            Call CellEndInterrupt
```

リスト 1

```
    Private Sub CellEndInterrupt()
        'ZIPC Simulator に 100 マイクロ後に VB にタイマを入れさせる
        ZVipComm1.SetVipTimer 0, 100000000

    End Sub

    Private Sub ZVipComm1_VipTimerTick(ByVal nTimerID As Long)
        If nTimerID = 0 Then
            'ここで割込みを発生させる
            ZVipComm1.SetVipEvent "TextINTERRUPT.Port", 1
            LineINT1.BorderColor = &HFF
            LineINT2.BorderColor = &HFF
            LineINT3.BorderColor = &HFF
        End If

    End Sub
```

リスト 2

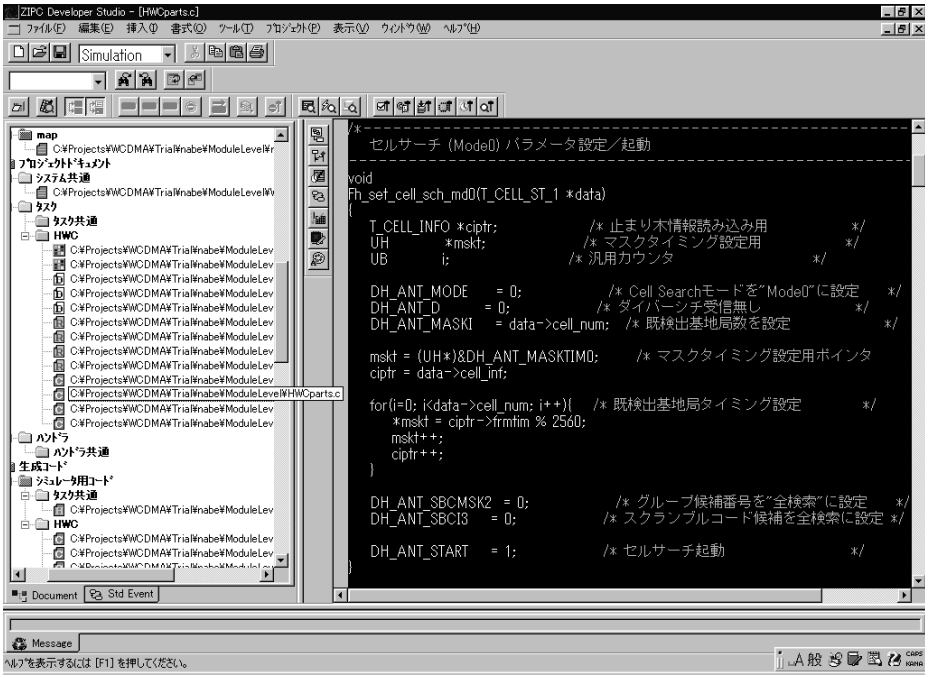


図 5　既存関数の取り込み

ュレーションを行える方式を選択しました。C 言語レベルといっても置換情報を用意すれば見かけは日本語レベルでデバックを進めることが可能です。BBIC アクセス部分の既存 C 言語関数はまったく変更せずに、そのまま ZIPC プロジェクトに登録してシミュレーションができます。(図 5)

5-3.自動試験・検証

本適用においては時間的な問題で、ATV の自動試験機能のみ適用しました。自動検証機能に関しては本項では触れません。

HWC は TSM からのコマンド要求を受けて BBIC に対してアクセスを開始します。これをシミュレーションする際に STM から直接イベントを発生させるやり方と、MSC によりイベントを自動的に発生させる方式を選択することができます。MSC による試験スクリプトは添付資料 G を参照して下さい。

6．自動生成

ZIPC は４つのコード生成方式を持っていて、オプション選択することができます。ZIPC による自動生成とプログラマによる開発のコードサイズ比較を表１に示します。

RAM について調査していないのは、ほとんど ZIPC とプログラマで差がないためです。

ROM 効率において、一般にアセンブラから C 言語に変換した場合のサイズアップが通常 1.3 倍といわれるのを考えると、プログラマ記述 C 言語からのサイズアップは約 1.2 倍というのは現実的な数値であり、プログラミング工数や後の保守を考えれば十分適用の範囲と云えます。

7．設計/検証に要する時間

本適用は、ZIPC 経験者と未経験者の二人で取り組みました。携帯端末のドライバ設計は二

表 1

	実行コード	const	Total
ZIPC	7508	336	7844
ZIPC(標準型生成)	7404	1072	8476
元の	6376	344	6720
比率(Offset 型生成)	*1.18	*0.97	*1.17
比率(標準型)	*1.16	*3.12	*1.26

単位 byte

人とも初めてです。検証までの環境を構築するのに要した時間は、以下の通りです。

- 要求仕様→システムレベルの MSC 化：3 日
- MSC→STD 化：3 日
- STD→STM 化：0.5 日
- タスクレベルの MSC 化：1 日
- タスクレベルの STM 化：1 日
- 仮想ターゲット環境構築：1 日

設計には時間を要します。この時間が、大規模化したシステム設計の後工程で不具合が発見された際にかかるフィードバックを低減することになります。

また、試験スクリプト一つを流すのにかかるシミュレーション時間は１分程度です。抽象化レベルの検証が有効といえる一つの数値だと思います。

8．今後の課題

ZIPC を SOC の CPU ソフトウェアに適用するには、今後以下のような課題があります。

まずは、ZIPC へお願いしたいことです。

1) 仮想ターゲットのモデリングは C 及び C++ が主流であり、それに対応できるようにしてほしい。
2) SOC として主流になっている CPU+DSP 等マルチプロセッサに対応してほしい。
3) シミュレーション速度向上のため、Visual C++を使用した Native Compile 環境を充実させてほしい。そのためには、Windows 上 RTOS の開発とその環境をサポートしてほしい。
4) 通信系の仕様表現の主流である SDL からのインポートに対応してほしい。

半導体ベンダとしての下流工程へのソリューションは、以下のように考えます。

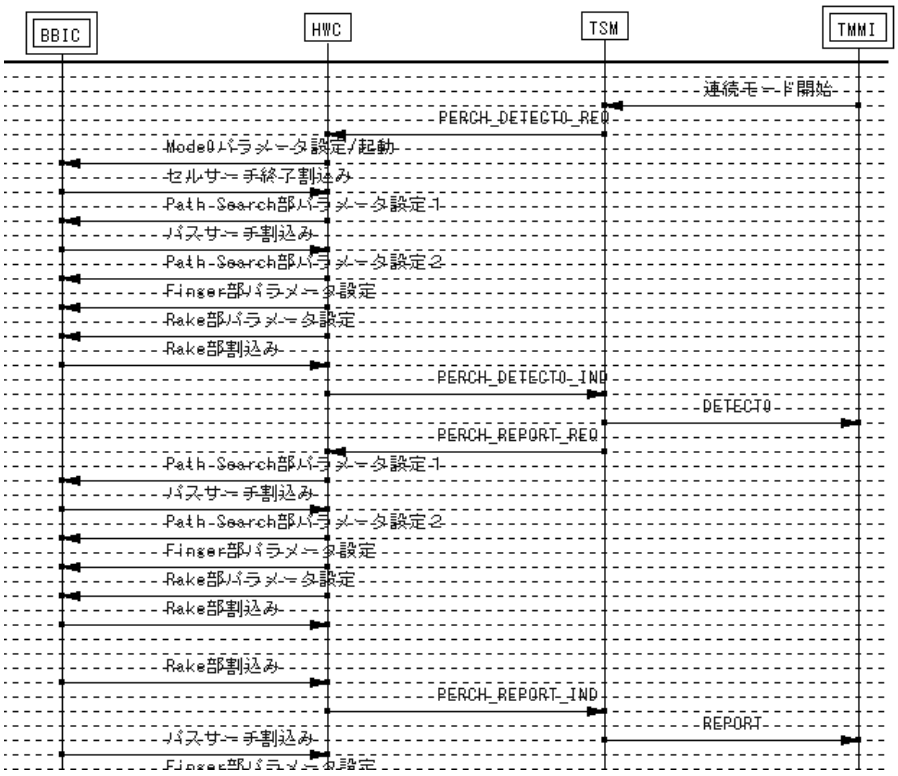
5) 上記 1)に対応して、仮想ターゲットを接続できるコードシミュレーション環境の実現。
6) 最後に実機環境の試験・検証と接続。

仮想ターゲットである BBIC を VB でモデル化するのは限度があり、デバイスの設計環境で構築したモデルと共有化すること、また上流環境で使用した仮想ターゲットを使用できるシミュレータの存在が重要になってきます。

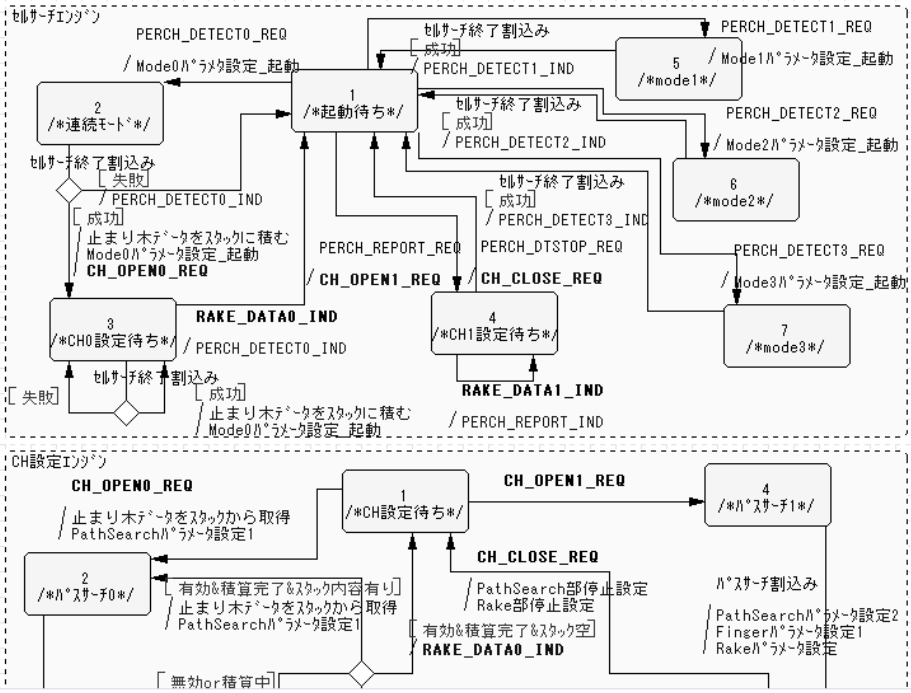
更には、今後いかに実機環境の試験・検証と接続できるかが需要です。上流環境で流した試験スクリプトを実機環境でも同じように流せ、かつシミュレーション時の結果と実機上の結果を自動的に付き合わせる機構が必要です。

（みずせ はるみ）

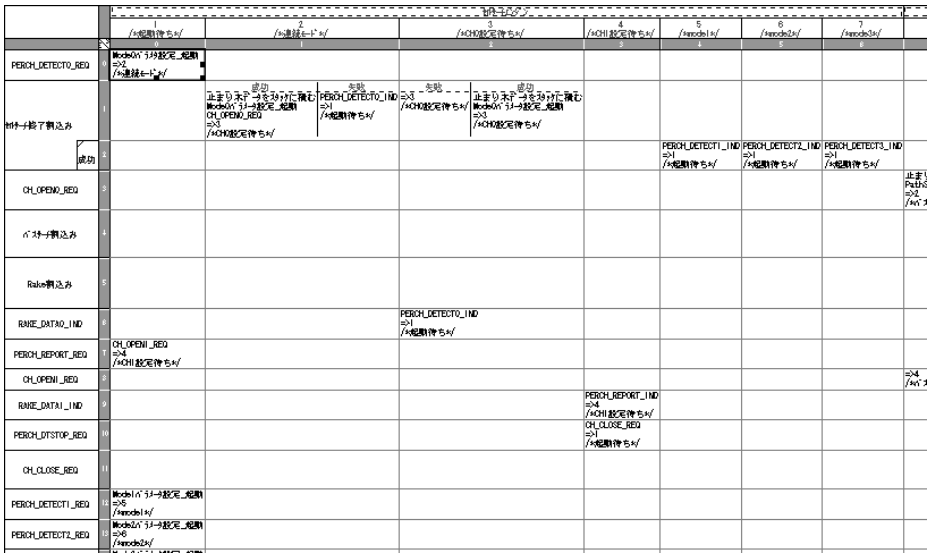
添付資料 A


BBIC
HWC
TSM
TMMI
連続モード開始
PERCH_DETECT0_REQ
Mode0パラメータ設定/起動
セルサーチ終了割込み
Path-Search部パラメータ設定1
パスサーチ割込み
Path-Search部パラメータ設定2
Finger部パラメータ設定
Rake部パラメータ設定
Rake部割込み
PERCH_DETECT0_IND
DETECT0
PERCH_REPORT_REQ
Path-Search部パラメータ設定1
パスサーチ割込み
Path-Search部パラメータ設定2
Finger部パラメータ設定
Rake部パラメータ設定
Rake部割込み
Rake部割込み
PERCH_REPORT_IND
REPORT
パスサーチ割込み
Finger部パラメータ設定

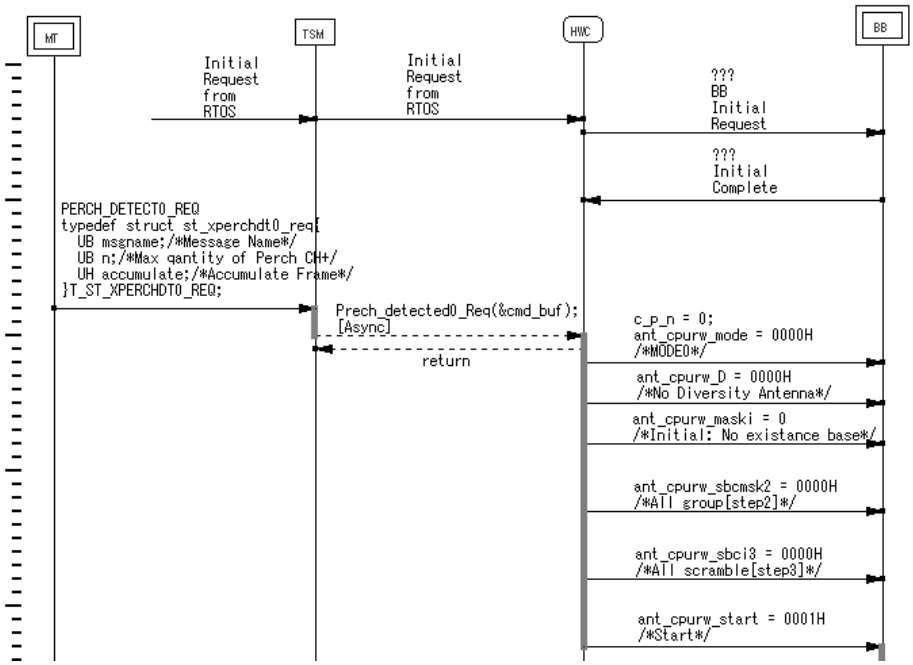
添付資料 B


ｾﾙｻｰﾁｴﾝｼﾞﾝ
PERCH_DETECT0_REQ
/ Mode0ﾊﾟﾗﾒﾀ設定_起動
ｾﾙｻｰﾁ終了割込み
[成功]
/ PERCH_DETECT1_IND
PERCH_DETECT1_REQ
/ Mode1ﾊﾟﾗﾒｰﾀ設定_起動
5
/*mode1*/
2
/*連続ﾓｰﾄﾞ*/
1
/*起動待ち*/
ｾﾙｻｰﾁ終了割込み
[成功]
/ PERCH_DETECT2_IND
PERCH_DETECT2_REQ
/ Mode2ﾊﾟﾗﾒｰﾀ設定_起動
6
/*mode2*/
ｾﾙｻｰﾁ終了割込み
[失敗]
/ PERCH_DETECT0_IND
[成功]
止まり木ﾃﾞｰﾀをｽﾀｯｸに積む
Mode0ﾊﾟﾗﾒｰﾀ設定_起動
CH_OPEN0_REQ
ｾﾙｻｰﾁ終了割込み
[成功]
/ PERCH_DETECT3_IND
PERCH_REPORT_REQ
/ CH_OPEN1_REQ
PERCH_DTSTOP_REQ
/ CH_CLOSE_REQ
PERCH_DETECT3_REQ
/ Mode3ﾊﾟﾗﾒｰﾀ設定_起動
3
/*CH0設定待ち*/
RAKE_DATA0_IND
/ PERCH_DETECT0_IND
4
/*CH1設定待ち*/
7
/*mode3*/
ｾﾙｻｰﾁ終了割込み
[失敗]
ｾﾙｻｰﾁ終了割込み
[成功]
止まり木ﾃﾞｰﾀをｽﾀｯｸに積む
Mode0ﾊﾟﾗﾒｰﾀ設定_起動
RAKE_DATA1_IND
/ PERCH_REPORT_IND
CH設定ｴﾝｼﾞﾝ
CH_OPEN0_REQ
/ 止まり木ﾃﾞｰﾀをｽﾀｯｸから取得
PathSearchﾊﾟﾗﾒｰﾀ設定1
1
/*CH設定待ち*/
CH_OPEN1_REQ
4
/*ﾊﾟｽｻｰﾁ1*/
2
/*ﾊﾟｽｻｰﾁ0*/
[有効&積算完了&ｽﾀｯｸ内容有り]
/ 止まり木ﾃﾞｰﾀをｽﾀｯｸから取得
PathSearchﾊﾟﾗﾒｰﾀ設定1
CH_CLOSE_REQ
/ PathSearch部停止設定
Rake部停止設定
ﾊﾟｽｻｰﾁ割込み
/ PathSearchﾊﾟﾗﾒｰﾀ設定2
Fingerﾊﾟﾗﾒｰﾀ設定1
Rakeﾊﾟﾗﾒｰﾀ設定
[有効&積算完了&ｽﾀｯｸ空]
RAKE_DATA0_IND
[無効or積算中]

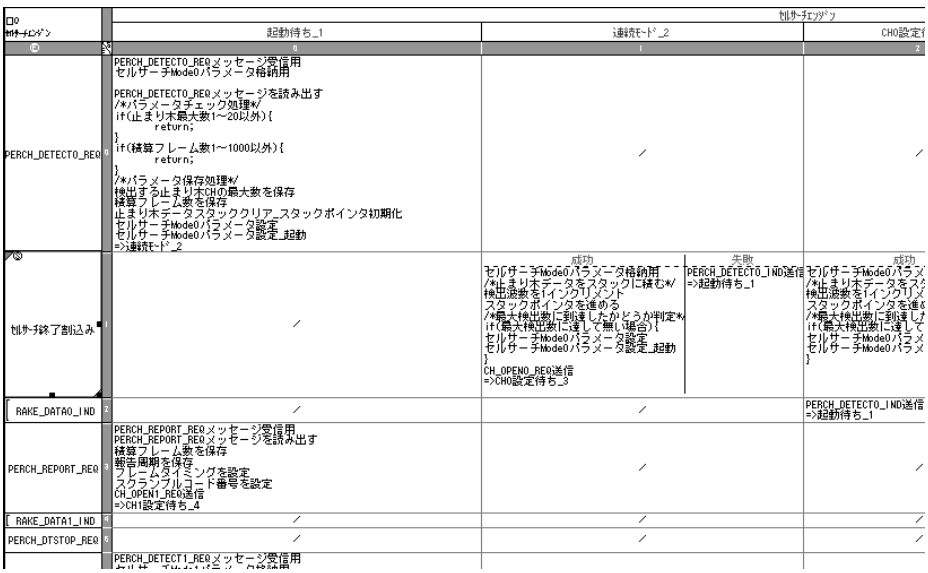
添付資料 C



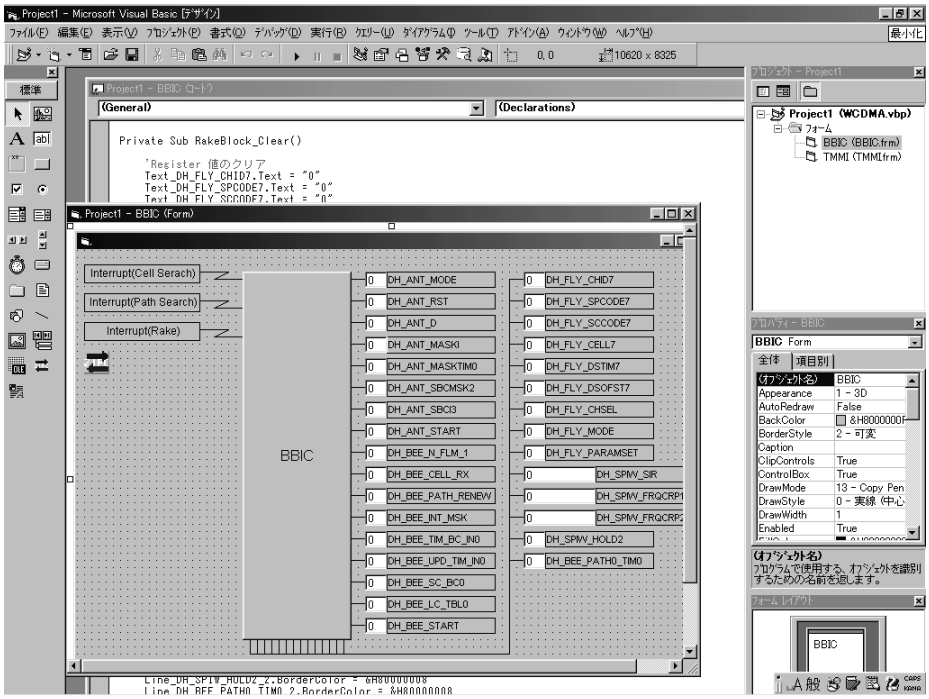
添付資料 D


MT
TSM
HWC
BB
Initial
Request
from
RTOS
Initial
Request
from
RTOS
???
BB
Initial
Request
???
Initial
Complete
PERCH_DETECT0_REQ
typedef struct st_xperchdt0_req{
UB msgname;/*Message Name*/
UB n;/*Max qantity of Perch CH*/
UH accumulate;/*Accumulate Frame*/
}T_ST_XPERCHDT0_REQ;
Prech_detected0_Req(&cmd_buf);
[Async]
return
c_p_n = 0;
ant_cpurw_mode = 0000H
/*MODE0*/
ant_cpurw_D = 0000H
/*No Diversity Antenna*/
ant_cpurw_maski = 0
/*Initial: No existance base*/
ant_cpurw_sbcmsk2 = 0000H
/*ATI group[step2]*/
ant_cpurw_sbci3 = 0000H
/*ATI scramble[step3]*/
ant_cpurw_start = 0001H
/*Start*/

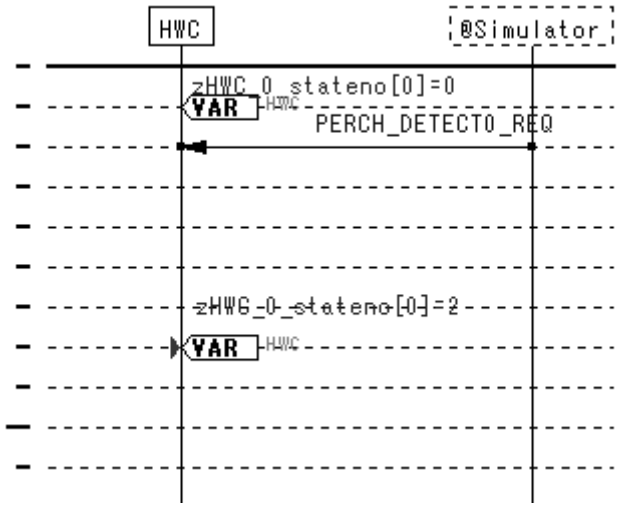
添付資料 E


起動待ち_1
連続ﾓｰﾄﾞ_2
PERCH_DETECT0_REQ
PERCH_REPORT_REQ
RAKE_DATA0_IND
RAKE_DATA1_IND
PERCH_DTSTOP_REQ
成功
失敗
PERCH_DETECT0_IND送信
=>起動待ち_1
CH_OPEN0_REQ送信
=>CH0設定待ち_3
CH_OPEN1_REQ送信
=>CH1設定待ち_4

添付資料 F


Project1 - Microsoft Visual Basic [デザイン]
Project1 - BBIC (Form)
Interrupt(Cell Serach)
Interrupt(Path Search)
Interrupt(Rake)
BBIC
DH_ANT_MODE
DH_ANT_RST
DH_ANT_D
DH_ANT_MASKI
DH_ANT_MASKTIM0
DH_ANT_SBCMSK2
DH_ANT_SBCI3
DH_ANT_START
DH_BEE_N_FLM_1
DH_BEE_CELL_RX
DH_BEE_PATH_RENEW
DH_BEE_INT_MSK
DH_BEE_TIM_BC_IN0
DH_BEE_UPD_TIM_IN0
DH_BEE_SC_BC0
DH_BEE_LC_TBL0
DH_BEE_START
DH_FLY_CHID7
DH_FLY_SPCODE7
DH_FLY_SCCODE7
DH_FLY_CELL7
DH_FLY_DSTIM7
DH_FLY_DSOFST7
DH_FLY_CHSEL
DH_FLY_MODE
DH_FLY_PARAMSET
DH_SPMV_SIR
DH_SPMV_HOLD2
DH_BEE_PATH0_TIM0
Project1 (WCDMA.vbp)
BBIC (BBIC.frm)
TMMI (TMMI.frm)
BBIC Form

添付資料 G


HWC
@Simulator
zHWC_0_stateno[0]=0
VAR
HWC
PERCH_DETECT0_REQ
zHWC_0_stateno[0]=2
VAR
HWC