Vol.2012-CDS-5 No.26
Vol.2012-DCC-2 No.26
2012/10/18

既存資産の拡張開発に適した ドライバ層エミュレーションによる実機レス開発方式

岡本周之[†1]

近年，組込み機器ではテスト・デバッグの効率向上が重要な課題であり，アプリケーション開発用の実機レス開発方式などが提案されている．しかし，組込み機器ではアプリケーション層とミドルウェア層が密接に絡んでおり，アプリケーション層に加えてミドルウェア層がエミュレータ上で開発できることが求められる．本稿では，ミドルウェア及びアプリケーションの開発が可能な，ドライバ層エミュレーションによる汎用 PC 上での実機レス開発方式について提案する．既存資産の拡張開発を行っている実際のデジタル TV 製品のソフトウェア開発に本方式を適用した結果，実機によるテスト・デバッグ期間を 42%に短縮し，組み合わせテスト時のテスト・デバッグ効率が 129%に向上する効果が得られ，大規模組込み機器における，本提案手法の有効性を検証できた．

Driver Emulator Based Software Development Environment for Extended Embedded System

CHIKASHI OKAMOTO[†1]

Testing and debugging efficiency is important for software development of the embedded system. Many emulators have been proposed to provide the software development environment before hardware completion. However simulator for application is not enough for developing middleware. CPU emulator is slow. Some latest platforms are provided with simulator for middleware developing, but it isn't suitable for the existed software. So we propose simulator based software development environment for the extended embedded system. We have applied it to the digital TV software development and confirmed its efficiency. It reduced to 42% duration and improved to 129% efficiency.

1. はじめに

近年，組込みシステムの開発ではシステムの多機能化に伴いソフトウェア開発規模が増加している．製品開発費に占めるソフトウェア開発費は, 2002 年度の 36.3%から 2010 年度の 50.0%にまで増加した[1].
一方組込みシステムの分野においても，開発製品の競争力強化のために開発期間の短縮と開発コストの低減がよりいっそう求められている．

組込みシステムの開発に特徴的な点として，ハードウェアとソフトウェアの開発が並行して進められることが挙げられる．デジタル TV（以下 DTV）の例では，例えば新たにネットワーク機能を搭載する場合，放送番組などの動画データを著作権に留意しながら効率よく伝送するための専用ネットワーク LSI を搭載したハードウェアの開発と，ネットワーク機能を活かすソフトウェア システムの開発が同時進行する．

このような組込みシステム開発をソフトウェア開発の視点から見た場合に，情報系システム開発のような汎用プラットフォーム（PC）を想定したソフトウェア開発とは異なる問題として，

(1) 開発の前半の段階において，ソフトウェアを動作させるハードウェアがない
(2) 開発期間を通じて，テスト・デバッグのために用意できるハードウェア数が不足する傾向にある

という点が挙げられる．

これらの問題に対しては，ハードウェアを必要としないソフトウェアのテスト・デバッグ環境（以下実機レス開発環境）を用いる方式が多数提案されている．

例えば，汎用の PC 上で動作するアプリケーション開発用のエミュレータがある．しかし，組込み機器ではアプリケーション層とミドルウェア層が密接に絡んでおり, かつ, ミドルウェア層に製品の重要機能が実装されている場合が多いため，アプリケーション層に加えてミドルウェア層がエミュレータ上で開発できることが求められる．

そこで，ミドルウェア及びアプリケーションの開発に適した，ドライバ層エミュレーションによる実機レス開発方式を提案する．本方式を実際のデジタル TV 製品のソフトウェア開発に適用し，手法の有効性を検証した．

2. 課題とアプローチ

従来の実機レス開発方式とその問題点を表 1 に示す.

(1) ミドルウェア層エミュレーション方式

アプリケーションプラットフォームである BREW[2]や brewmp[3]では, アプリケーション開発用にミドルウェア層エミュレーションが用いられている．実機の機能をミドルウェア層でシミュレートするため，ミドルウェア層の開発

†1 (株)日立製作所　横浜研究所
Hitachi Ltd.　Yokohama Research Laboratory
a) 本稿で述べられたシステムおよび製品名は、一般に各社の商標または登録商標である．

ⓒ2012 Information Processing Society of Japan

には適用できない．このため，ミドルウェア層と密接に絡んだアプリケーション層の開発にも適用困難である．

(2) CPU エミュレーション方式

CPU エミュレータ[4][5][6]は，CPU の動作をエミュレーションすることでミドルウェア層の実機レス開発を実現する．しかし，CPU エミュレータは，CPU の動作を逐一置き換えるため，実行処理が遅くなる．このため，ユーザのリモコン操作後に一定時間が経過した時の機器の反応テストなど，時間に関わるテストには適用困難である．

(3) ドライバ層エミュレーション方式

Android[7][8]などのプラットフォームでは，ドライバ層エミュレーション方式がプラットフォーム設計時から考慮され，プラットフォームと共に提供されている．しかし，このエミュレーション機能を利用するためには，当該プラットフォームを組込み機器に導入する必要がある．このため，過去に開発済みのソフトウェア既存資産を用いた拡張開発に適用する場合，プラットフォーム間でのソフトウェア移植作業が発生し，工数がかかるという問題がある．

表 1 従来実機レス開発方式とその問題点
Table 1 Emulation types and issues

#	実機レス開発方式	問題点
(1)	ミドルウェア層エミュレーション	ミドルウェア層の開発に適用不可のため，ミドルウェア層や, ミドルウェア層と密接に絡んだアプリケーション層の開発には適用困難
(2)	CPU エミュレーション	実行処理が遅く, 操作時間に関わるテストに適用困難
(3)	プラットフォームと共に提供されるドライバ層エミュレーション	当該プラットフォームの導入が必要であり, 既存資産の拡張開発には適用困難

上記問題点を解決するため, 実機のドライバ層およびハードウェア層と同等の機能を, ドライバ層エミュレータと PC のハードウェアを用いて実現する．これにより，本エミュレータ上でミドルウェア層とアプリケーション層のテスト・デバッグが行えるようになる．また，エミュレーションによるオーバーヘッドが小さいため処理遅延が少ない．また，既存資産の拡張開発が可能となる．

一方, 実機と同等機能を持つエミュレータを実装する必要があるため，その確認が課題である．

本稿で提案する実機レス開発方式の要件を以下に示す．

(1) 実機を使わずに PC 上でアプリケーションとミドルウェアの動作確認が可能であること

(a) PC のディスプレイに実機の表示画面，ソフトウェア動作確認用のログ出力，及びソースデバッガを表示

(b) リモコン画像とマウス操作，またはキーボード入力によりリモコンまたはボタン類を操作

(c) キーボード入力により，実機での記録媒体の挿抜を擬似的に実現

実機の応答（表示画面とログ出力）とソースデバッガが一つのディスプレイに統合表示されることで，効率よいテスト・デバッグが可能となる．また，実機におけるリモコン本体ボタン類, 記録媒体の操作を, PC 付属のデバイス（マウスまたはキーボード）で実現することで，テスト・デバッグ時の入力が容易となる．

(2) PC は，従来から開発に利用している Windows/x86 環境をそのまま活用できること

ソフトウェア開発において，仕様書表示やソースコード編集などに用いるために各開発者に配備されている PC の OS は，Windows である場合が多い．実機レス開発環境を構築する際には，この Windows PC をそのまま活用した方が，実機レス開発環境とその他の開発環境の統合や，担当者の習熟期間などの面で効率的である．

(3) 既存資産を流用した拡張開発が可能であること

(4) リモコン操作時間に関するテストが可能であること

3. 既存資産のドライバ層をエミュレートする実機レス開発方式

2 章で示した要件を満たすよう本稿で提案する，実機レス開発方式の特徴を以下に示す．

3.1 Windows と Linux のマルチ OS 環境

ドライバ エミュレータは，アプリケーションとミドルウェアの実機プログラムを PC 上で動かすため，実機のエミュレーション機能を PC 上で提供する環境である．

実機の OS は Linux であるので, エミュレータも OS として Linux を用いれば，実機とエミュレータの OS 部分が共通化できるため，効率的に開発でき，かつ，より正確に動作をシミュレートすることができる．

一方，2 章で示した通り，PC は，従来から開発に利用している Windows/x86 環境をそのまま活用できることが求められる．

そこで，本研究の実機レス開発環境では，Windows PC 上で Linux 実行環境を提供可能な仮想マシン（VMware Player）を利用した．仮想マシンは，Windows アプリケーションとして Windows OS 上で動作しつつ, 内部では Linux 環境を提供することができる．これにより，Windows アプリケーションでソースコード編集などを行なった後，クリック 1 つで仮想マシンに切り替えて，仮想マシンの Linux OS 上でプログラム生成や動作確認を行なうことができる．Windows と Linux のマルチ OS 環境による実機レス開発環境の構成を図 1 に示す．

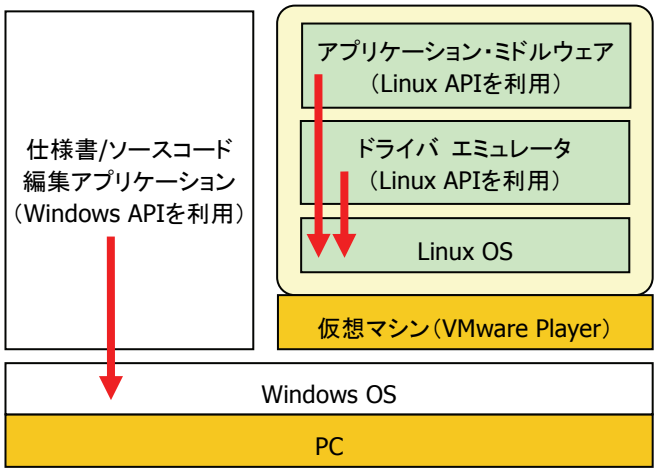

図 1 Windows と Linux のマルチ OS 環境による
実機レス開発環境の構成
Figure 1 Emulator based software development environment
with Windows and Linux

3.2 エミュレータの実装方式

本稿で提案するエミュレータの実装方式について，図 2 に示す．図は，下からハードウェア層（PC），OS 層（Linux カーネル，Linux ライブラリ，Linux デバイス ドライバ，ファイル システム），ドライバ層，アプリケーション・ミドルウェア層を示している．前述のように，実際の実機レス開発環境では Windows OS 上で動作する仮想マシン（VMwarePlayer）の中で Linux OS が動作しているが，ここでは仮想マシンは省略している．また，Linux OS 中の Linux カーネルと Linux デバイス ドライバも省略している．

本稿で提案するエミュレータの各ドライバは，ミドルウェアに対して実機と同じドライバ I/F（API）を提供する．また，内部では固有のエミュレーション処理が実装してあり，これが下層の Linux ライブラリを呼び出す．実機用のソースコードから流用したドライバ I/F と，新たに開発する内部のエミュレーション処理の組み合わせにより，ドライバ エミュレータとして動作させる．以下では，エミュレーション処理，Linux OS，PC ハードウェアの関係について，エミュレーション機能の実装方式ごとに代表例を挙げて述べる．

3.2.1 Linux ライブラリを利用した実装

例えば，ネットワーク ドライバは，上層のミドルウェアから通信設定用の I/F（API）を呼ばれると，同等機能を持つ Linux のソケット ライブラリの通信設定用 API を呼び出し，PC のネットワークデバイスを用いて通信を行う．

実際には，Linux OS と PC ハードウェアの間には仮想マシンが介在しているが，仮想マシン上の Linux 環境は，Windows が存在せず直接 Linux だけが OS として動いている Linux PC と同じ環境であるので，仮想マシンの有無はドライバの処理に影響を与えない．以下の説明では，省略する．

3.2.2 データストレージのエミュレーション機能

IDE（Integrated Drive Electronics）ドライバや Flash ドライバのようにデータストレージ（HDD や FlashROM）を読み書きするドライバの場合は，ストレージをイメージ ファイルとして PC の HDD 上に置き，Linux のファイルシステムを用いて管理する．

例えば Flash ドライバが提供する FlashROM の読み書き機能を，PC の FlashROM を用いてシミュレートすると，PC のシステムデータが書き換えられて PC が動作不良を起こす可能性がある．そこで，本提案方式では，PC の HDD 上に DTV の FlashROM イメージファイル作成して利用する．Linux では，FlashROM などのハードウェアに対する読み書

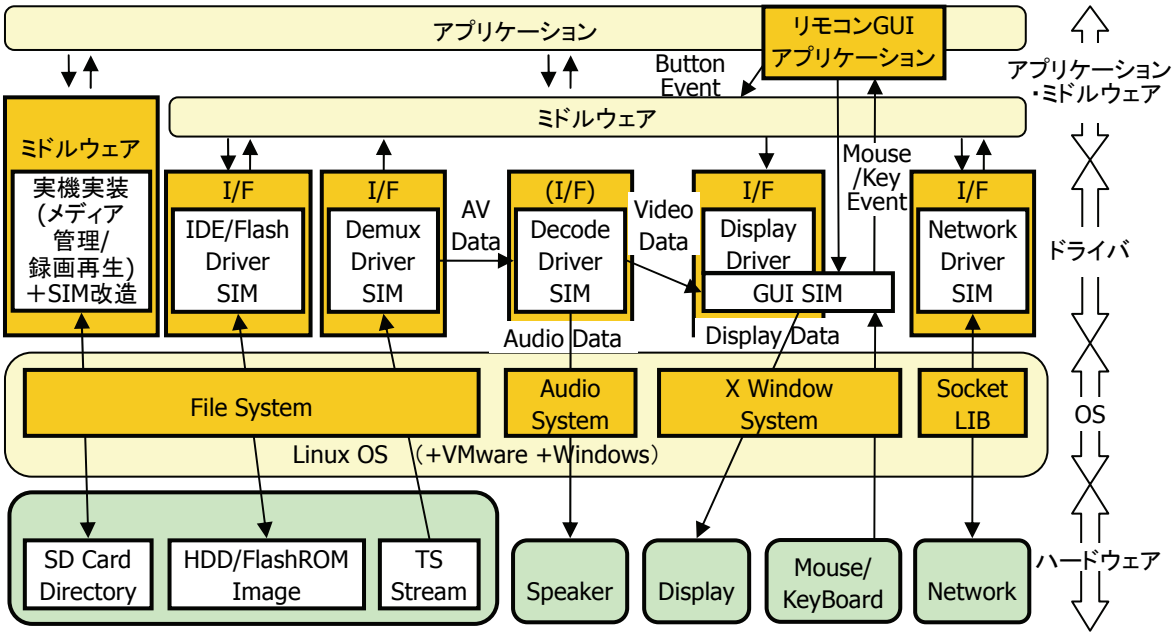

図 2 ドライバ エミュレータの実装方式
Figure 2 Driver emulator architecture

ⓒ2012 Information Processing Society of Japan

き API と，ファイル システムに対する読み書き API が共通化されているため，これを利用して，Flash ドライバ内のシーケンスを変更することなく，エミュレーション機能を実現できる．

3.2.3 GUI エミュレータ

実機では，高速処理用にハードウェアで実装した図形描画機能を表示ドライバが呼び出して利用する一方, PC のハードウェアには図形描画機能が備わっていない場合が多い．このため，本提案方式では，エミュレータ上にソフトウェアで同等機能を実装する．

また，実機では，ユーザの操作端末として赤外線リモコンが用いられる場合があり，PC 上でテスト・デバッグを行うためには, リモコン操作をエミュレートする必要がある．例えば，DTV でリモコンのボタンが押されると，ハードウェアの赤外線受光部がリモコンからの信号を受けて，リモコン ドライバを通じてイベント管理ミドルウェアにボタン イベントを渡す．しかし，PC のハードウェアには赤外線受光機能が備わっていないため, 実機レス開発環境では，PC のマウスまたはキーボードでリモコンを操作する方式とした．

具体的には，Linux の X Window System（以下 X）という GUI 環境構築用の入出力システムを用いて, ディスプレイ，マウス，キーボードなどの GUI を統合管理する GUI エミュレータを開発する．表示ドライバとリモコン GUI アプリケーションは，GUI エミュレータを呼び出して実機の表示画像とリモコン画像を表示する．また，リモコン画像のボタン部分をマウスで操作するかキーボードでキーを押すと，GUI エミュレータが，X 経由で受けたマウス/キーボードのイベントをリモコン GUI アプリケーションに転送し，リモコン GUI アプリケーションが，ボタン部分を押された状態の画像に描画し直して，ボタンイベントをイベント管理ミドルウェアに転送する．

また，実機の表示機能では，制御するハードウェアが異なるため静止画像と動画像で担当するドライバが異なる場合が多い．しかし，PC では表示ハードウェアが 1 つであるため，静止画像と動画像（ビデオ データ）を，両方とも GUI エミュレータが表示する．

3.2.4 ミドルウェア層でのエミュレーション機能

本稿で提案する実機レス開発環境では，基本的にはドライバ層でシミュレートし，アプリケーションとミドルウェアはそのまま実機のソースコードを使えるように設計している．しかし，製品開発適用時のテスト・デバッグ効率の向上が図れることから，SD カードなどの記録媒体の読み書き機能については, ミドルウェア層でシミュレートした．

例えば実機では，記録媒体の挿抜を検出してマウント状態を管理するメディア管理ミドルウェアが，Linux OS が提供する記録媒体ドライバの読み書き API を呼び出す．これに対してエミュレータでは，Linux が管理するファイル システム上の特定ディレクトリを，マウントした記録媒体用ディレクトリに見せかけるようメディア管理ミドルウェアを修正する．これにより，PC 上でファイルをコピーするだけで，記録媒体に様々なデータを入れた状態をテストできるようになる．

また，メディア管理ミドルウェアを修正してマウント状態を擬似的に管理する機能を実装し，キーボードから擬似的にマウント状態を変更してやることで，記録媒体の挿抜テストを簡単にシミュレートできるようにする．

これにより, PC の記録媒体用ハードウェアを用いたドライバ層のエミュレータを実装し，各種データが入った記録媒体を用意して PC に記録媒体を挿抜してテストする場合よりも, 効率良くテスト･デバッグを行なうことができる．

3.3 エミュレータの開発手順

本稿で提案するドライバ エミュレータの開発手順を図 3 に示す．

まず，(1)ドライバの機能仕様書を基にドライバの機能テストを作成し，(2)作成したテストを実機のドライバ上で動かして，テストが正しく動作することを確認する．

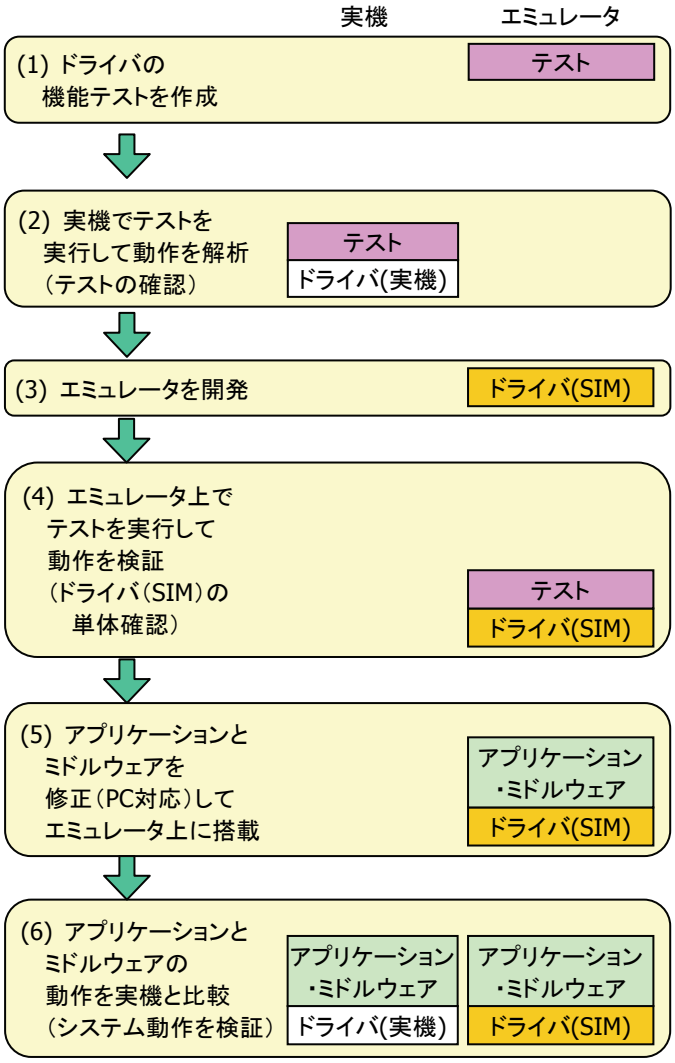

図 3 ドライバ エミュレータの開発手順
Figure 3 Process of driver emulator development

次に，(3)ドライバ エミュレータを開発し，(4)開発したドライバ エミュレータの単体機能を検証する．ここでは，(2)で確認したテストを用いて実機ドライバと同様のテスト結果になることを確認する．(3)は，(1)，(2)と並行開発してもよい．

最後に，(5)アプリケーションとミドルウェアを修正（PC対応：後述）して(4)で開発したドライバ エミュレータ上で動作するようにし，(6)アプリケーションとミドルウェアが実機システムと同じ動作をすることを確認して，システム機能を検証する．

4. デジタル TV 向け実機レス開発環境の開発

3 章で示した実機レス開発方式に従い，デジタル TV 製品のソフトウェア開発用に実機レス開発環境を構築した．

4.1 DTV 向け実機レス開発環境の概要

TV 向けの実機レス開発環境を Windows PC 上に構築した例を図 4 に示す. 図 4 では, 右前面にリモコンと TV 画面, 左背面にソースデバッガのソースコード，ログ画面，変数の値などが表示されている．マウスやキーボードでリモコンを操作し，その反応を TV 画面とログ画面で確認する．また，ソースデバッガを操作し，ソースコードの任意の場所でプログラムを止めて，変数の値を確認しながら 1 行ずつプログラムを動作させることなどができる．

図には示していないが，同じ PC（Windows）上で，従来通り仕様書を表示したり，使い慣れたテキスト編集ツールでソースコードを編集することも可能である．

TV 向け実機レス開発環境で動作確認可能な主な TV 機能を以下に示す．

・TV 放送（表示，選局，各種設定）
・番組情報（番組表，番組検索，録画予約など）
・ネットワーク（映像機器との連携）
・HDD（録画，一覧表示，編集，再生），SD カード（写真表示）

4.2 アプリケーションとミドルウェア内の実機依存処理の対策

本稿で提案する実機レス開発方式では，ハードウェアとドライバをエミュレータ実装し，その上でアプリケーションとミドルウェアを開発できるようにする．実機レス開発環境のハードウェア（PC）と実機のハードウェア（組込み機器）の違いは基本的にエミュレータが吸収するが，開発対象であるアプリケーションやミドルウェアに実機依存処理が含まれている場合，開発対象にも対策が必要となる．本稿の提案方式を DTV に適用した際の，主な実機依存項目，その洗い出し方法と対策を表 2 に示す．

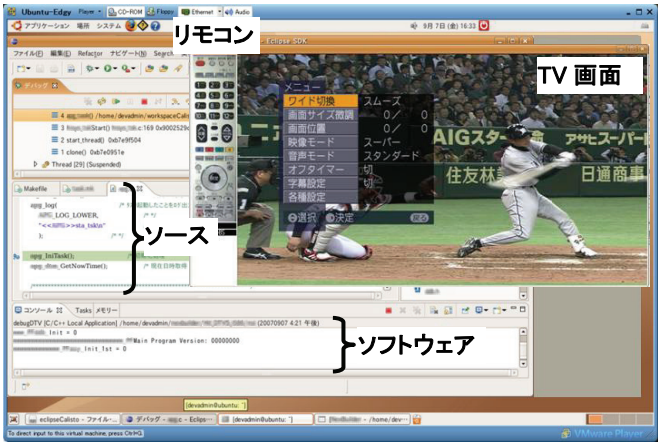

図 4 実機レス開発環境の画面例
Figure 4 Sample picture of software development environment

表 2 実機依存処理の洗い出し方法とその対策
Table 2 How to find and solve platform dependency

#	実機依存内容	洗い出し方法	対策（ソースコード修正）
(1)	固定アドレス利用	8 桁の 16 進数値でソースコードを検索（マスクも該当するが目視で除外）	アドレス管理モジュールでエミュレータ対応（malloc 利用）
			レジスタ アクセス等の処理を個別に削除
(2)	エンディアン依存	シフト演算，2 バイトコードでソースコードを検索	ソースコード上に両方（ビッグ・リトル）の場合を記述し#define でビルド時に切替
(3)	アセンブラコード	ビルドエラーを参照	実機用と PC 用の両方のアセンブラ実装ファイルを用意し Makefile でビルド時に切替

(1) 固定アドレス利用

実機ではメモリアドレスは固定の設定だが，エミュレータではメモリアドレスが動的に設定され，起動の度にアドレスが変化する．このため，アドレス管理モジュールのソースコード中で固定アドレスを用いている部分を，malloc を利用した動的なアドレス指定方法に変更した．

また，レジスタやバスへのアクセス処理にも固定アドレスが利用されていたが，これらの処理は実機依存であり実機レス開発環境では利用できない．システム起動時のこれらのアクセス処理はハードウェアの初期化指示であり，実機レス開発環境上では不要であるため，ソースコードから削除した．また，レジスタを一時的なデータ保存場所として利用している場合は，エミュレータ内に代わりのデータ保存場所を確保した．

固定アドレスを利用している箇所は，8 桁の 16 進数値をソースコードから検索して洗い出した．この時，マスク処理なども該当してしまうが，目視で判断して除外した．

(2) エンディアン依存

実機環境と実機レス開発環境とで，エンディアンが異な

るため，実機用ソースコード中のエンディアン依存の部分に，実機レス開発用エンディアンに対応させた実装を併記し，#define でビルド時に切り替えることで対応した．

エンディアン依存処理は，シフト演算や 2 バイトコードをソースコードから検索して洗い出した．

(3) アセンブラ コード

実機環境と実機レス開発環境では CPU が異なり，実機用のアセンブラ コードはそのままでは実機レス開発環境で動かない．このため，実機と実機レス開発環境の両方の CPU 用のアセンブラ実装ファイルを用意し，Makefile でビルド時に切り替えて対応した．

対応していない CPU のアセンブラ コードは，ビルド時にコンパイル エラーが出るため，エラー メッセージを参照して洗い出した．

5. デジタルテレビ開発への適用評価

4 章で開発した DTV 向け実機レス開発環境を，毎回既存資産に対する拡張開発を行っている，実際の DTV 製品のソフトウェア開発に適用し，評価した．

5.1 製品開発での適用対象

DTV の約半数のアプリケーション開発における，Unit Test（以下 UT），Combination Test（以下 CT）及びデバッグに DTV 向け実機レス開発環境を適用して評価した．アプリケーションの CT では，ユーザによるリモコン操作に応じた，アプリケーションと，それが呼び出すミドルウェアの組み合わせによる応答を確認する．アプリケーション（APP）全体に対する適用対象分の規模を表 3 に示す．

表 3 適用対象ソフトウェアの規模
Table 3 Software size

	APP 全体		
		適用対象	適用比
モジュール数	68	33	49%
完成ステップ数	688k	187k	27%

5.2 適用手順

5.2.1 UT

UT では，関数単位での動作を確認するため，ミドルウェアやドライバなどとの関数間での呼び出しは考慮せず，関数の入力や内部変数などの条件に応じて，アプリケーションの各関数内で正しい経路が実行され正しい値が出力されることを確認する．このため，実機レス開発環境を適用し，デバッグ用ツールまたはログ出力を用いて，経路や値の確認を行った．

5.2.2 CT

CT では，以下の手順のように，実機レス開発環境における制約を詳細に調査した後，実際のテストを行なった．

(a)テスト対象であるアプリケーションをエミュレータ上で実行し，アプリケーションの実機用 PCL（Program Check List）に従い，1 項目ずつテスト（動作確認）の可否を調査

(b) (A)エミュレータ上でテストできる項目，(B)スタブ作成によりエミュレータ上でテストできる項目，(C)実機を使わないとテストできない項目に分類

(c) 必要なスタブを作成

(d) (A)(B)をエミュレータ上でテスト

(e) (C)を実機上でテスト

具体的には以下の機能について実機テストを実施した．

・エミュレータが対応していない機能
・実際の TV 放送の受信が必要な機能
・画質・音質調整など

5.3 評価項目・評価方法

実機レス開発環境の適用評価について，評価項目と評価方法を表 4 に示す．

表 4 実機レス開発環境の評価項目と評価方法
Table 4 Evaluation items

#	評価項目	評価方法
(1)	適用可能テスト件数	実機レス開発環境のみで（実機がなくても）確認可能なテスト項目の件数を計上
(2)	テスト・デバッグ期間（実機準備完了後の開発期間）	実機レス開発環境と実機の両方を用いた場合（実績値）と，実機レス開発環境の適用がない場合（実機のみの利用を仮定して算出）で，実機準備完了後の開発期間を比較
(3)	テスト・デバッグ効率（工数当たりのテスト件数）	実機レス開発環境で実施したテストと，実機で実施したテストで，工数当たりのテスト件数を算出して比較

5.4 評価結果

5.4.1 適用可能テスト件数

UT と CT のそれぞれについて，今回の適用対象ソフトウェアでのテスト件数，その内実際に適用可能であったテスト件数，及び(c)適用対象ソフトのテストにおける適用比率（=b/a）を表 5 に示す．

表 5 実機レス開発環境の適用可能テスト件数
Table 5 Applicable tests

	(a)適用対象		
		(b) 実際に適用可能分	(c)適用対象ソフトのテストにおける適用比率（=b/a）
UT 件数	12015	12015	100%
CT 件数	15174	6100	40%

5.4.2 テスト・デバッグ期間

適用対象開発のテスト・デバッグでは，実機は，テスターとプログラマー全員で 1 台を共有し，実機レス開発環境は，テスターとプログラマーのそれぞれに 1 人 1 台必要な台数だけ配備した．

実機レス開発環境と実機での UT および CT の実績値から，実機レス開発環境の適用がない場合，すなわち実機 1 台のみでテスト・デバッグした場合のテスト・デバッグ期間を，下記の手順で推定した．

(a) (A)テスト・デバッグ工数，(B)月当たりの作業時間，(D)テスト・デバッグ期間の実績値から，(C)月当たりのテスト・デバッグ工数を算出（C=(A/B)/D）．

(b) (A)の実績値を合計して，実機レス開発環境を適用しない場合の(A)を算出．

(c) 実機レス開発環境を適用しない場合の(B)と(C)は，実機における(B)と(C)の実績値と同じだと推定して設定．

表 6 テスト・デバッグ期間の短縮効果
Table 6 Effect on testing and debugging duration

		適用時の実績値				適用なしを仮定
		実機レス		実機		
	テスト環境	必要な台数		実機 1 台		実機 1 台
	テスト分類	UT	CT	UT	CT	UT と CT
A	テスト・デバッグ工数（人・時間）	1100	800	0	1440	3340 (*2)
B	月当たりの作業時間（時間／月）	200	200	-	240	240 (*3)
C	月当たりのテスト・デバッグ工数＝実質作業者数（人）	1.3 (*1)	1.6 (*1)	-	1.3 (*1)	1.3 (*3)
D	テスト・デバッグ期間（ヶ月間）	4.2	2.5	0	4.5	10.7 (*4)
適用効果					42%に開発期間を短期化	
					6.2 ヶ月間の短縮	

*1) 実績値 A, B, D より算出，*2) 実績値 A を合計して算出，*3) 実機の実績値と同じだと推定して利用，*4) 実機レス適用なしの A, B, C より算出

(d) 実機レス開発環境を適用しない場合の(A)，(B)，(C)から，実機レス開発環境を適用しない場合の(D)を算出（D=(A/B)/C）．

以上の結果を表 6 に示す. 実機レス開発環境の導入により，実機を用いたテスト・デバッグの期間（実機準備完了後の開発期間）を 6.2 ヶ月間短縮（42%に削減）することが出来たと考えられる．

5.4.3 テスト・デバッグ効率

実機レス開発環境と実機開発環境で, UT および CT それぞれにおけるテスト・デバッグ効率（工数当たりのテスト件数）を算出して比較した．(A)テスト・デバッグ工数は，(2)テスト・デバッグ期間の評価に用いた値と同じ，適用対象部分の開発における実績値であり，同じく実績値である(E)テスト件数と合わせて(F)テスト・デバッグ効率を算出した（F=E/A）．ただし，適用対象部分の開発では UT100%に実機レス開発環境を適用したためテスト・デバッグ効率が求められないので，参考データとして，ソフトウェア開発全体でのテスト・デバッグ効率を記した（表 7）．

表 7 テスト・デバッグ効率の向上効果
Table 7 Effect on testing and debugging efficiency

		適用対象部分の開発での実績値			
		実機レス		実機	
	テスト環境	必要な台数		実機 1 台	
	テスト分類	UT	CT	UT	CT
E	テスト件数（件）	12015	6100	0	8481
A	テスト･デバッグ工数（人・時間）	1100	800	0	1440
F	テスト･デバッグ効率（件／人・時間）＝工数当たりのテスト件数	10.9 (*1)	7.6 (*1)	(10.2) (*2)	5.9 (*1)
適用効果（実機レス/実機）		(107%)	129%		

*1) 実績値 E, A より算出，*2) 実際の開発データがないため，ソフトウェア開発全体での参考値

5.5 考察

5.5.1 適用可能テスト件数

本適用評価では，UT の 100%，CT の 40%に実機レス開発環境を適用可能であった．UT において 100%適用可能となったのは，本適用対象がアプリケーションであり，CPU レジスタやバスなどの制約事項に影響されなかったためである．一方，CT は，スタブを作成することで非実装機能のテストを可能にしたが，実際の放送電波を受信しないとテストできないなどの制約の影響を受け適用率が 40%に留まった. リモコン操作時間に関するテストは可能であった．

5.5.2 テスト・デバッグ期間

表 6 の結果から，実機レス開発環境がなく実機 1 台でテスト・デバッグを実施した場合は，実機レス開発環境を適用した場合に比べ，テスト・デバッグの終了が 6.2 ヶ月間遅延すると考えることができる．実機レス開発環境の適用により，実機準備が完了する前からテスト・デバッグの前倒し着手が可能になったことによる開発期間短縮の効果である．これにより，実機レス開発環境によるハードウェアとソフトウェアの並行開発の効果を確認することができた．

5.5.3 テスト・デバッグ効率

実機レス開発環境で実施した CT は，実機開発環境で実施した CT に対して，テスト・デバッグ効率が 129%に向上している．一方，UT の場合は，参考値との比較のため単純には評価できないが，107%とほぼ同等である．これは，テスト・デバッグ工数に占めるデバッグ工数の割合が UT では小さく CT では大きいため，実機レス開発環境の適用によるデバッグ効率向上の効果が，CT に顕著に現れていると考えられる．

5.6 実機レス開発環境のその他の活用事例

開発した DTV 向け実機レス開発環境は，本適用評価対象でのテスト・デバッグ以外にも適用した．その他の活用事例の代表例とその事例で明らかになった実機レス開発環境の効果について表 8 に示す．

表 8 その他の活用事例

Table 8 Other merits

#	適用対象	適用効果
(1)	録画再生機能, ネットワーク通信機能	実機が確保できる前の開発序盤に適用し，ソフトウェアの初期検討と，一部テスト・デバッグの前倒し/並行実施に貢献した．
(2)	SD カード読み書き機能	SD カード読み書き機能に関連するアプリケーションにおいて，テスト・デバッグに適用した. 各種データが入った SD カードを実機で挿抜してテストする場合よりも効率向上した．
(3)	実機と離れた環境での開発	開発者用 PC と実機が離れていた環境に適用．初期検討，コーディング，一部テスト・デバッグに適用し，同じ PC 上でコーディングと動作確認を行うことで, これらを頻繁に繰り返す開発序盤の開発効率を向上した．
(4)	試作開発, 初心者の環境習熟用	実機は製品開発を優先して配備される．新技術を検討しているグループ，および初めて DTV 開発に携わる開発者の多いグループに適用して実機確保の問題を解消し, 効率の良い試作開発，環境習熟を実現した．

6. おわりに

テスト・デバッグ効率の向上と開発期間の短縮を目的に，実機レス開発方式を提案し，既存資産の拡張開発を行う DTV 製品向けのドライバ層エミュレータを開発した. アプリケーション開発に適用し，ミドルウェアと組み合わせたアプリケーションのテスト・デバッグにおける効果を確認した．リモコン操作時間に関するテストが可能であることを確認した. 今後は, 既存資産の拡張開発を行う他の Linux 搭載製品へも適用を図る．

参考文献

1) 田丸喜一郎: 組込みソフトウェア産業の現状と課題, http://sec.ipa.go.jp/events/2012/esec_02/events_esec_20120510-13.pdf, pp.2 (2012).
2) Qualcomm, BREW, http://brew.qualcomm.com/
3) brewmp, http://www.brewmp.com/
4) 松田稔彦, 北村俊明: 計算精度低下を検出する PC エミュレータの開発, 情報処理学会研究報告, Vol.2011-HPC-132 No.24, pp.2-3 (2011).
5) F. Bellard: QEMU, a Fast and Portable Dynamic Translator, Proc. USENIX 2005 Annual Technical Conference, pp. 41-46 (2005).
6) F. M. David, E. M. Chan, J. C. Carlyle, and R. H. Campbell: QInject: A Virtual Machine based Fault Injection Framework, In International Conference on Architectural Support for Programming Languages and Operating Systems (2008).
7) Android Emulator, http://developer.android.com/tools/help/emulator.html
8) A Developer's First Look At Android, www.linuxforu.com, LINUX FOR YOU, JANUARY 2008, pp. 48-50 (2008).

ⓒ2012 Information Processing Society of Japan