

Hacking the Xbox 日本語訳

目次

読者のみなさんへ	6
謝辞	7
プロローグ：README.1ST	8
ビデオゲームコンソール市場	9
ハッカーとハッキングについて	11
ハッキングの政治学	13
ハックの裏にいる人々	17
注	20
第1章 保証を無効にする	20
道具の選択	20
カバーを開けるための道具	21
コンポーネントを取り付け・取り外しする道具	22
テストと診断のための道具	23
設計のための道具	25
Xbox を分解する	26
ステップ1：安全確認	26
ステップ2：ケースのネジを外す	27
ステップ3：上部カバーを外す	29
ステップ4：ディスクドライブを移動する	31
ステップ5：ディスクドライブを取り外す（オプション）	34
Xbox の再組み立て	35
第2章 箱の中で考える	37
回路基板を読む	37
回路基板の基礎	37
部品	39
テストポイント	43
Xbox アーキテクチャ	44
高レベルの構成	44
機能の詳細	46
CPU	47
ノースブリッジとサウスブリッジ	49
RAM	50
ROM	50
その他	51

パターンマッチング	52
比較：Xbox 対 PC	52
対比：Xbox 対ゲームキューブ	54
注	56
第 3 章 青色 LED を取り付ける	56
必要なもの	57
フロントパネルの取り外し	57
フロントパネル回路基板の取り外し	62
青色 LED の取り付け	64
フロントパネルの再組み立て	68
デバッグ	70
第 4 章 USB アダプタを作る	71
用意するもの	72
方針	73
実装	74
第 5 章 壊れた電源を交換する	78
壊れた電源の診断	79
電源の交換	80
方針	81
手順	81
Xbox 電源ケーブルの製作	81
交換電源の取り付け	89
交換電源での起動	91
デバッグのヒント	92
第 6 章 最高の Xbox ゲーム：セキュリティハッキング	92
妄想的な設計との最初の出会	93
ROM を読み出す	93
Microsoft との邂逅	96
ROM の中身を解析する	97
注	105
第 7 章 セキュリティ入門	106
セキュリティって本当に必要なの？	106
暗号技術入門	109
暗号アルゴリズムの種類	109
SHA-1 ハッシュ	112
TEA	112
RC-4	115
RSA	117
セキュリティ全体像	120
注	121

第 8 章 Xbox セキュリティのリバースエンジニアリング	121
ハードウェアから秘密情報を取り出す	121
高速バスの盗み聞き	123
予算内でのバスタップ	123
データロガーの構築	133
バスの順序と極性の特定	137
キャプチャデータの解読	138
第 9 章 裏口から忍び込む	142
バックドアとセキュリティホール	143
ヴァイザーのジャムテーブル攻撃	144
MIST プリマチュア・アンマップ攻撃 1	145
Microsoft の反撃	145
Xbox セキュリティバージョン 1.1 のリバースエンジニアリング 3	146
バックドアの脅威	151
注	153
第 10 章 さらにハードウェアプロジェクト	153
LPC インターフェース	153
Xbox の LPC インターフェース	154
LPC インターフェースの活用	155
もう 64MB の SDRAM	158
Xbox VGA	160
大容量ストレージの交換	161
注	162
第 11 章 Xbox 向けソフトウェア開発	162
Xbox-Linux	162
Xbox-Linux のインストール	163
「プロジェクト B」	166
OpenXDK	172
第 12 章 注意せよ、ハッカーよ	172
知的財産法の概観：古典的枠組み	174
著作権	174
特許	176
営業秘密	176
憲法上の著作権の取引	177
リバースエンジニアリングの伝統的な見解	177
営業秘密と「不正な手段」	178
著作権法と中間コピーの問題	178
特許法	179
リバースエンジニアリングの新たな課題	179
デジタルミレニアム著作権法（DMCA）と不正アクセスの問題	180

不正アクセス	180
回避技術	181
DMCA の免除規定を利用する	181
エンドユーザーライセンス契約とリバースエンジニアリングの契約上の禁止	183
営業秘密と経済スパイ法	184
責任あるハッカー：無知は弁護にならない	184
民事および刑事上の罪と罰則	184
リバースエンジニアリングとしての「いじる自由」とその他の法的問題	185
注	186
第 13 章 前進！	188
コミュニティへの貢献	188
トラステッドコンピューティング	191
一歩引いて考える	192
Palladium と TCPA の比較	194
トラステッド PC をハックする	195
展望	197
おわりに	198
注	198
付録 A ハッキング機器の入手先	198
ホビイスト向けベンダー	199
機材注文フォーム	200
基本注文フォーム	201
上級注文フォーム	201
付録 B はんだ付け技術	202
はんだ付けの基礎	202
フラックスを使う	203
最初のコツ	204
表面実装はんだ付け	205
単純な部品の手順	205
複雑な部品の手順	207
部品の取り外し手順	212
付録 C PCB レイアウト入門	215
PCB レイアウト	216
一般的な配置・配線のガイドライン	217
表面実装デバイスのビアファンアウト領域を確保する	217
デカップリングコンデンサを SMD ランドの下に配置する	218
特殊な配線を把握する	219
回路基板はヒートシンクにもなる	221
各レイヤーの優先ルーティング方向を決める	221
直交レイヤーで基板をスタックする	221

2 層基板ではフィンガーを使って電源をバス配線する	222
オートルーターの使い方のヒント	222
CAD ツール	222
基板製造会社	223
Sierra Proto Express	223
Data Circuit Systems	223
Advanced Circuits	224
Alberta Printed Circuits	224
スターターポイント：ATX 電源交換アダプタ基板	224
付録 D FPGA 入門	225
FPGA とは何か	226
FPGA の設計	228
プロジェクトのアイデア	233
購入先	234
付録 E デバッグ：ヒントとコツ	235
よくあるバグ	236
剥がれた配線やランドからの復旧	238
付録 F Xbox ハードウェアリファレンス	244
電源コネクタのピン配置	244
ハードディスク電源コネクタのピン配置	246
ビデオコネクタのピン配置	246
USB コネクタのピン配置	249
ゲームコントローラー拡張スロットのピン配置	251
イーサネットコネクタのピン配置	253
ATA コネクタのピン配置	253
DVD-ROM 電源コネクタ	254
LPC コネクタ	256
ファンコネクタ	257
フロントパネルコネクタ	257
ライセンスとクレジット	258
コンテンツライセンス	258
原著ライセンス	258
本サイトのコンテンツ	258
CC BY-NC-SA 1.0 の要点	258
再利用について	258
免責	259
クレジット・謝辞	259
原著	259
翻訳プロジェクト	259
免責事項	259

読者のみなさんへ

本書をダウンロードして読んでくれてありがとう。No Starch Press と僕は、Aaron Swartz を追悼して、この『Hacking the Xbox』の無料電子書籍版を公開することにした。本書を読む中で、自由がハッキングコミュニティにとっていかに重要かを改めて感じ、Aaron が信じた大義を支持する気持ちになってもらえたら嬉しい。

本書を無料で公開することに同意したのは、MIT による Aaron への扱いが、僕にとって他人事ではないからでもある。本書には、僕が MIT 大学院生だった頃に初代 Microsoft Xbox のセキュリティキーを抽出した話が収められている。MIT の法務部から僕の研究との関係を否定する書状が届いたときの打ちのめされた気持ちや、Microsoft と一人で向き合わざるを得なくなった経緯も読めるはずだ。

違ったのは、僕が所属する人工知能研究所の教員たちがこの扱いに激怒したことだ。彼らは MIT 法務部に公然と反発し、僕の研究を正式な「AI Lab メモ」として発表することを約束した。これにより、Microsoft との交渉で大きな足場を得ることができた。Microsoft は、正当な学術研究者を訴えることへの世論の反発を意識し、この問題について穏当な和解に達した。

「人民の、人民による、人民のための」政府であるはずのアメリカが、一企業よりも同胞への思いやりと理性を持てないことが悲しい。その後も別の団体から法的な脅迫を受けた経験から、高額な訴訟がいかに醜く、内臓をえぐられるような体験であるかを嫌というほど知っている。幸いにも、僕の場合はリスクがそこまで高くなく、相手も Aaron ほど強大ではなかった。そうでなければ、僕も絶望と恐怖に屈していたかもしれない。数年前、海外で生活を立て直し始めた。海外に住んでいることで訴状を受け取りにくくなるという事実が、わずかな慰めになっている。

アメリカの法体系は公正さを目指してはいるが、そのルールは資金力を持つ者に有利な非対称な戦いを生み出す。力の弱い側に、正当かどうかにかかわらず決着を迫る最も有効な手段の一つは、裁判前の駆け引きによって相手のリソースと戦意をじわじわと削り取ることだ。人生のすべてが電子顕微鏡の下に置かれたような感覚になる——些細な傷が申立・反論・証拠開示・召喚状・宣誓供述書という激しい応酬に拡大され、一つひとつの行為が何万ドルもの法的費用として積み重なっていく。友人、同僚、雇用主、家族までもが証人としてこの屈辱の舞台に引きずり込まれる。さらに悪いことに、誰かに率直に話すことさえ、その人が証人として召喚されるリスクがあるとして控えるよう忠告される。孤立し、恐怖にさらされたとき、資金力のある相手に技術的な問題で負けるリスクを冒すより、公正さにかかわらず折れて和解する方が得策に思えてしまう。

アメリカ政府は世界で群を抜いて資金力があり、恐るべき敵だ。著作権法には、残酷とも言える異常に重いペナルティが設けられている。Aaron とは面識がなかったが、彼が受けた仕打ちの深刻さが、彼の最後の決断に反映されていると思う。

Larry Lessig が言う「アメリカの法体系には羞恥心が必要だ」という考えに、僕は全面的に同意する。外からの目には、アメリカ政府の一部の検察官が、追い求める個人を犠牲にして名声を得ることに執着しているように映る。訴訟に勝つことで昇進や、より注目度の

高い案件への抜擢に必要な評判と信頼を得られるからだ。彼らにとって、問題は正義ではない——勝利と自己宣伝だ。

こうしたインセンティブ構造が、勇気を持って立ち上がり何か大胆なことをしようとする個人や小さな組織への恥知らずないじめを助長している。起訴されること自体が判決と同等の罰になりうるため、個人は自分が正しいと信じることのために戦う意志と力を奪われてしまう。その結果、市民的不服従の時代が終わりに近づいているのではないかと、僕は恐れている。

人間として、個人として、ハッカーとして、僕たちはこの流れに抗い、心の底から正しいと感じることをし続けなければならない。Aaron の物語は悲劇で終わったが、本書の中に、読者が励まされるようなハッピーエンドの物語を見つけてもらえることを願っている。いじって探求する権利がなければ、僕たちはテクノロジーに奴隷化される危険がある。そして、ハックする権利を行使すればするほど、その権利を奪うことはより難しくなる。

bunnie

シンガポール、2013 年 3 月

謝辞

献身的で愛情深い両親に感謝する。今の僕を育ててくれた。

オンラインハッキングコミュニティのアドバイスと指導にも感謝したい。特に、政府による迫害や雇用主からの報復を恐れて匿名で活動せざるを得ない人たちに。

Electronic Frontier Foundation の Lee Tien、Boston College Law School の Joseph Liu、そして MIT 人工知能研究所の Tom Knight 博士と Hal Abelson 教授には、Xbox セキュリティシステムに関する最初の論文を発表するプロセスを通じて助けてくれたことに特別な感謝を贈りたい。彼らの支援と助言なしには、発表できなかっただろう。

Xbox-Linux チームにも感謝している : Michael Steil、Milosch Meriac、Franz Lehner (詳細なテクニカルレビューをありがとう!)、そして素晴らしい Andy Green (別名 numbnut) ——最新の Xbox ハックへの深い洞察と、本書のための面白い素材を提供してくれた。みんな、本当に最高だ。これからも頑張ってもらいたい。XboxHacker.net BBS の創設者 Dan Johnson (別名 SiliconIce) にも感謝したい。BBS を立ち上げ、本書への興味深い素材を提供し、有益なテクニカルレビュー、アドバイス、励ましをくれた。また、Xbox-Linux チームの写真を提供してくれた Gerhard Farfeleder にも感謝する。

Via Technologies 社の Timothy Chen には、Xbox と PC の比較用に P4M266 マザーボードを提供してくれたことと、PC 業界への興味深い洞察に感謝する。また、Xilinx University Program を通じた寛大な FPGA 提供について、Xilinx にも感謝したい。

あなたたちは自分が誰で、どう助けてくれたかを知っている xor、adq、luc、head、visor、roastbeef、kgasper、xerox、lordvictory、pixel8、El (GCN)、tom from HK、そして sween (Scotch!)

本書の無制限版についても感謝を伝えたい人がいる。No Starch Press の Bill Pollock には、本書の印刷と流通を大胆に引き受けてくれたことに特別な感謝を。Paul Yoon には、数多くの誤植を修正してくれたことに心からの感謝を。そして、O'Reilly & Associates 社の Rael Dornfast と Tim O'Reilly には、有益なアドバイスと励ましに特別な感謝を贈りたい。

プロローグ：README.1ST

Microsoft の Xbox™ ビデオゲームコンソールは、単に最新ゲームが遊べるだけでなく、それ自体として魅力的なハードウェアだ。安価でパワフルな Xbox は、PC として、オールインワンのメディアプレーヤーとして、あるいはウェブサーバとして使える可能性を秘めている。しかし残念なことに、Xbox のような現代の電子ハードウェアを探求・改造する方法を教える本はほとんどない。電気工学の教科書はたいてい理論寄りで専門分野に集中しすぎており、実際のハッキング (hacking) が必要とする幅広い実践的スキルとはかけ離れている。また、子どもの頃に僕のインスピレーションとなったハードウェアハッキング (hardware hacking) の実践書は、技術の急速な進化によってとっくに時代遅れになってしまった。本書は、現代のコンピュータを理解しリバースエンジニアリング (reverse engineering) するための実践ガイド——新世代のハッカー (hacker) のためのハンドブック——というニーズを満たすことを目的としている。

Xbox をハックする最大の価値は、その教育的効果にある。「魚を与えれば一日食えるが、釣りを教えれば一生食える」という言葉の通りだ。そのため本書は、はんだ付け (soldering)、リバースエンジニアリング、デバッグ (debugging) といった基本的なハッキング技術を初心者ハッカーに紹介しつつ、経験豊富なハッカーにも役立つハードウェアの知識と洞察を提供することを目指している。

Xbox が注目されたのは、セキュリティ設計の優れた手本だったからではない。Microsoft の会長が「セキュリティ」を同社の最優先課題として打ち出した直後に、まさにその Microsoft が世に送り出した——知名度が高く、大量に出回る——製品だったからこそ、Xbox はセキュリティコミュニティとハッキングコミュニティの双方にとって格好の教材となった。¹

Xbox の経験は、ユーザーの手元という「敵対的な」環境で信頼できるクライアントを構築することがいかに難しいかを示している。たとえ大企業であっても、資金が潤沢であっても、それは容易ではない。このことは一つの示唆を与える。安価なハードウェアクライアントに委ねられる秘密情報の重要度には、おのずと上限がある。安く大量に配布される機器を完全に守ることは難しい。だからこそ、その内部に置く秘密鍵や保護情報も、「破られたとしても許容できる範囲」に抑えるべきなのだ。

さらに Xbox は、執筆時点で約 1,000 万台のほぼ同一の個体が存在しており、一貫した教材として最適だ。その構成も一般的な PC に近いため、本書の議論の多くは PC というより広い対象にも直接応用できる。これも Xbox ハッキングの教育的価値を高める要因だ。

Xbox ハッキングのもう一つの興味深い側面は、Xbox を追いかけるハードウェアハッカーたちのアンダーグラウンドなコミュニティの存在だ。Xbox をハックした人々と彼らが積み上げた専門知識は、Xbox がほこりをかぶったフリマの売れ残りになった後も長く価値を持ち続けるだろう。だから本書は、意識的に社会的な側面にも目を向けている。さまざまな Xbox ハッカーのプロフィールを収録したのは、ロールモデルを通じて、ドライバーとはんだごてを手に取りハッキングを始めようという気持ちに人々を駆り立てたいからだ。そうした探求の精神を若い世代に植えつけることは、技術革命を牽引してきた有能なエンジニアの層を維持するために、長期的に見て重要なことだ。今日のエンジニアの多くは、当時は完全な回路図とソースコードが同梱されていたアマチュア無線機、電話機、コンピュータをいじり倒すところから出発した。こうした技術者の人材基盤は、健全な経済と、コンピュータ時代における強固な国家安全保障を維持するために不可欠だ。

ビデオゲームコンソール市場

2002 年は、国内外ともに波乱の年だった。テクノロジー市場では PC 販売が横ばいになり、サーバビジネスは縮小し、一部の例外を除いて通信市場の見通しは暗かった。そんな低迷の中であって、ビデオゲームのハードウェア・ソフトウェア・アクセサリ市場は記録的な年を達成した——2001 年比 10% 増の 103 億ドルという数字だ。2 これは 2001 年の米国音楽産業の売上 130 億ドルに匹敵する。

ビデオゲーム市場の規模は大きいですが、コンソールビジネスで利益を出すことは容易ではない。ゲームの顧客は好みがうるさく、トレンドに敏感で、財布の紐は固い。彼らは高性能でかっこいいコンソールハードウェアを、豪華な家族ディナーや医者診察代程度の値段で求める。こうした「高性能・低価格」への要求が、コンソールメーカーにハードウェアを赤字で販売することを強いる。その結果、コンソールメーカーは「クローズドコンソール (closed-console)」という戦略をとる。コンソールは赤字覚悟の目玉商品として売り出し、利益は将来のゲームタイトル販売で回収するという仕組みだ。この戦略はコンソールハードウェアと広告への多額の先行投資を必要とする。そしてコンソールメーカーには、ゲーム開発者が安心してプラットフォームに時間と資金を投資できるよう、自社ハードウェアの市場を作り上げる責任がある。

「鶏と卵」のジレンマがある。ゲームタイトルが少ないコンソールは誰も買おうとしない。だから、何百万台というハードウェアを製造・展開するリスクと、ハードウェアで発生する数億ドルもの先行損失は、ほぼすべてコンソールメーカーが背負うことになる。その結果、現在コンソールゲームビジネスで存在感があるのは 3 社だけだ——ソニー、任天堂、そしてマイクロソフト。ソニーはコンソール市場でダントツのリードを持ち、任天堂はゲームボーイシリーズでハンドヘルド市場を押さえている。マイクロソフトはゲームコンソール市場への新参者だ。2 位争いはまだ決着がついていない。2003 年初頭の時点では、日本とヨーロッパではゲームキューブの販売台数が Xbox を上回る一方、北アメリカという大市場では Xbox がゲームキューブをリードしていた。

クローズドコンソールビジネスモデルの肝は、消費者をロイヤリティが発生する承認済みゲームタイトルだけを買う状態に縛り込むことだ。つまり、海賊版や未承認ゲームタイト

ルはビジネスの収益性を根底から脅かしうる。だからコンソールには、ゲームコピーや未承認ゲームの開発・流通を阻むセキュリティ機構が不可欠だ。セガ・ドリームキャストの失敗は、セキュリティ機構が破れたとき何が起るかを如実に示した事例だ。

ドリームキャストは1998年11月に日本で発売された。グラフィックスアクセラレータである NEC PowerVR2 DC チップの製造問題が初期出荷を制限した。その後の3年間はドリームキャストにとってジェットコースターのような時期だった。ソウルキャリバー、デッドオアアライブ2、バイオハザード、クレイジータクシー、シェンムーといった人気ゲームがドリームキャストを支え続ける一方で、ソニーのプレイステーション2の登場がドリームキャストの販売を侵食し、最終的にはソフトウェア開発者の信頼も奪っていった。皮肉なことに、プレイステーション2の高い処理能力にもかかわらず、デッドオアアライブ2のような初期のプレイステーション2タイトルと比べても、ドリームキャストのグラフィックス品質は同等かそれ以上だった。(プレイステーション2はプログラムが難しく、開発者がその潜在能力を引き出すまでに数年かかった。)

ドリームキャストの棺に最後の釘が打ち込まれたのは、2000年の春から夏にかけてのことだ。ドイツのハッカーグループ「チームユートピア」が、ドリームキャストのマスク ROM BIOS の中に、標準 CDROM からの起動を可能にするバックドアを発見した。本来、ドリームキャストはゲーム配布に「GD-ROM」と呼ばれる独自フォーマットを使っている。GD-ROM フォーマットは標準の CD や DVD バーナーではコピーできない。しかし ROM BIOS のバックドアにより、海賊版業者は最終的にハードウェア改造なしで起動できる単一 CD-ROM イメージのビデオゲームを作れるようになった。インターネットで無料でダウンロードできるのに、誰がゲームを買うだろうか？蔓延した海賊行為がゲームの売上を落とし、ゲーム開発者のドリームキャスト向け開発意欲を削ぎ、セガのビジネスを直撃した。600万台を売り上げたものの、発売からおよそ3年後にドリームキャストは市場から撤退した。今のセガはゲーム開発専業となり、かつての競合相手であるソニーや任天堂、さらにはマイクロソフト向けにもゲームを作っている。

ドリームキャストの経験から学べることは多いが、メッセージは明快だ。CD-R、DVD-R、ネットワークといったほぼ無料のソースから、大がかりなハードウェア改造なしにコードを実行できてしまうことは、クローズドコンソールモデルに基づくあらゆるコンソールビジネスにとって致命傷だ。これは Microsoft の Xbox にとって切実な問題だ。Xbox は元来、さまざまなソースからコードを読み込むよう設計された標準的な PC ハードウェアで構成されているからだ。したがって、コンソール市場における Microsoft の命運は、Xbox セキュリティシステムがどれだけうまく機能するかと切り離せない。Xbox のセキュリティシステムは、これまでのところかなりよく持ちこたえている。これまでに見つかった弱点はいずれも、少なくとも、保証が無効になるようなハードウェア改造を必要としていた。もっとも簡単なものでも、はんだ付け不要の改造パーツを取り付ける必要があった。ハードウェア改造が必要だという事実は、これらの弱点の現実的な影響を限定的なものにしている。多くのユーザーは、機器のカバーを開けるだけでもためらうからだ。一方で、ハードウェアに手を入れずに、Xbox 上で好きなコードを実行したいという需要

は非常に大きい。その需要は、正当な目的を持つ人々から、そうではない人々まで、幅広い層に存在している。

Xbox は自らの設計の犠牲者だ。標準 PC ハードウェアを採用したことで、「開かれた」Xbox としての価値がハッカーと海賊版業者の双方にとって大きく高まってしまった。Xbox は Microsoft が PC アーキテクチャを採用した理由とまったく同じ理由で、週末ハッカーやホビイストの格好のターゲットになる。既存の PC プログラムは容易に Xbox に移植できるのだ。加えて、PC ハードウェアについては、すでに広く深い知識が共有されている。そのため、Xbox をハックするための学習コストは、ほかのゲーム機ほど高くない。一方で、PlayStation 2 や GameCube は理解するまでのハードルが高い。しかも、PC 向けアプリケーションを移植するうえで妨げとなるアーキテクチャ上の制約も抱えている。古いゲーム機のエミュレータを移植しやすいこと、知名度が高いこと、互換性のあるデバッグ・テスト用ハードウェアを入手しやすいこと。こうした条件がそろっているため、Xbox は海賊版ソフトを作る側にとっても人気のターゲットになっている。

さらに、Xbox アーキテクチャと PC アーキテクチャの類似性は、Xbox を優れた教育の場にしている。本書で得られる知識は、組み込みハードウェアやゲームコンソールを超えた領域にも応用できる。本書の内容の大半は PC に直接応用できるはずだ。また、PC 世界から受け継がれた Xbox 関連のドキュメントリソースが豊富にあり、ウェブ検索エンジンから簡単にアクセスできる。こうした資料へのアクセスのしやすさは、本書の知識を土台に学びを深めようとする読者の力になるだろう。

Xbox はまた、ありきたりな PC より魅力的な教育素材でもある。PC 実装はハードウェアの詳細にバリエーションが多すぎて、PC を対象とした実践的なステップバイステップのハッキングガイドを作ることは難しい。一方、Xbox を対象としたガイドは、ほぼどのショッピングモールや電子機器店でも買える何百万台もの同一ユニットで正確であることが保証されている。

ハッカーとハッキングについて

本書は伝統的な意味でのハッキング——探求のプロセスと方法——についての本だ。ゲームのリッピングや特定のセキュリティチェックのパッチ当てに特化した章がないことを不思議に思う人もいるかもしれない。なにしろ、それこそがハッキングのすべてではないか？ 実際、「ハッカー (hacker)」という言葉は、コンピュータに対する一般の関心が高まるにつれて、またセンセーショナルリズムを売り物にするマスメディアがハッカーに対する世間のイメージを塗り替え続ける中で、長年かけて大きく変化してきた。

もともとハッカーとは、好奇心と探求心のために情熱を注いで取り組む人のことを指した。コンピュータのカバーを外してその設計を最適化しようとしたハードウェアハッカーもいた（初期のコンピュータは個別の部品で構成されており、単純な工具で意味ある改造ができた）。計算リソースが乏しく遅かったため、最も簡潔でエレガントなコードを書こうと励んだソフトウェアハッカーもいた。電話システムの隅々を探索するハッカーや、大学キャンパスの建物の屋根やトンネルを探り歩くハッカーもいた。初期のハッカーはこうした活動のすべてに関わるが多かった。ハッカーたちは発見や成果（ハック）

を互いに自由に共有した——動機は金銭ではなく、知的好奇心を満たすことと仲間たちの熱狂にあったからだ。その結果、ハッカーたちは実力主義のグループを形成する傾向があり、メンバーシップも昇進も、もっぱらハックする能力によって決まった。

テクノロジーが進化してコンピュータが高速化・集積化されるにつれ、ハードウェアハッキングの労力は見返りに釣り合わなくなっていった。コンピュータの興味深い部分は、気密性の高いセラミックパッケージの奥深くに埋まり、高性能な顕微鏡でも見づらいシリコン構造の中に刻み込まれるようになっていった。コンピュータの性能を倍増させるほどのハードウェアハックも、数ヶ月以内にムーアの法則によって無意味になってしまった。一方ソフトウェアハッキングは、アルゴリズムや最適化よりもアプリケーションに重心を移していった。メモリとプロセッサが安価で豊富になるにつれ、コードのコンパクトさやエレガントさはもはや直接の価値を持たなくなった。コンパイラ技術も、手書きアセンブリとほぼ同等の速度でコードを実行できるまでに進歩していた。1980年代末には、「ハッカー」といえば、眠りながらも大量のCコードを書き上げ、一晩で優れたアプリケーションを作り上げられる人物を指すようになっていた。かつてのハードウェアハッカーたちは、ソフトウェアハッカーに転身するか、高価な趣味を支援できる大学の研究室や企業へと退くことになった。³

当時、「ハッカー」という言葉はパスワードやプログラムを破り、本来アクセスできないマシンやソフトウェアに侵入する人々と結びつけられることが多くなっていた。ハリウッドもそのイメージ形成に一役買った。十代の若者が数回のキー操作で世界を核戦争の瀬戸際に追い込んだり、地下室でAIのサイバーモンスターを作り出す天才が暗躍したりする映画が次々と公開された。⁴そうした映画の誇張は一般の視聴者にはそのまま受け取られ、「ハッカー」の暗いイメージが定着していった。この不正確なステレオタイプが、システムやプログラムへの不正アクセスに焦点を当てた「クラッカー (cracker)」という新たな言葉の誕生を後押しした。

テクノロジーがハッカーを形成してきたのと同様に、現代のハッカーもテクノロジーに形成されている。新世代のハッカーは、現代のコンピュータテクノロジーを包む「使いやすい」ユーザインターフェースとメディアのきらびやかさを突き破るために努力しなければならない。誰もがコンピュータを使い、直感的かつ完璧に動作することを当然と思っている。しかし、フードの下で本当に何が起きているかを理解している人はごくわずかだ。

計算テクノロジーはあまりにも複雑になり、初心者ますます「象に触れる7人の盲人」の寓話のようになっている。インターネットを探索することからハッキングの旅を始める人もいれば、コンピュータのオペレーティングシステムを探ることから始める人もいる。コンピュータのカバーの下を覗き込むことから始める人もいる。それぞれが1年間自分の切り口を探索したとして、最終的に持つコンピュータテクノロジーへの視点はまったく異なるものになるだろう。

若いハッカーと古参の文化的断絶が僕にはっきりと見えたのは、MITの自称ハッカーの生意気な新入生が「Windowsのコンピュータはどこにあるんですか？……こんなダサイSunのコンピュータばかりで、AOLすら入っていないじゃないですか！MITならインター

ネット環境がいいと思ってましたよ」と鼻で笑ったときだ。彼は、「ダサい Sun のコンピュータ」が世界で最も堅牢なオペレーティングシステムの一つを搭載したかなり強力なワークステーションであることを理解していなかった。AOL の向こうにインターネットが広がっているという事実も——さらに言えば、MIT キャンパスがインターネット誕生の地の一つであり、ほとんどの ISP より多くの IP アドレスを持ち、インターネットのバックボーンに直接接続されているという事実も、彼の頭の中には存在しなかったようだ。

コンピュータテクノロジーが日常生活のあらゆる隅々に浸透したことで、ハッカーのステレオタイプはさらに強化された。特に、ハッカーを現代のロビンフッドとするメディアの描き方が、ハッキングをセキュリティへの侵害やコンピュータリソースへの不正アクセスと切り離せないイメージとして定着させてしまった。今や典型的なハッカーはウェアーズ、Code Red、ピングフラッドの担い手とされ、「開発者」が Linux や BSD を担うとされる。ハッカーとは「0wn jh00r b0x0r」する「31337 d00d (エリートなやつ)」であり、ハードウェアハッカーとはネオンライトでコンピュータケースをオーバークロックして改造するやつらだ。

ハッキングはトレンドになり、メディアが作り出したステレオタイプに合わせようとする人々が増えている。「Xbox をハックしたのはそこに Xbox があったからだ」——挑戦的で、新鮮だったからだ——という単純な理由を、今日では人々に納得させることが非常に難しい。同様に、ハッキング以来 Xbox に取り組んでいない理由を理解してもらうことも難しい。Xbox のセキュリティを解析してしまえば、後に残るのは標準的な PC に過ぎない——僕にとってそれはそれほど面白い作業対象ではなく、Microsoft からの訴訟リスクを冒す価値も明らかにない。

ハッキングの政治学

1998 年にデジタルミレニアム著作権法 (DMCA) が施行されると、暗号技術はハッカーの手の届かないものになった——法律によって、著作物へのアクセス権を保護する暗号手法を調査できるのは、「正規の学習課程に従事し、雇用されているかまたは適切な訓練や経験を持つ」5 研究者だけと規定されたのだ。その結果、Xbox ハッキングは政治的に敏感なテーマとなった。暗号技術をハッカーの法的権利の範囲に留めようとするハッカーと立法者の戦いだ。

Xbox ハッカーへの Microsoft の対応は称賛に値する——少なくとも今のところ、Xbox ハッキングプロジェクトを迫害したり閉鎖しようとする動きはない。この姿勢が、DMCA を使ってハッキング活動を止めようと考えている者たちへのロールモデルとなることを願っている。あれほどの Xbox ハックが世に出ているにもかかわらず、Microsoft はゲームの堅調な販売を続けている。Xbox ハッキングが生み出した関心とバズは、海賊行為によるダメージを上回るほど Microsoft の販売を押し上げたかもしれない。(もちろん、僕はハッカー寄りなので、この解釈は偏っている。リバースエンジニアリングのより客観的で詳細な法的分析は、第 12 章「注意せよハッカーよ (Caveat Hacker)」で Electronic Frontier Foundation の Lee Tien が書いている。)

ハッカーにとってDMCAで最も憂慮すべき点は、「社会に利益をもたらすイノベーションは研究機関や企業の内部からしか生まれない」という誤った前提を法制化していることだ。突然、自宅で趣味として著作権保護に使われる暗号手法を探索することが犯罪になってしまった。こうした技術の研究を確立された機関だけに限定することは、所属なき個人による技術開発の可能性を封じることになる。自分のガレージで自由に研究・開発する権利がなければ、ビル・ヒューレットとデーブ・パッカード、スティーブ・ジョブズとスティーブ・ウォズニアックは今どこにいたろうか？ハッカーがコードで自由に表現する権利が規制されていたとしたら、Linux や netBSD は生まれていただろうか？

ハッカーに破られた著作権保護スキームが一つあるたびに、その破られ方から、より優れた保護策の作り方を学ぶ者がいる。著作権を保護する技術的手段の研究とその成果の普及を規制する法律を制定することは、著作権技術は壊れていて永遠に改善できないと認めることに等しい——一般の人々が著作権管理技術を理解できるようにすることの唯一の結果は、その技術の消滅だということを。僕はその考え方への反論を示そう。Xbox ハッキングの研究に対して受けた最も優れたピアレビューのいくつかは、学術コミュニティからではなかった。世界中の個人ハッカーから——特に外国から——来たものだった。彼らはアクセス制御技術を自由に探索し理解できる環境にいた。米国のより厳格な法律と企業の訴訟好き傾向は、電子セキュリティにおける米国の競争力にすでに悪影響を及ぼしており、これはまだほんの始まりに過ぎない。

DMCA の社会的影響は、世界中のハッカーコミュニティに広く感じられている。Xbox に取り組んでいた間、世界中の優れたハッカーたちに出会う幸運に恵まれた。アメリカのハッカーはグループの中で最も萎縮していた。才能あるエンジニアでありながら、迫害を恐れてそのスキルを Xbox 関連の問題に適用することをひどく嫌がっていた。その結果、Xbox ハッキングで最も興味深い成果の一部はヨーロッパやアジアのハッカーコミュニティによって得られている。重要なことに、これらの成果はアメリカではほとんど知られていない。こうしたハッカーたちは自分の発見をアメリカ人と共有しようという動機をほとんど持たないからだ。実際、多くの外国のハッカーは、アメリカ企業からの報復を恐れるなどの理由から、自分たちの発見をコミュニティの外に出さないよう意識的に努めている。この「頭脳流出」は、デジタル著作権の公平かつ効果的な管理という重要な技術分野における米国の競争力を強化するうえで、何ら寄与していない。そして今日のグローバル経済では、アメリカ企業が外の世界から切り離されたふりをして生き残ることはできない。

Xbox セキュリティシステムに関する論文が無事に発表できたことを、DMCA が言論の自由の権利と著作権管理技術の経済的利益の両方を保護している証拠として挙げる人もいるかもしれない。だが、僕の状況はアメリカのハッカーの典型ではなかった。当時は大学院生だったので、研究をめぐる訴訟に巻き込まれたとしても、心配する家族もなく、失う重要な資産もなかった。Electronic Frontier Foundation (EFF) からの惜しみない法的支援もあり、法的な地雷原を案内してもらえた。EFF は論文を最も法的に安全な形に位置づける手助けをし、DMCA のもとでの権利と義務を教えてくれた。

たとえば、「回避の前に Microsoft から承認を得るための誠実な努力」6 が義務付けられている。(承認そのものは必要ないが、誠実な努力は必要だという点に注意。) EFF はそのような手紙の起草を手伝ってくれた。また、MIT が僕の研究を関連機関として発表することを認めさせるための戦いも必要だった。Xbox セキュリティのリバースエンジニアリングに向けたすべての直接的な作業は、自費で、自分のアパートで、自分の時間を使って行ったものだ。MIT は当初この事実を口実にして自分たちと僕の研究を切り離そうとし、EFF に助言を求めることを余儀なくされた。MIT は最終的に、共感してくれた教授たちのさまざまな説得と、Microsoft から届いた建設的で脅迫的でない手紙を受け取った後、僕が MIT の学生として論文を発表することを認めた。

言論の自由に弁護士は必要ないはずだし、自由な思考に研究の承認書が求められてはいけない。論文の発表のために戦ったのは、失うものが何もなく、ハッカーとしての権利について声明を出すことに意義があると信じていたからだ。残念ながら、表に出てこない多くのハッカーたちは養うべき家族と失いかねない仕事を抱えており、EFF の支援を得られるほど恵まれてもいない。

今あなたが読んでいる本は、DMCA が言論の自由に萎縮効果をもたらしているもう一つの実例だ。もともと技術出版社のジョン・ワイリー・アンド・サンズ社に委託されていたこの本は、Microsoft からの訴訟とバックラッシュへの恐れから、土壇場でキャンセルされた。そうした検閲は不満で落胆させるものだ。著者によってはそこで立ち止まり、恐怖に声を封じ込められることを許してしまうかもしれない。しかし僕は、ハッカーとして自由に妨げなく語る権利のために声明を出すべく、自費出版という法的・財政的リスクをとっている。それでもこの道が障害なしというわけではない。本の予約注文プロセスは2日目に止まった。当初のeコマースプロバイダー、Americart が「ハッカー素材の販売のためのカートサービスの提供を断った……月15ドルでは、DMCA の訴訟に名前を連ねられるリスクを冒す余裕はない」と言ったからだ。

強調しておかなければならないが、本書は Microsoft の著作権を侵害しておらず、本書で紹介する知識は著作権の回避に直接使えるものではない。侵害行為を行うには、著作権管理の回避を明確に目的とした、相当量の追加的なスキルと専門的なノウハウを磨く必要がある。本書が回避ツールだと主張することは、回路基板、組み込みソフトウェア、暗号学についてのすべての本が回避ツールだと主張することと同じだ。

ハードウェアの「フェアユース (fair use)」に対する DMCA の適用範囲は、経済的に大きな意味を持つもう一つの重要な政治テーマだ。正規に購入または作成した代替ソフトウェアを実行する目的で、暗号保護されたブートシーケンスを改造または回避することは違法か？この問いへの答えは、Xbox ハッカーの行く末によって部分的に決まるかもしれない。DMCA のリバースエンジニアリング免除規定7の厳格な解釈は、こうした回避行為を違法とする強力な論拠を示している。

特に、リバースエンジニアリングが許可されるのは相互運用性 (interoperability) のためだけであり、相互運用性とは「コンピュータプログラムが情報を交換し、そのプログラムが交換された情報を相互に使用できる能力」を意味する。しかしこの定義には二つの地雷

が潜んでいる。第一に、ハードウェアベースのセキュリティ手段を回避することは、(ソフトウェアの) プログラムのセキュリティ手段を回避することとは異なると主張する余地がある。技術的には強い議論ではないかもしれないが、この条項が法的にテストされた事例は、僕の知る限りまだない。第二に、目的はハードウェアのセキュリティ手段と「情報を交換する」ことではなく——それらを「バイパスする」ことだ。

ハードウェアセキュリティ機構のリバースエンジニアリングを認めるべきでないとする最後の論拠は、「付随的な著作権回避」だ。リバースエンジニアリングのプロセスで得られた情報は、著作権回避デバイスの作成にも同様に使えてしまう。言い換えれば、少なくとも Xbox の場合、相互運用性を可能にする基礎研究は、回避デバイスを構築しようとする者にも間接的に使えてしまう可能性がある。実際には、Xbox にある非常に特定の設計上の欠陥のおかげで、著作権回避を有効にしなくてもブートセキュリティの回避が可能になっている。ただし、この欠陥は近い将来パッチされる可能性があり、そうなれば元の問いに正面から向き合うことになる。

「フェアユース」が代替アプリケーションの実行を目的とした Xbox セキュリティのリバースエンジニアリングをカバーしないと判明した場合、重大な経済的影響が生じる。最も重大な点は、Microsoft が法的に制限されたハードウェアをエンドユーザーに販売し、ユーザーを自社ソフトウェアに縛り込むことができるということだ。これを使えば、コンピュータハードウェアとソフトウェアに対する破れない独占を作り出せる。たとえば Microsoft は、Microsoft のオペレーティングシステムのみを実行するよう自社ハードウェアをセキュア化するベンダーに補助金を提供できる。この金銭的インセンティブは顧客に転嫁され、割引されたハードウェアを購入する動機になる。インストール済みハードウェアの相当部分が Microsoft のオペレーティングシステムに固定されれば、ロックされたハードウェアで他の OS を実行することが違法になるため、Microsoft は競争のない市場で価格を設定できるようになる。

現実には、DMCA がハードウェアのフェアユースを制限したとしても、政府や市民団体が Microsoft の行動を独占的行為の観点から注視しているため、このシナリオの実行は難しいかもしれない。しかし、スマートフォン、PDA、セットトップボックスといった新興市場では、ベンダーがこうした低価格戦術によって競合相手に優位に立とうとすることは、あながち非現実的ではないかもしれない。少なくとも、そうした戦術は訴訟期間中——競合相手の市場ポジションに取り返しのつかないダメージを与えるのに十分な長さになりうる——競争を遅らせるために使えることになる。こうした懸念から、多くの Xbox ハッカーは意識的に、自らのエンジニアリング活動を通じて政治的信念を表明してきた。



図: 自分の作業台での著者。

ハックの裏にいる人々

本書を通じて、インタビューに応じてくれたさまざまなハッカーのプロフィールを収録している。このグループが唯一のハッカーたちというわけではない。実際これは自己選択的なグループだ——多くのハッカーは、Microsoft と強いつながりを持つ企業に雇われているか、迫害を恐れて水面下で活動しているからだ。インタビューの目的は、ハックの裏側にいる人々を少し紹介し、彼らの動機と方法を伝えることで理解を深め、新しいハッカーが仲間に加わるきっかけを作ることだ。

まず、僕自身を紹介させてほしい。Andrew「bunnie」Huang、たいていは bunnie と呼ばれている。この執筆時点で 27 歳。Andrew C. と Margaret Huang の息子だ。ミシガン州カラマズーで生まれ育ったが、今はカリフォルニア州サンディエゴで素晴らしい婚約者の Nikki Justis と暮らしている。最近 MIT で電気工学の博士号を取得した。この本を書くために選ばれた理由の一つは、Microsoft の Xbox のセキュリティシステムにおける最初の既知の弱点を発見・公表したからだ。

一般的に言って、僕がハックするのは、自分が作ったものによって誰かの生活が良くなったと知ることが純粋に嬉しいからだ。社会から受け取ったものを、自分の才能を通じて社

会に返す義務があると感じている。また、探求の挑戦そのものを楽しんでいる。エレクトロニクスをできる限り深く理解したい。ブラックボックスは僕をイライラさせる。開けることも理解することも許されていない箱ほど、好奇心をかき立てるものはない。その結果、暗号とセキュリティの手法には職業的な責務として深い関心を持っている。

ハードウェアをハックするのは、エレクトロニクスの美学を楽しんでいるからだ。はかないソフトウェアコードのビットと違い、一日の終わりに手で触れられる物理的な成果物があることに、何か満足感がある。少し馬鹿げているかもしれないが、趣味の一つは電子機器を分解して回路基板を「読む」ことだ。帯電防止袋から取り出したばかりの新品の電子機器の匂いには、何かわくわくするものがある——新しい冒険の始まりの匂いだと思っている。真っ白な紙の積み重ねのように誘いかけてくる。あの白紙の山を前にして、何を書き込もうかと考える。それは「有用な情報で埋めてみろ」と挑戦してくるようだ。

この探求心は子どもの頃に根ざしている。7歳くらいのとき、父が Apple II のクローンを買った。マザーボードだけを買ったのでケースはなかった。箱から取り出した瞬間を今でも覚えている——緑色の回路基板、光るチップ、色とりどりの抵抗器とコンデンサ。触りたくて仕方なかった！Apple II への好奇心はあふれるほどあったのに、マザーボードに触れることは許されていなかった。もちろん、そうなれば親の目が届かない隙に、チップをソケットから外して逆向きに差し込んでみるといった馬鹿なことをするのは当然の流れだった。

コンピュータを壊しかけること数回、ついに両親は Radio Shack から Shack 200-in-1 エレクトロニクス実験キットと、僕の最初のエレクトロニクスの本——Forrest Mims III 著『Getting Started in Electronics』——を買ってくれた。これらは僕にとって素晴らしいエレクトロニクス入門となった。回路や部品で遊びたいという欲求を満たしてくれたのだ。叔父も Horowitz と Hill の『Art of Electronics』の古いコピーと、マイクロプロセッサに関する本を何冊か分けてくれた。当時はハードウェアプロジェクトに関する連載コラムが回路図と写真付きで掲載されていた雑誌、Byte も購読していた。

少しずつエレクトロニクスの感覚が身についてきて、Apple II のユーザーマニュアルに収録された回路図と ROM リストを読み解けるようになった。（今でも、コンピュータは完全な回路図とソースコードと一緒に出荷されるべきだと信じている。）8年生（中学2年相当）になる頃には、Apple II 用の自作増設カードを作れるだけの知識をкаろうじて身につけた。そのカードには Radio Shack で購入した General Instruments SPO-256 音声シンセサイザーを搭載した。また Apple II にアナログ-デジタルコンバータを追加し、Apple II をしゃべるボルトメーターに変えるアプリケーションも書いた。ハードウェア製作を続け、MIT 入学前には 80188 マイクロプロセッサを使った独自の組み込みコンピュータを自作した。

MIT 在学中は、ZBT フラタニティのためにリモコン式ライトスイッチや音楽に反応するパーティーライトを作るといった楽しい小さなプロジェクトで、学業の退屈さをやり過ごした。この時期に付録 C「PCB レイアウト入門」で紹介するような、手頃な価格のプロトタイピングサービスと PCB CAD ツールを初めて知った。

大学生の予算でも使える回路基板製造サービスの登場は、ハードウェアハッカーにとって画期的な出来事だ。ついにワイヤーラップツールをしまうことができ、表面実装部品と複雑な回路が日常のホビイストの手の届くところに来た。

これまで、自分のプロジェクトはウェブページ (<http://www.xenatera.com/bunnie>) に記録して、誰もが経験から恩恵を受けられるよう心がけてきた。多くのプロジェクトは回路図、ガーバーファイル、ソースコードとともに公開しているが、最近のプロジェクトの中にはコンサルティングの仕事もあるため、残念ながらその成果を世界と共有することはできない。

目が向いているうちに、一つだけ明確にしておきたい。Xbox をハックしたことで MIT の博士論文を書いたわけではない。Xbox ハッキングは実際には論文とは少し関連していたが、論文の中心テーマではない脱線だった。

スーパーコンピュータに関する僕の論文⁸は、効率的なコードとデータ移行のアーキテクチャに焦点を当てていた。ビデオゲームコンソールへの関心は、すべてのハードウェアに対する僕の自然な好奇心と、指導教員の Tom Knight 博士の励ましから生まれた。ビデオゲームコンソールはコストパフォーマンスの頂点を体現しており、コストは今日のスーパーコンピュータにとって重大な課題だ。そこで、費用対効果の高いハードウェア構築から学ぶことができるかどうかを確かめるために、すべてのビデオゲームコンソールを調べるよう勧められた。Xbox にさらに Xbox に興味深いセキュリティシステムがあったことはボーナスだった。政府機関はスーパーコンピュータ技術に大きな関心を持っているため、スーパーコンピュータのセキュリティは常に検討すべきテーマだ。(実際、信頼できるコンピュータの構築について非常に興味深い論文⁹が、研究グループの同僚によって書かれている。Palladium や TCPA のような暗号的にセキュアな信頼されるコンピューティングプラットフォームの代替案に興味があれば、読むことを勧める。)

ハードウェアハッカーとして成長したいなら、最も大事なアドバイスは、粘り強く、徹底的であることだ。好きなことをやっているなら、粘り強さと徹底性は自然と身につく。また、ハードウェアハッカーであることの一部は、がらくた収集家 (pack rat) であることだ。新品の機器を買うのは法外に高いので、何に使えるか、僕は修理できるかどうかも分からなくても、壊れた機器や値の下がったツールを習慣的に集めている。試験機器の修理を試みることで自分が学習体験であり、結局スเปア部品のためにジャンクにしたとしても、非常に充実した経験だと分かった。

元 Apple Evangelist で Garage Technology Ventures 現役幹部の Guy Kawasaki の言葉を借りれば、「鳥のように食べ、象のように排泄せよ」。ハチドリは毎日自分の体重の 50% 相当の食物を摂取する。つまり「鳥のように食べる」とは、情報に対して飽くなき食欲を持つべきだということだ。無料のエレクトロニクス業界誌を購読し、ウェブを閲覧し (ただし閲覧するサイトは選ぶこと——あなたは食べたものになる)、無料のトレードショーに行き、入手できるあらゆるカタログや定期刊行物に申し込む。自分と友人のすべての電子機器を分解して、設計からできる限り学び取る。

ハードウェアハッキングでは、最も難しい問題の半分は、適切な部品や技術を選ぶだけで解決したり、ぐっと楽になったりする。「象のように排泄する」とは、情報や発見を仲間のハッカーと惜しみなく共有することだ。どれだけ情報を消化しても、すべてを知ることはいできない。発見を自由に共有することで、仲間のハッカーのアドバイスと善意を引き寄せ、心のシナジーが生まれる。すべての結果が物理的な成果物に根ざすハードウェアハッキングにおいては特に、技術や結果を隠すことは、他の人々がいずれ自分の助けなしに同じものを再発明するだけだという意味にしかない。一方で、何を言い共有するかについては判断も必要だ。人々が受け取れる情報には限りがある。新しいものや何らかの意味で興味深い発見を共有すれば、より注意深く聞いてもらえる。

言いたいことはこれだけだ、ドライバーを手に取り、ハッキングを始めよう！

注

1. “Trustworthy Computing” by Bill Gates, <http://www.microsoft.com/mscorp/execmail/2002/07-18twc.asp>
2. source: NPDFunworld
3. 最近のハードウェアハッキング技術はムーアの法則に追いつきつつあり、ハードウェアハッキングルネサンスが到来している。手頃な価格の基板製造サービスが登場し、インターネットの誕生によってコンポーネントの入手プロセスが簡素化された。さらに、MOSIS チップファウンドリサービスや FIB（集束イオンビーム）サービスなどが、個人のハードウェア愛好家にとっても集積回路ハッキングを財政的に可能な領域に引き込み始めた。
4. MIT の人工知能研究所所長である Rodney Brooks はかつて、地下室で AI の存在を作り出す変人発明家というハリウッドのアイデアは、裏庭でボーイング 747 ジャンボジェットを作ろうとするのとはほぼ同等だと述べた。
5. 17 U.S.C § 1201(g)(3), Factors in determining exemption. もちろん、「適切な訓練や経験を持つ」の意味は定義されていない。応用暗号研究のための最良のトレーニングには、実際の暗号システムをハックする実践的な経験が伴うべきだと僕は考える。
6. 17 U.S.C. § 1201(g)(2), Permissible acts of encryption research
7. 17 U.S.C § 1201(f), Reverse Engineering
8. 僕の PhD 論文のテキストは <http://www.xenatera.com/bunnie/phdthesis.pdf> で見ることができる。
9. 「A Minimal Trusted Computing Base for Dynamically Ensuring Secure Information Flow」 Jeremy Brown と Tom Knight による論文は、<http://www.ai.mit.edu/projects/aries/Documents/Memos/ARIES-15.pdf> で見ることができる。

第 1 章 保証を無効にする

道具の選択

ハードウェアハッキングは、一部のプロジェクトに必要な高度な道具のせいで、最初は敷居が高く感じるかもしれない。幸いなことに、ほとんどの基本的なプロジェクトはビデオ

ゲーム1、2本分の価格で揃う最低限の道具投資で十分に実施できる。付録A「機器の入手先」には、入門用道具の推奨リストと注文方法が掲載されている。

本章では、本格的なハードウェアハッキングに必要な基本的な道具を紹介する。カバーを開ける道具、電子部品の取り付けと取り外しの道具、回路を診断・探索する道具、そして回路基板を設計する道具だ。このうち最初の2種類は、品質のよいものをかなり手頃な価格で入手できる。オシロスコープやロジックアナライザといった診断・テスト用の機器は非常に頼りになるが、かなり重くて相当な出費になる。回路基板設計ツールは、驚くほど手頃な価格で優れたものが手に入ることもある。

本章の最後に、Xboxを開ける方法をステップバイステップで写真付きで説明する。経験豊富なハードウェアハッカーは、次のいくつかの章を読み飛ばしてもよい。

カバーを開けるための道具

何かをハッキングする最初のステップはカバーを外すことだ。ほとんどの電子機器はフィリップスドライバーと平ドライバーのセットだけで開けられるが、最も面白い箱には特殊なセキュリティビット (security bit) のセットが必要になる。

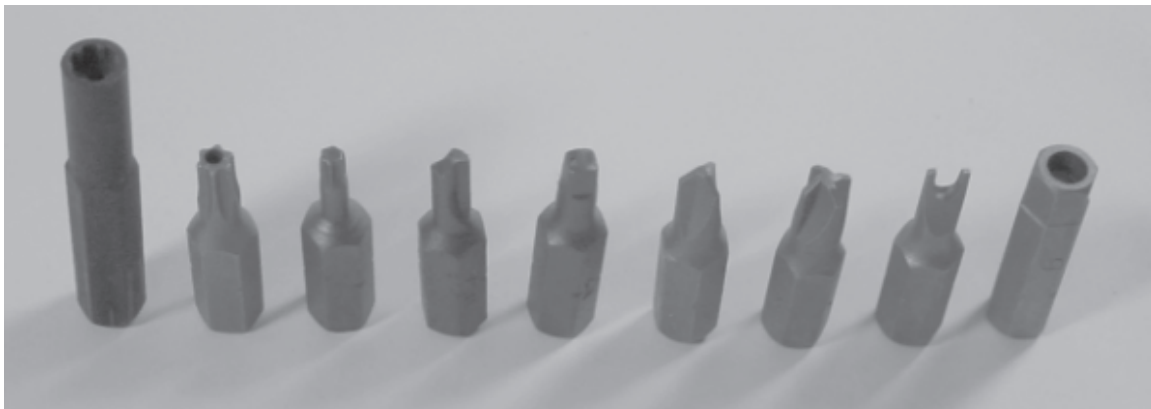


図 1-1: セキュリティビットの一覧。左から右へ：任天堂 4.5mm、セキュリティトルクス、標準トルクス、クラッチ、ロバートソン (四角)、トライウィング、トルク、スパナ、セキュリティアレン (六角)。

図 1-1 はよく使われるセキュリティビットを並べて示している。驚くことに、セキュリティビットセットは手頃で入手しやすい。MCM Electronics (www.mcmelectronics.com) では 105 ピースのセキュリティビットセット (MCM 注文番号 22-3495) を 20 ドル以下で、32 ピースセット (MCM 注文番号 22-1875) を 10 ドル以下で販売している。投資する価値は十分にある。任天堂のセキュリティビットは別売りだ。任天堂ゲームキューブで使われる大きな任天堂セキュリティビットは数ドルで入手できる (MCM 注文番号 22-1150、「4.5mm Security Bit」)。小さいバージョン (MCM 注文番号 22-1145、「3.8mm Security Bit」) は旧型の任天堂システムとゲームカートリッジにも使われている。

XboxにはT10、T15、T20サイズの標準トルクス（六角星形）ビットが使われている。これらのビットはかなり一般的で、Home Depotなどのホームセンターで購入できる。ハードドライブとDVDドライブ周辺の狭い場所に届かせるために、マグネット付きの延長ビットホルダーがあると便利な場合もある。

カバーを外す際に過度な力を使わないこと。ネジをすべて外したと思ってもカバーが動かない場合、ネジを1本見落としているか、摩擦ロックタブを押し込む必要がある。また、機器の底のゴム足の下やシールラベルの下にネジが隠れていることも多い。ラベルの下にネジを探すには、ラベルの表面を指でしっかりと擦る。ネジがある場所では指にへこんだ感触がある。（そのようなラベルを破ってネジにアクセスすることで保証は即座に無効になるが、心配はいらない。ほとんどの機器はサービスのために設計されているので、カバーを外すだけで損傷が生じることはほとんどない。）

頑固なアセンブリに遭遇してどうしても分離できないことがある。カバーやパネルが端の周辺でたわんで動く余裕があるなら、何らかの摩擦ロックがカバーを固定しているかもしれない。摩擦ロックは通常タブとスロット構造で、差し込みより取り外しがはるかに難しい形状だ。この場合は、ケースがどこで固定されているかを観察してタブを見つけ、小さな平ドライバーでタブを押し込みながらケースを優しく引き上げる。このようなタブが複数ある場合は、別のドライバーやペーパークリップを楔として入れて、他のタブを解除する間にタブが再係合するのを防ぐ。

カバーやパネルが強く押しても全く動かない場合は、接着剤で固定されているか溶接されている可能性もある。たとえばコンセントに直接差し込む「ウォールワート」型電源（正方形の黒い箱）はしばしばこのような方法で封鎖されている。そのような機器を分解すると、元の形に戻せなくなる可能性がある。

コンポーネントを取り付け・取り外しする道具

電子部品ははんだ付け（soldering）によって基板に取り付けられる。はんだ付けでは、低融点合金のはんだ（solder）を加熱して接合する金属の周囲に流し込む。はんだと母材金属の間に合金層が形成され、接合部が冷えると部品が電氣的・機械的に接続される。

はんだ付けの基本道具は、はんだごて（soldering iron）、はんだ、フラックス（flux）、はんだ吸い取り線（desoldering braid）だ。（細先ピンセットも、ファインピッチ部品や小さい部品を扱う作業に重宝する。）はんだごては加熱素子とチップで構成される手持ち式のツールで、チップを直接接触させて熱を伝えはんだ合金を溶かす。熱風や赤外線を使う他の方法とは異なる。最適な熱伝達に必要なチップの種類は状況によって異なる。たとえばほとんどの小さい表面実装部品をはんだ付けする場合、単純な尖ったチップよりも平らな「鑿型」または「円錐形の鑿型」チップの方が優れた性能を発揮する。

はんだごてにも多くのグレードがある。最安値のものは10ドル程度で、大きく扱いにくいチップが付属し、温度調節機能がない。ただ可能な限り熱くなるだけだ。より良いはんだごては価格が高く、チップの温度を積極的に調節するセンサーが付いていて動作が安定し、チップの寿命も延びる。より良いはんだごてには今日の電子機器に使われる微細な部品の作業に適した非常に細いチップを含む、幅広い選択肢がある。軽い用途には、良い

チップを取り付けた品質の高い直差し型はんだごてで十分だ。しかし基板を自分で作り本格的にハードウェアハッキングをしたいなら、Weller WTCPT や Weller WES50 のような品質の高い温度制御式はんだごてに 100 ドルを投資する価値は十分にある。

はんだには多種多様なものがある。ほとんどの目的には、ノークリーンまたは水洗い可能なフラックスコア付きの共晶 (eutectic) Pb-Sn 合金はんだ線が十分だ。共晶合金は液体から均一な固体へと直接変化するため望ましい。Kester ははんだの主要メーカーで、標準コアードワイヤーはんだの Formula 245 と 331 はどちらも非常に優れている。Formula 245 はノークリーンフラックスを使用しているが、必要ならイソプロピルアルコールを染み込ませた綿棒で残渣を拭き取ることもできる。Formula 331 は 245 よりも多くの材料に使えるフラックスコアを持つが、はんだ付け後すぐに水で基板を洗い流さないとフラックス残渣がベタつき、回路動作に干渉する可能性がある。Kester はんだは多くのディストリビューターが販売している。たとえば Kester 24-6337-8802 (25 ゲージ Formula 245 はんだ線、1 ポンドスプール) は Digi-Key (www.digikey.com) の部品番号 KE1410-ND だ。Radio Shack で販売されているはんだも十分良いが、粘着性のある黒い残渣を残す傾向があり、有機溶剤でのクリーンアップが必要だ。

はんだはフラックスマトリックスに微細なはんだボールを懸濁させたペーストとしても提供される。はんだペーストは、ファインピッチの表面実装部品を取り付ける際に非常に役立つ。(詳細は付録 B 「はんだ付け技術」を参照。)

はんだ接合が形成しにくい場合は、フラックスが万能薬だ。常にフラックスを手元に置いておこう。接合がうまく形成されないときは、接合部に少量のフラックスを直接塗布すると通常問題が解決する。フラックスにはペーストや液体など多種多様な製品があり、それぞれ異なるクリーンアップ方法が必要だ。便利なフラックス塗布ツールとしてフラックスペンがある。たとえば Kester 83-1000-0951 (Formula 951 ノークリーンフラックスペン) は Digi-Key (部品番号 KE1804-ND) から数ドルで購入できる。Radio Shack もチューブ入りのフラックスペーストを販売しているが、扱いにくく後片付けが必要だ。

最後に、はんだ吸い取り線ははんだ付けのミスや失敗を片付けるのに役立つ。はんだ吸い取り線は細い銅線の編み組みで、通常ドライフラックスが含まれている。使い方は、はんだごてとクリーンアップしたい接合部の間に挟むだけだ。編み組みが熱くなると、接合部の余分なはんだが毛細管現象で吸い込まれていく。あらかじめフラックスが含まれている場合でも、使用前に一滴フラックスを垂らすとより効果的だ。Chemtronics は優れたはんだ吸い取り線を製造しており、たとえば Chemtronics 60-3-5 「No-Clean Solder-Wick」 (Digi-Key 部品番号 60-3-5-ND) がある。

はんだ付けの基本的な技術については第 2 章の冒頭で説明する。そこでは Xbox のフロントパネルに青色 LED を取り付ける方法を学ぶ。

テストと診断のための道具

電子テスト機器は電子製品と同じくらい多くの形態がある。初心者にとっての基本的な「必須」ツールはデジタルマルチメータ (digital multimeter、DMM) だ。DMM はここ数年で非常に多機能かつ手頃になり、典型的なものは抵抗、電圧、電流、静電容量、ダイオー

ド極性、導通を約 50 ドルで測定できる。Radio Shack と Jameco (www.jameco.com) はどちらも入門用マルチメータをリーズナブルな価格で取り揃えている。(付録 A「ハッキング機器の入手先」には入門レベルのマルチメータの提案がある。)

基本的な改造やキット製作では、DMM はショートしている接続の確認と、電源を入れる前後の回路の健全性確認に役立つ。導通モードは、はんだ接合を失敗した可能性があるときに特に便利だ。

導通モードでは、テストプローブ間に低抵抗パスが存在すると DMM がトーンを発する。したがって、導通機能ははんだ接合の完全性を確認するのにも、隣接する接続との短絡を確認するのにも役立つ。ただし電源回路の短絡確認には導通モードを使ってはいけない。一部の基板では電源とアース間の抵抗が十分に低い (10 オーム程度) ため、正常な状態でも導通トーンが鳴ってしまうことがある。改造または組み立てた基板に電源を入れる前には、抵抗測定モードで電源ラインに完全な短絡 (0 オームの抵抗) がないことを必ず確認すること。

リバースエンジニアリングやより高度なプロジェクトに必要な基本ツールは、オシロスコープと、場合によってはロジックアナライザだ。オシロスコープは電気波形の詳細な形状を捉えるのに役立ち、タイミング、ノイズ、干渉の問題を診断できる。

オシロスコープの基本的な定義特性は、同時に表示できるチャンネル数と最大電気帯域幅だ。高品質のオシロスコープは通常 4 チャンネルで 500 MHz 以上の帯域幅を持つ。ディスカウントや中古のオシロスコープは 2 チャンネルで 20 MHz から 100 MHz の有効帯域幅を持つことが多い。すべてのオシロスコープの主な制限は、電気波形の短いセグメントしか表示できない点だ。

ロジックアナライザは大量のデジタルデータを捉えるのに役立つ。波形形状の取得能力を犠牲にして、広範なデータ分析・ログ機能を持つ。ロジックアナライザは複雑なデジタルバス (bus) や回路の診断に役立つ。基本的な定義特性は、サンプリングできるデジタルチャンネル数、最大サンプリングレート、最大サンプリング深度だ。典型的な現代のロジックアナライザには数十チャンネル、数百 MHz のサンプリングレート、数メガバイトのサンプリング深度がある。プログラム可能なトリガーアルゴリズムや、グリッチ・ランドパルスの検出機能を持つものもある。

残念ながら、新品のオシロスコープやロジックアナライザの平均価格は数千ドルから数万ドルに及ぶ。ただ、ほとんどのプロジェクトは最新・最高の機器を必要としないため、中古で済ませることができる。中古・ジャンク機材が集まるスワップフェストは、古いオシロスコープやロジックアナライザを安く手に入れる絶好の場所だ。eBay にも時々良い取引がある。オシロスコープとロジックアナライザのどちらかを選ばなければならない場合は、まずオシロスコープを入手することを僕はお勧めする。ロジックアナライザはオシロスコープほど汎用性がなく、通常は高価だ。オシロスコープをうまく使えばある程度のロジックデータを捉えることもできるが、ロジックアナライザでアナログ波形を測定することは決してできない。また、同等品質のオシロスコープを自作するよりも、FPGA とカスタム基板を使って自作のロジックアナライザを作る方が簡単だ。自作のロジック

アナライザは比較的安価にハイエンド・高速アプリケーションで動作するように構築できる。(第8章では、Xboxの重要な高速バスを傍受するために自作のロジックアナライザを構築した話を紹介している。)

いざとなれば、Radio Shackのパーツ約50ドルで非常にシンプルなデジタルトレースキャプチャデバイスを自作することもできる。かつてPS/2キーボードポートのデータを素早くキャプチャしなければならなかったことがあった。手元にテスト機器がなかったが、バーグラフLEDをいくつか配線し、データをシフトする8ビットレジスター(部品番号74HCT574)のセットを組んだブレッドボードで解決した——すべてRadio Shackで買ったコンポーネントだ。実際の設計はかなりシンプルだが、使用目的が非常に限定的なので詳細は省く。要点は、デジタルデータをキャプチャするための独自デバイスを自作できるということだ——ロジックアナライザに数千ドルをつぎ込む前に考えてみる価値がある。

設計のための道具

ハッカーのコレクションを仕上げるために必要な最後の道具セットは、PCボードとFPGAの電子設計ツールだ。PCボードとFPGAの設計については付録で詳しく説明するが、品質のよいバージョンがほぼコストなしで手に入ることは特筆に値する。設計・製作ツールのコストを含め、Xboxの価格以下で洗練された再構成可能なハードウェアコンポーネントを持つ完全な回路基板を設計・構築できるのだ。

PCボード設計はかつて非常に高価だった。ツールは数千ドルかかり、シンプルな基板の製造でも数百ドルかかった。今では初心者でも合計70ドル以下でシンプルな基板を製造できる。PCボード設計ツールとして、Altium(旧称Protel)はCircuitMaker2000というツールを販売している。CircuitMaker2000を詳しく試したわけではないが、第一印象ではAltiumの現在は廃止されたProtel 99SEに非常に似ている。www.circuitmaker.comから無料の30日間デモまたは無料の学生版(制限あり)をダウンロードできる。最初の設計プロジェクトには最適だ。無料ツールで最初の基板を設計したら、Sierra Proto Express(www.sierraprototexpress.com)などのベンダーで執筆時点では基板1枚約30ドル、最低2枚注文から製造できる。価格はもはや深刻な障壁ではない。ぜひ独自のカスタムプリント回路基板を使ったプロジェクトに挑戦してほしい。

コラム：静電気：回路の天敵

静電気、つまりESD(Electro-Static Discharge、静電気放電)は集積回路の天敵だ。現代のICは特にESDに敏感で、むき出しのトランジスタを破壊するには数ボルトあれば十分だ。静電気放電は数百ボルトから数千ボルトの範囲にならないと感じないため、気づかぬうちにデバイスを破壊してしまうことがある。

良いニュースは、ほとんどのチップはESDへの耐性を高めるための特殊な構造で作られているということだ。それでも、自発的にテストに参加しない方がいい。体の静電気を中和するために、回路基板やチップに触れる前に常にアースされた金属物体に触れること。適切に配線されたコンセントに挿しているコンピュータのケースの金属露出部分が、いい出発点だ。

ほとんどのコンピュータショップで入手できる静電気防止リストストラップを着用することで、ESDによるXboxへのダメージのリスクを大幅に減らせる。リストストラップが有効に機能するためには、アースされた物体に取り付けなければならない。

思い切ったやり方として、カーペットなしのコンクリートの床に裸足で立つことでもアースできる。むき出しのコンクリートの床は驚くほど導電性が高く、不適切に配線されたコンセントの電子機器との長時間の接触で感電や火傷を受けることがある。リノリウムや木の床も、使われているタイルやワックスの種類によっては有効なアース接地点になる。特殊な導電性ワックスやスプレーを塗布して床の導電性を確保することもできる。

FPGA (Field Programmable Gate Array、フィールドプログラマブルゲートアレイ) は安価なシリコンプロトタイピングのソリューションだ。FPGAはプログラム可能な相互接続を持つ大型のゲートとストレージ素子のアレイで構成されていて、FPGAのゲートとワイヤの容量の範囲内で、あらゆる種類のデジタルデバイスを実装できる。大型のFPGAは数百万ゲートの容量を持ち、マイクロプロセッサと周辺機器を含む完全なシステムを格納できる。FPGAはまた非常に手頃だ。10万ゲートのXilinx Spartan II FPGAは単品で約20ドルだ。さらに、Xilinx FPGAの設計・合成環境を制限なしで無料で入手できる！XilinxはVerilogやVHDL合成、HDLテストベンチ生成、消費電力解析ソフトウェアなどの機能を含む「ISE WebPack」という無料製品を自社ウェブサイト (www.xilinx.com) で公開している。Verilogはハードウェア設計用のC言語に似た言語で、厳密に型付けされたマルチスレッドのCと思えばいい。ハードウェアを試してみたいソフトウェアハッカーには朗報だ。www.opencores.org のようなオープンソースハードウェア設計コミュニティもあり、マイクロプロセッサやその他の興味深いデジタルコンポーネントのコードを無料でダウンロードできる。

Xbox を分解する

道具について説明したので、実際にハッキングを始めよう。Xboxをハッキングする最初のステップはボックスを開けることだ。カバーを外すために必要な道具は以下の通りだ：

- T10 と T20 のトルクスビット (六角星形のビット)
- ビット用のドライバーハンドル
- 静電気防止安全具 (「静電気：回路の天敵」のコラムを参照)
- 小型の平ドライバー (あると便利だが必須ではない)

[!NOTE] Xboxを分解し始める前に、いくつか頭に入れておこう。まず、ものを分解するときには常に永久損傷のリスクがあり、Xboxを分解すると保証が無効になる。次に、作業を始める前にこのセクション全体を必ず読み通すこと。そして、楽しもう。

ステップ1：安全確認

Xboxのプラグを抜く。Xboxをプラグに挿したままにすると、危険な、場合によっては致命的な電圧にさらされることになる。

ステップ2：ケースのネジを外す

Xboxをひっくり返して底面を確認する。外殻の上半分と下半分を固定する6本のネジがあり、すべてラベルまたはゴム足の下に隠れている。図1-2はすべてのネジ位置を示している。

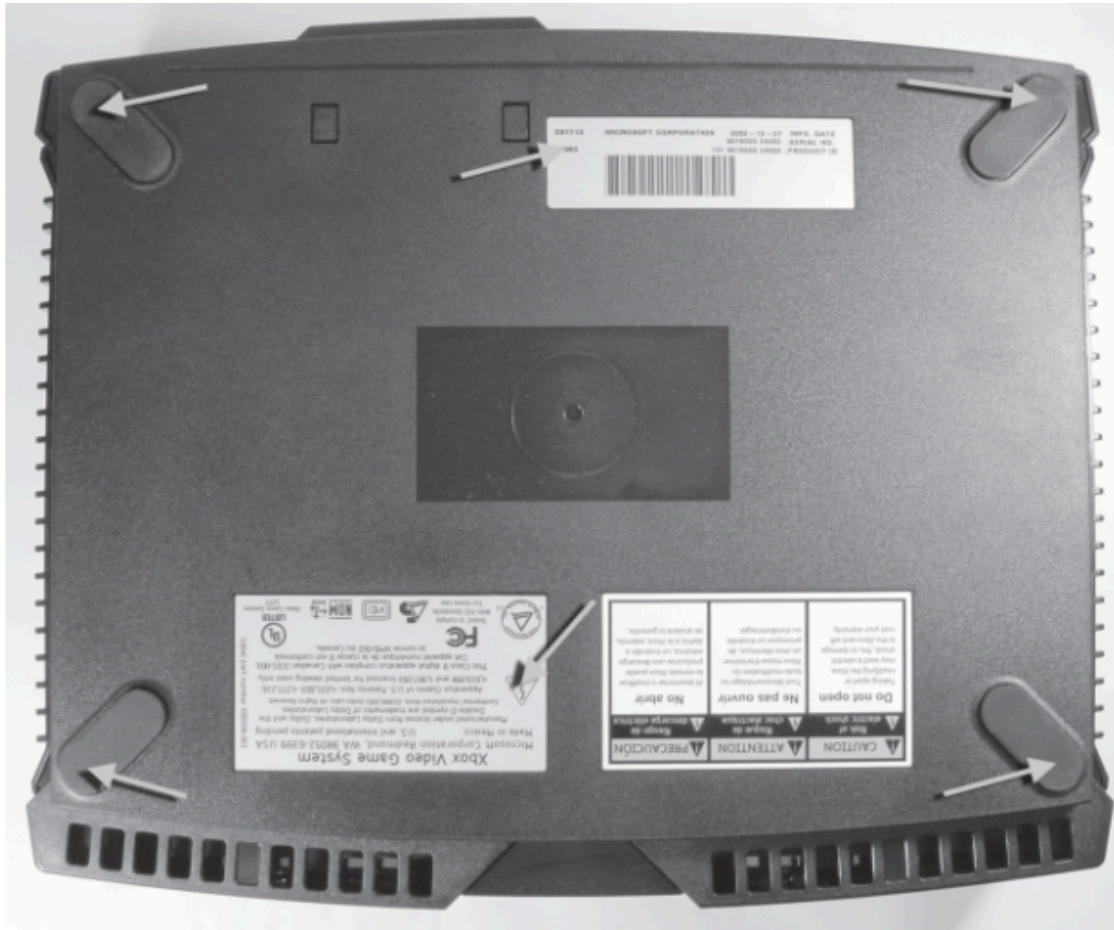


図1-2: Xbox ケースのネジの位置。Xbox の底面を下から見た図。

ゴム足は強力な接着剤でXboxに接着されている。足を外すには小さな平ドライバーの助けが必要な場合がある。図1-3はこの手順を示している。ゴム足の端を剥がしたら、接着剤のバックグを保持するためにできるだけ均等に剥がす。注意すれば後で足を再接着できるが、何度か取り外すと粘着力が失われる。代わりに、ハードウェアストアでゴム足を購入して、滑りやすい面や傷つきやすい面でXboxを使用する場合に取り付けることもできる。

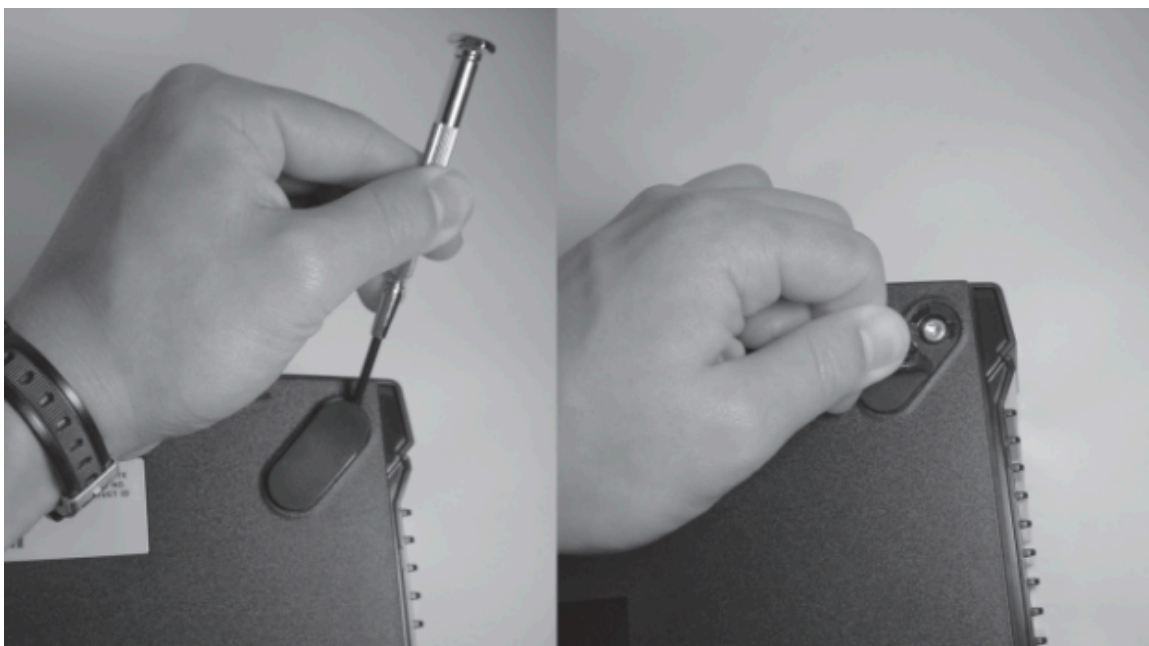


図 1-3: 小型の平ドライバーで Xbox のゴム足の端を持ち上げ、慎重に剥がす。

T20 サイズのトルクスビットを使ってゴム足の下にあった 4 本のネジを外す。ネジはかなり長い、ネジ山は短いので取り外しは素早くできる。

残り 2 本のネジはシリアル番号ラベルと製品認証ラベルの下に隠れている。図 1-4 はこれらのネジの位置を示している。



図 1-4: シリアル番号と製品認証ラベルに隠されたネジの位置。

ネジを見つけるには、ネジがあるべき大まかな領域の上で指をしっかりと擦る。ラベルはネジ穴の上でわずかにへこむ。ビットでラベルを刺してネジ穴に引っかかるまで回転させ、ネジを外す。ラベルの見た目を気にするなら、ネジを外す間ラベルを押さえておく。そうしないとラベルが剥がれるか、破れることがある。

[!TIP] 小さなトレイやプラスチックの袋をネジの保管に使って、なくさないようにしよう。Xbox を固定しているネジはかなり特殊で、地元のハードウェアストアで適切な代替品を購入するのは難しいかもしれない。

ステップ 3：上部カバーを外す

この時点で 6 本の同一の長いネジを取り外しているはずだ。Xbox を表向きに戻し、両手の手のひらで Xbox の側面を優しく握り、優しく揺らしながらカバーを持ち上げようとする。この方法でカバーが外れない場合は、背面から指でケースをこじることによって「開始」する必要があるかもしれない。まれに、前面からドライバーでこじる必要もあるが、注意して優しく行うこと。図 1-5 はケースを外すのに使えるいくつかの箇所を示している。

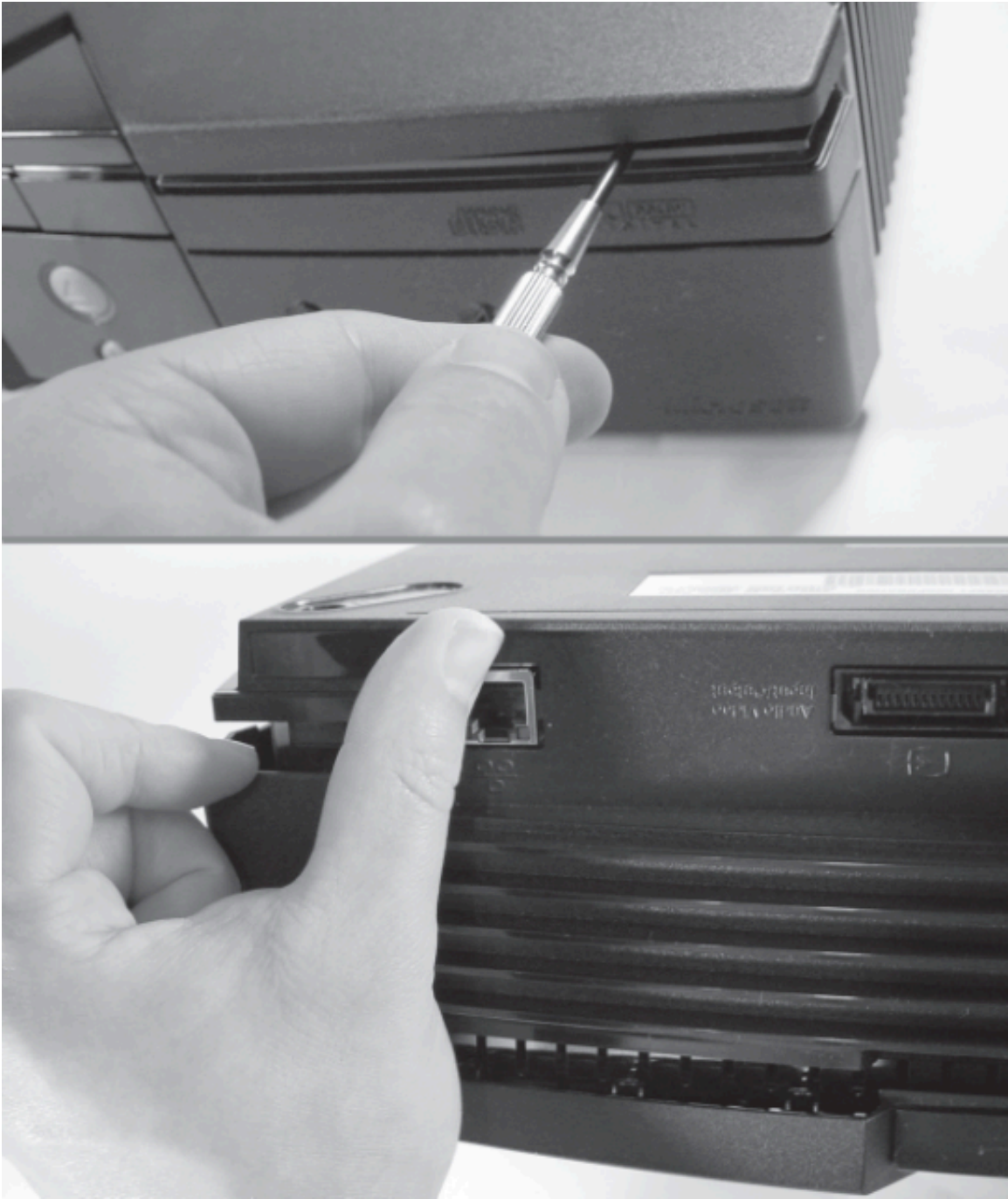


図 1-5: 頑固なカバーを開けるためにこじることができる場所。

ケースカバーを無理に外さないこと。上述のこじり方法でカバーが外れない場合は、この本の出版以降に追加のネジが加えられているかもしれない。Xbox 背面のラベルを触って追加のネジを探すこと。

ステップ4：ディスクドライブを移動する

内側に入ったら、黒いプラスチックキャリアに取り付けられた2つのドライブが見えるはずだ。マザーボードにアクセスするには、ディスクドライブを移動（必ずしも取り外す必要はない）する必要がある。ドライブケーブルを切断する必要はないが、ドライブキャリアのネジを外す必要がある。

3本のT10トルクスネジがキャリアを固定している。図1-6はこれらのネジの位置を示している。1本はケース背面付近のグレーのIDEリボンケーブル（IDE ribbon cable）の下に隠れており、2本はケース前面付近のドライブ表面から約1インチ奥に埋め込まれている。埋め込まれたネジを見るには懐中電灯か直接の頭上照明が必要かもしれない。

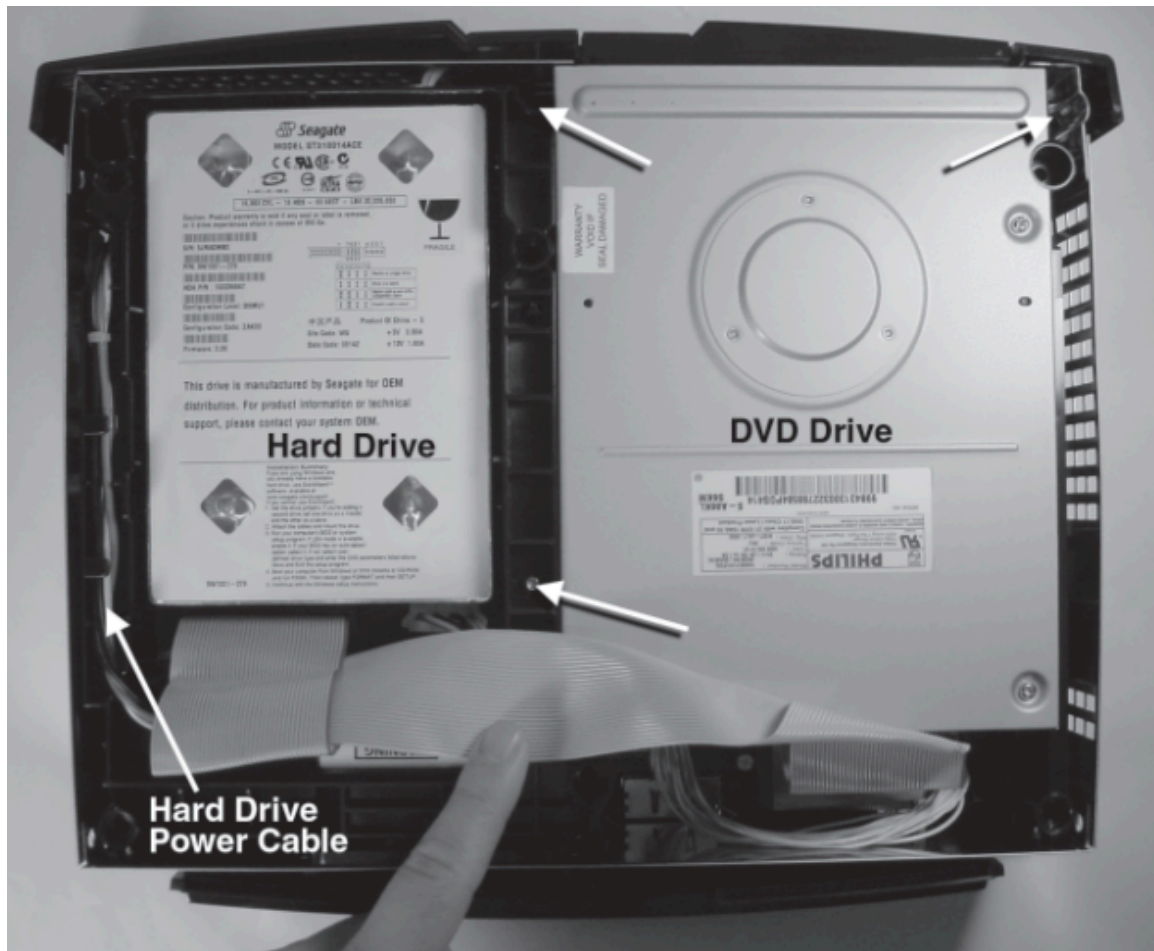


図1-6: 3本のドライブキャリアネジの位置。グレーのIDEリボンケーブルは写真のために持ち上げられている。左がハードドライブ、右がDVDドライブ。

埋め込まれたネジは、ドライバーにマグネット付きビットホルダーがない場合、ビットが位置決めの際に滑り落ちる傾向があるため少し難しい。ビットはネジとプラスチックキャリアの間のスペースに噛み込めるほど小さい。ネジを外したと思っているのにドラ

イブキャリアが動かない場合は、実際にネジが外れているか再確認すること。過度な力をかけなくてもキャリアを少し持ち上げられるはずだ。

ネジを外したら、ハードドライブの電源ケーブルをノッチから解放する必要がある。解放しないとドライブキャリアの取り外しの妨げになる。電源ケーブルはハードドライブの外側に黒、黄、赤のワイヤーの束だ。ハードドライブは図 1-6 の左側のデバイスで、キャリアの端に沿ったノッチに保持されている。ケーブルにいくらかの余裕ができるまで、ノッチからケーブルを優しく解放する。図 1-7 は作業完了時にどれくらいの余裕が必要かを示している。

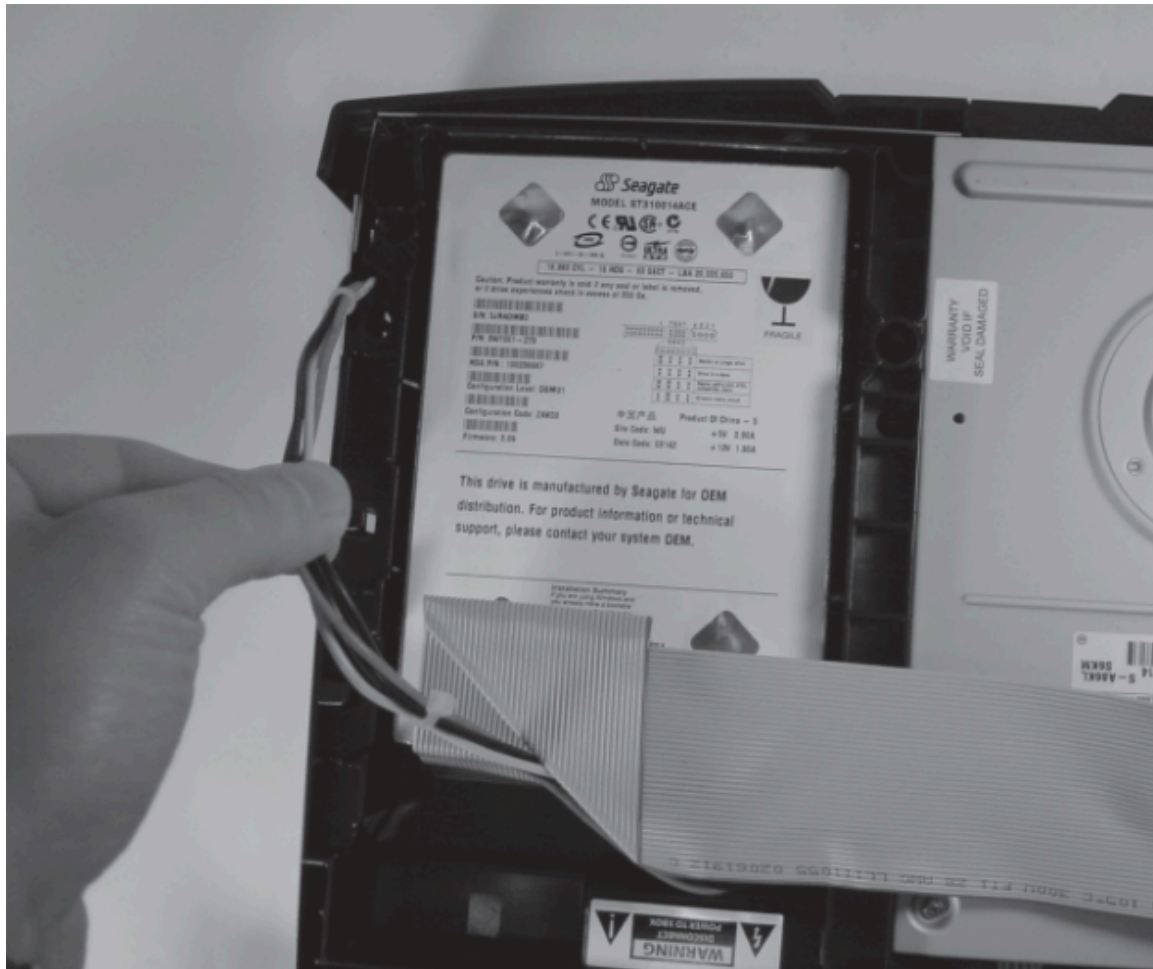


図 1-7: ハードドライブケーブルを保持ノッチから解放する。解放したら、ケーブルに数インチの余裕ができるはずだ。

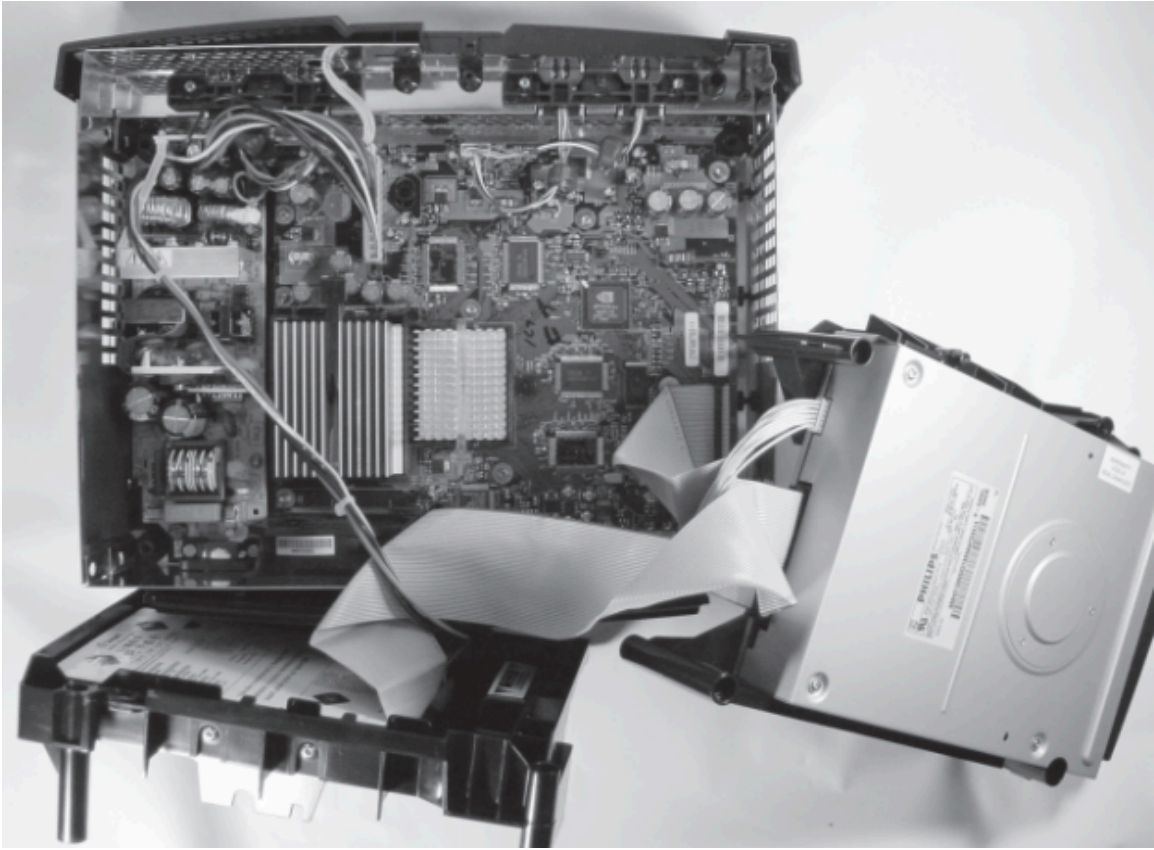


図 1-8: ディスクドライブを外側に折り出した状態。テストと実験のために電氣的な接続は維持されている。

ケーブルが解放されたら、ハードドライブをその格納位置から持ち上げて DVD ドライブの上に置く。両手で DVD とハードドライブを取り出し、図 1-8 に示すように外側に折り畳んで側面に垂らす。ドライブを接続しているケーブルは抵抗なく簡単に折り返るはずだ。(DVD ドライブ背面から来る黄色いケーブルに注意すること。注意しないとソケットから引き抜いてしまうことがある。)

これで Xbox のマザーボードと電源の全体を、ドライブを切断せずに確認できる。この配置はハードウェア改造後の Xbox のテストに有利だ。

電源には常に注意すること。Xbox がプラグに挿さっている限り、Xbox がオフになっていても電源は「ライブ」だ。触れると怪我または死亡する可能性のある電圧がある。また、ハードドライブキャリアの底部と DVD ドライブは Xbox 内部の空気ダクトとしての二次的な役割を果たすことにも注意。ドライブを側面に垂らした状態で Xbox を長時間動作させると、Xbox が過熱する可能性がある。同様に、CPU と GPU の大型アルミニウムヒートシンク (heat sink) に注意すること。Xbox が動作中に不快なほど熱くなり、やけどするほどになる可能性がある。

次の章のプロジェクト（Xbox のフロントパネルの LED を交換する）では、ハードドライブを切断する必要はないが、切断することが好ましい。Xbox のケースを操作する際にケーブルに不要なストレスをかけることを防げる。

ステップ 5：ディスクドライブを取り外す（オプション）

多くの場合、ディスクドライブを完全に取り外す方がより便利でハードウェアにとっても安全だ。

そのためには、まずグレーの IDE リボンケーブルを Xbox のマザーボードから外す。次に、DVD ドライブに接続された黄色のディスクリットワイヤーケーブルをマザーボードから外す。この黄色いケーブルは DVD への電源を供給し、DVD ドライブのトレイ状態をマザーボードに伝達する。黄色いケーブルの 1 本のワイヤーを引っ張ってはいけない。ケーブルヘッドからワイヤーが外れてしまう可能性がある。白いコネクタを掴んで引くのが好ましいが、指が狭いスペースに入らない場合はワイヤーの束全体を掴んで優しく引いてコネクタを外すことができる。DVD ドライブを脇に置く。

次に、ハードドライブの電源コネクタを外す。コネクタはハードドライブにしっかりと差し込まれているため、直接引っ張るだけでは外れた際にケースの鋭い縁で怪我をする危険がある。怪我を避けるために、図 1-9 に示すように小さな平ドライバーを使ってハードドライブの本体からコネクタを優しく取り外す。



図 1-9: 平ドライバーでハードドライブの電源コネクタを取り外す様子。

これでディスクドライブアセンブリを完全に取り外すことができる。グレーの IDE リボンケーブルはまだ各ドライブユニットをつないでいる状態だ。必要に応じて取り外せるが、後で Xbox に再接続できるよう向きを覚えておくこと。

Xbox の再組み立て

Xbox を分解したので、次のセクションの楽しいプロジェクトに進もう。終わったらこのセクションに戻り、Xbox の再組み立て方法を確認すること。

Xbox に何かを取り付ける前に、Xbox をひっくり返して優しく揺さぶって、誤って中に落としてしまった緩んだネジや部品がないか確認する。緩んだネジはビデオゲームコンソールの終わりを意味し、火災の危険もある。少しでも疑問があればこの確認をする価値は十分にある。

Xbox の再組み立ての最初のステップは、ディスクドライブを再取り付けすることだ。前のセクションの手順に従った場合、DVD ドライブとハードドライブはすでにグレーの IDE リボンケーブルで接続されているはずだ。接続されていない場合は、折り目が最も少ないケーブルの端をハードドライブに接続し、次にケーブルの中間コネクタを DVD ドライブに接続する。ケーブルは一方向にのみ接続できる。コネクタの突起とドライブコネクタのノッチの位置に注意すること。

グレーの IDE リボンケーブルの残りの空き端を Xbox のマザーボードに差し込む。ケーブルはマザーボードに一方向にのみ差し込める。IDE プラグ中央の突起とマザーボードの受け口のノッチに注意すること。次に、黄色い DVD ケーブルを Xbox のマザーボードに接続する。このケーブルもマザーボードのコネクタに一方向にのみ差し込み、ケーブルに突起があり、マザーボードの受け口に対応するノッチがある。DVD ドライブを Xbox に収め、Xbox ロゴのついたトレイがケースの縁に対してどのように収まっているかを観察することで、水平で均一かどうかを確認する。うまく収まるまでに何度か試す必要がある可能性が高い。向きについて迷ったら、グレーの IDE リボンケーブルの折り目を参考にすること。

もうすぐ完了だ。次にハードドライブを DVD ドライブの隣に収める。このドライブも所定の位置に収まるまでに少し調整が必要になる可能性が高い。DVD ドライブと水平で、Xbox ケース周りのメタル製 EMI 放射シールドの縁と揃っているはずだ。ハードドライブの電源ケーブルをドライブトレイの保持ノッチに通して、ハードドライブに差し込む。コネクタを確実に接続するにはかなりの力が必要で、コネクタが完全に差し込まれると軽いスナップ感がある。最後に、グレーの IDE リボンケーブルをドライブキャリアの元々の保持フックに通す。

この時点で Xbox をプラグしてテレビにつなぎ、Xbox が正常に起動することを確認するといいい。カバーを外したままでも Xbox を継続して動作させることができる。ドライブキャリアの底部と DVD ドライブによって形成されるすべての冷却ダクトが所定の位置にあるからだ。ドライブが Xbox に正しく接続されていない場合、コンソールは起動するがサービスが必要だというメッセージを表示する。このメッセージが表示されたら接続を注意深く確認すること。

今度はドライブをネジで固定する時だ。この時点で 9 本のネジがあるはずだ。ドライブキャリア用の短い T10 ドライブネジ 3 本と、ケースカバー用の長い T20 ドライブネジ 6 本だ。ネジが 1~2 本足りなくても、余分にあっても慌てないこと。Xbox はネジが 1~2 本足りなくても固定できる。ネジとケースカバーを外した逆の順番で取り付ける。(どこに入るか分からなくなった場合は、このセクションの前の写真を参照すること。)

ケースカバーが取り付けられたら、再び Xbox を使う準備ができた！

第2章 箱の中で考える

リバーズエンジニアリングは、歯ごたえがあってやりがいもあるゲームだ。勝つには、スキルと運の両方が少し必要になる。そしてどんなゲームとも同じで、スキルを磨くにはひたすら実践あるのみだ。

ハッカーとしてのスキルを磨く第一歩は、素材に対する直感を養うことだ。ハードウェアの場合、感覚をつかむにはあらゆるもののカバーを外して、部品が何で何をするものかを把握しようとするのが近道だ。Digi-Key、Jameco、Newark Electronics といった部品販売業者から紙のカタログを取り寄せて、空き時間にページをめくってみるのもいい。最初は部品カタログを読むのが辞書を引くように感じるかもしれないが、回路基板(circuit board)を多く見ていくうちに、次第にすべてが理解できるようになっていくだろう。

リバーズエンジニアリングの次に強力なツールは、パターンマッチングだ。ハードウェアエンジニアはみな同じ物理法則の制約を受け、同じ種類の部品を使う。また、既存の設計を部品化して使い回すのも好む。その結果、同じ設計パターンがさまざまな製品に繰り返し現れる。設計パターンを見分けられるようになれば、部品番号がひとつも分からなくても回路の機能を特定できる。正式な電気工学の訓練を受けなくても、リバーズエンジニアリングでかなりのスキルを身につけられる。

リバーズエンジニアリングの最後のツールは実験だ。直感とパターンマッチングで回路の秘密が解けないときは、システムをいじって応答を観察し、機能を推測するしかない。実験でハードウェアが壊れることもあるが、民生用ハードウェアのほとんどは製造上の要件として検査できるよう設計されているものだ。しかも Xbox の場合、新品が比較的安く手に入る。最初から2台買って1台を「生け贄」にしてしまえば、ハードウェアの実験に思い切って踏み込める。

本章では、リバーズエンジニアリングの基礎を紹介する。直感を養うための回路基板の読み方といった基本テクニックから、パターンマッチングや基本的な設計パターンの認識といった中級テクニックまで取り上げる。

回路基板を読む

典型的な電子機器のカバーを外したときに最初に目に入るのが、回路基板（プリント基板）だ。一般に緑色や茶色をしたこの基板は、銅箔・ガラス繊維・エポキシ樹脂を重ねた多層構造で、その配線パターンには部品同士の接続関係、つまりネットリストがそのまま刻み込まれている。配線パターンをたどれば、各部品がどのように接続されているかを正確に把握できる。部品の配置と配線パターンのレイアウトにも、設計者の思考プロセスへのヒントが含まれている。

回路基板の基礎

典型的なプリント基板は、エポキシ樹脂を含浸させたガラス繊維のシートで絶縁された複数層の銅箔の配線パターンから成る。生の基板の色は白みがかかった色や茶色で銅の配線が見えているが、ほぼすべての基板にははんだレジスト (soldermask) と呼ばれる薄い樹脂膜が塗られており、おなじみの緑色が生まれる。溶けたはんだははんだレジストに

付着しないため、製造中に余分なはんだが基板に付着してショートを起こすことがない。ランドやパッドにあたる部分のはんだレジストには開口部があり、銅の酸化を防いではんだ付け性を高めるため、薄いスズやはんだのめっきが施されていて銀色に光っている。

はんだレジストの上には通常、シルクスクリーン (silkscreen) と呼ばれる白い文字のレイヤーがある。基板上の各部品はシルクスクリーンレイヤーに輪郭と固有の指定記号 (reference designator) を持つ。指定記号を見れば、基板上の部品と回路図の部品を素早く対応させられる。指定記号は部品の命名規則に基づいているため、部品の機能を推測する手がかりにもなる。表 2-1 は Xbox で使われている部品命名規則をまとめたものだ。

指定記号	部品種別
C	コンデンサ
R	抵抗器
U	集積回路またはトランジスタ
L	インダクタ
RP	抵抗アレイ (resistor pack)
Q	トランジスタ
CR	ダイオード
J	コネクタまたはジャンパ
RT	リセットブルヒューズ
Y	クリスタル (水晶振動子)

表 2-1: Xbox の部品命名規則。

[!TIP] Xbox のマザーボードにはシルクスクリーンレイヤーで基板の端に沿って印刷された便利な座標系がある。部品面では側面に沿って A から G、上下に沿って 1 から 8 までの座標になっている。基板の裏面は側面に沿って M から V までだ。文字 I、O、Q、S はそれぞれ数字の 0、1、5 と混同しやすいため省略されている。Xbox マザーボードの部品指定記号はこの座標系を使ってエンコードされており、たとえば J7D1 (LPC デバッグポート) は表面の座標 7D にある。本書ではこの座標系と部品指定記号を頻繁に使って特定の部品を指し示す。

配線レイヤー間の接続は、銅が充填された穴であるビア (via) によって行われる。回路基板のコストはレイヤー数が増えるにつれて上がるため、民生用電子機器のほとんどはレイヤー数を最小限に抑えるよう設計されている。ラジオ受信機やオーディオアンプは一般的に片面基板を使うが、最新の PC マザーボードは 6~8 層に及ぶことがある。Xbox のマザーボードは 4 層構成だ。上の 2 層はチップ間の情報伝達に、内側の 2 層は電源供給に使われている。内側の電源層がほぼ銅の固体シートのため、Xbox のマザーボードは一見不透明に見える。リバーシエンジニアリングにとって朗報なのは、すべての信号層が不透明な電源層の外側にあるため、Xbox マザーボード上のすべての接続を目視でたどれるという点だ。これは 2~4 層の信号層を電源層の内部に埋め込んでいるマザーボードとは

対照的だ。信号層を埋め込むと信号の追跡が難しくなる。(なお、信号を電源層の内部に埋め込む判断は、通常セキュリティではなく、基板の層間での電気信号の相互作用の物理的特性によって決まる。)

信号をたどるのはそれほど難しい。発信元の部品から基板への接続点から始めて、配線をたどる。配線が円と交差する場合、信号が基板の反対側に続いている可能性が高い。配線が終端して基板の反対側への接続がない場合、電源を配るための広い銅箔の面、つまり電源プレーン (power plane) のいずれかに接続されている可能性が高い。

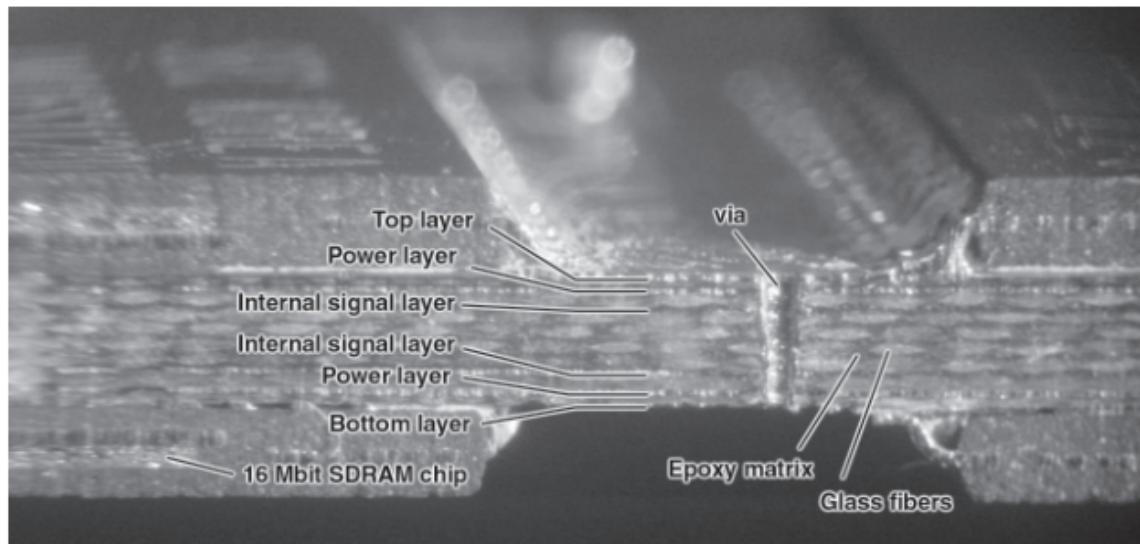


図 2-1: 典型的な回路基板の断面図。

[!NOTE] 試してみよう

Xbox のマザーボードでいくつかの信号をたどってみよう。手始めに、セクター 8C にある 40 ピン IDE コネクタ J8C1 を見る。IDE コネクタからの信号のほぼすべてが、いくつかの抵抗アレイ (resistor pack) を経由して MCPX と呼ばれるチップに向かっている。そのチップについて何が推測できるだろうか？ IDE コネクタからの配線のいくつかが前後に蛇行しているのに気づくだろう。これはすべての配線を同じ長さにそろえるためのテクニックだ。詳しくは「なぜ基板の配線はあちこちで蛇行しているのか？」のコラムを参照。

部品

信号をたどる経験が少し身についたところで、基本的な部品の見た目を学ぼう。

部品は受動素子 (パッシブ) と能動素子 (アクティブ) に分類される。大まかに言うと、受動素子は信号を増幅できないため、通常リードは 2 本だ。複数の受動素子が 1 つのパッケージにまとめられることもあるため、リードが複数になることもある。受動素子にはコンデンサ、抵抗器、インダクタが含まれる。Xbox のマザーボードで最も多い受動素子はコンデンサだ。コンデンサは電荷としてエネルギーを蓄える。Xbox では主に、CMOSデ

がある。2つ目は基板上の部品の隣のラベルを読み、表 2-1 を参考に指定記号から部品の機能を推測する方法だ。

能動素子は信号を増幅でき、3 本以上のリードを持つ。最も単純な能動素子はトランジスタで、3 本、ときに 4 本のリードを持つ（ディスクリートの「MOSFET」トランジスタには 4 番目の「ボディ」端子を持つものがある）。最も複雑な能動素子は集積回路（IC）で、CPU やメモリチップなどは数百、ときに数千のリードを持つ。集積回路はさまざまなパッケージに入っており、ボールグリッドアレイ（BGA：Ball Grid Array）パッケージのように接続部がパッケージの下に隠れることもある。

コラム：デジタル回路基板にはなぜこんなにも抵抗器とコンデンサがあるのか？

多くの基板を見ていると気づくのが、抵抗器とコンデンサの多さだ。コンデンサはノイズを抑え電源電圧を安定させるために必要で、至る所に置かれている。電源を配るための電源プレーンにもわずかな抵抗とインダクタンスが存在するためだ。こうした寄生成分は、電源を通じて大電流が切り替わると大きな問題を引き起こすことがある。コンデンサの正確な配置と選択は一種の職人芸だ。基板上で作業中に小さな砂粒ほどのコンデンサを一つ飛ばしてしまっても、交換なしで済ませられる場合が多い。ただし飛ばしたコンデンサをそのままにしておくと、たまにしか再現しない不安定動作や間欠不良が起きやすい。

コンデンサが各部品のすぐ近くで一時的に必要な電荷を供給するために至る所にある一方、抵抗器は余分なエネルギーを熱として逃がす。マザーボード上の高速信号は大量のエネルギーを運んでおり、抵抗器によって受信側で制御しながら消費されなければ、信号エネルギーが送信側に反射して問題を引き起こす。体育館の音響に似た現象だ。空の体育館で話すとエコーが起きる。話すのが速すぎると、エコーが干渉して聞き手には聞き取れなくなる。しかし壁に吸音材を貼れば、エコーは吸収されて邪魔されずに話せる。

抵抗器は壁の吸音材のようなもので、回路が高速で通信できるようエコーを抑える。コンデンサのほとんどとは異なり、これらの抵抗器を作業中に飛ばしてしまうと、回路を正常に動作させるために交換が必要だ。こうした「終端抵抗器」は 1 パッケージに 4 個か 8 個まとめられていることが多く、小型の集積回路のように見える。抵抗アレイ（resistor pack）は光沢があり、少し凸凹しており、白い縁があり、近くに「RP」という指定記号があることで他の部品と区別できる。抵抗アレイを通じて信号をたどる場合、信号は一方のピンから真っすぐ対応する反対側のピンに通過すると仮定するのがほぼ安全だ。

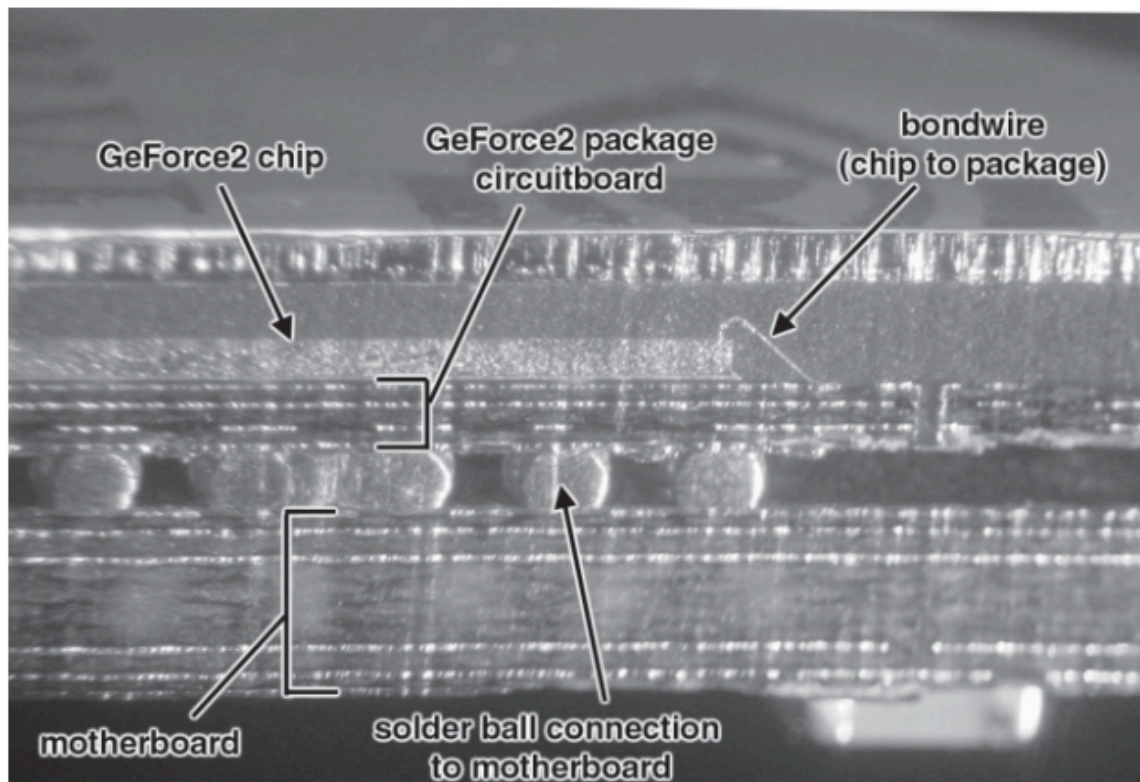


図2-3: マザーボードに実装されたBGAパッケージ部品 (GeForce2) の断面図。パッケージ下面の隠された接続を示す。

Xbox のマザーボードのグラフィックチップ MCPX と CPU はすべて BGA パッケージを使っている。図 2-3 は BGA デバイスの断面図で、パッケージ下の隠された接続を示している。

特定の集積回路の機能を特定することは、受動素子の場合より難しい。機能的に同一のシリコンが外観の大きく異なるさまざまなパッケージで購入できる。接続先や外観を見て機能を推測できる場合もあるが、最も確実な方法はチップの部品番号を読んでウェブで調べることだ。(一般に部品にはメーカーを識別するロゴや部品番号プレフィックスがあり、メーカーのウェブサイトで詳細を調べられる。)ロゴや部品番号プレフィックスが分からない場合は、以下のサービスが役に立つ。

1. www.findchips.com は部品番号や部品番号の一部を入力すると、多くのディストリビューターの在庫を検索できる。一般的な部品のほとんどは FindChips に表示され、部品の簡単な説明だけでなく価格や注文情報も得られることが多い。
2. www.google.com はウェブ上のあらゆるものをインデックスしており、部品番号も例外ではない。ロゴに含まれる文字と「semiconductors (半導体)」のような説明的な言葉を組み合わせて検索すると、メーカーのウェブサイトを見つけるのにも役立つ。メーカーのウェブサイトでは、専用の部品検索エンジンを探すか、半導体製品のサブ

ページに移動する必要がある。企業トップページの検索機能で部品番号が見つかることもあるが、多くの場合は役に立たない企業やマーケティングページしかインデックスされていない。

3. これらのサービスで見つからない場合は、部品番号のプレフィックスやサフィックスをいくつか取り除いてみよう。M29F080A の例では、部品番号 29F080 だけで検索すると、STMicroelectronics の部品と機能的に互換性のある部品を製造する複数のメーカーのウェブページが見つかる。

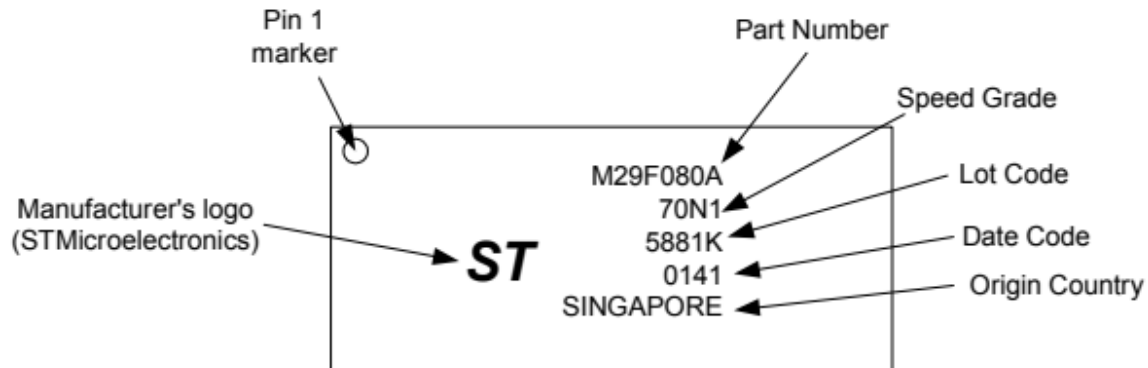


図 2-4: 典型的な IC 部品番号の構成。Xbox マザーボードの座標 U7D1 にあるチップのイラスト。

[!NOTE] 試してみよう

Xbox の部品番号を調べてみよう。Xbox のマザーボード上の U7D1 を探そう。図 2-4 は見つかるものを示している。部品番号は通常チップの上の最も長い数字で、1~2 文字のアルファベットから始まることが多い。メモリチップとプロセッサにはスピードグレードまたはクオリティサフィックスが続くことが多い。また、ほぼすべてのチップにデートコード（製造年週）がある。デートコードは通常 YY-WW 形式の 4 桁の数字で、YY はチップが製造された年、WW は製造週だ。この例では、M29F080A はシンガポールで 2001 年の第 41 週に製造され、スピードグレードは 70N1 だ。残りの番号 5881K はロットコードで、チップを製造施設での特定のシリコンウェハーまたはウェハーロットのトラッキング番号に紐づけるものだ。「ST」ロゴは STMicroelectronics を示しており、URL を www.st.com と推測するか、Google で検索すればすぐに見つかる。ホームページの検索フィールドに M29F080A と入力すると、この部品の詳細なデータシートと説明——8 Mbit ユニフォームブロックシングルサプライフラッシュ ROM——が直接見つかる。

テストポイント

民生用電子機器のほぼすべての回路基板には、工場での完成品基板の検査を容易にするための構造が備えられている。これらの「テストポイント (test point)」は、製造上の欠陥という避けられない現実に対処するために存在する。Xbox もテストポイントや製造上の欠陥とは無縁ではない。Xbox のマザーボードの底部には、Xbox 内のほぼすべての重要な信号にコンタクトプローブでアクセスできる数百もの小さな銀色の円——テストポ

イント——が並んでいる。テストポイントがあれば、顕微鏡や安定した手が必要な信号にも簡単にアクセスできるため、リバースエンジニアやハードウェアを改造したい人にとって歓迎すべきものだ。

量産ラインでは、針山のように多数のプロブを並べた検査治具を使い、基板上のテストポイントに一斉に接触させて検査する。この種の治具はベッド・オブ・ネイルズ（床）治具とも呼ばれる。名前の通り、スプリング内蔵のポゴピン（pogo pin）が何百本も並んでいる。マザーボードをテストベッドに位置合わせして、機械式プランジャーまたは真空チャックでクランプする。同様に、ポゴピンを使えばテストポイントを活用したはんだ不要の改造もできる。自分で回路基板を製作する必要がある（付録参照）が、できあがるのはネジで固定するだけのボードだ——はんだ付けは不要！

Xbox アーキテクチャ

パターンマッチングの例に入る前に、参考となるパターンが必要だ。Xbox の内部アーキテクチャをその参考として学び、最終的に Xbox のアーキテクチャを PC や他のビデオゲームコンソールと比較してみよう。

高レベルの構成

Xbox の CPU は 733 MHz で動作する Pentium-III 相当のプロセッサだ。CPU の「S-Spec」番号は Mobile Celeron のそれに最も近い。CPU は標準の P6 133 MHz フロントサイドバス（FSB：Front Side Bus）を介して、nVidia 製の NV2A と呼ばれるグラフィックス処理ユニット（GPU：Graphics Processing Unit）兼ノースブリッジコンボチップに接続されている。最も近い PC の類似品は nVidia 製の nForce IGP チップだ。

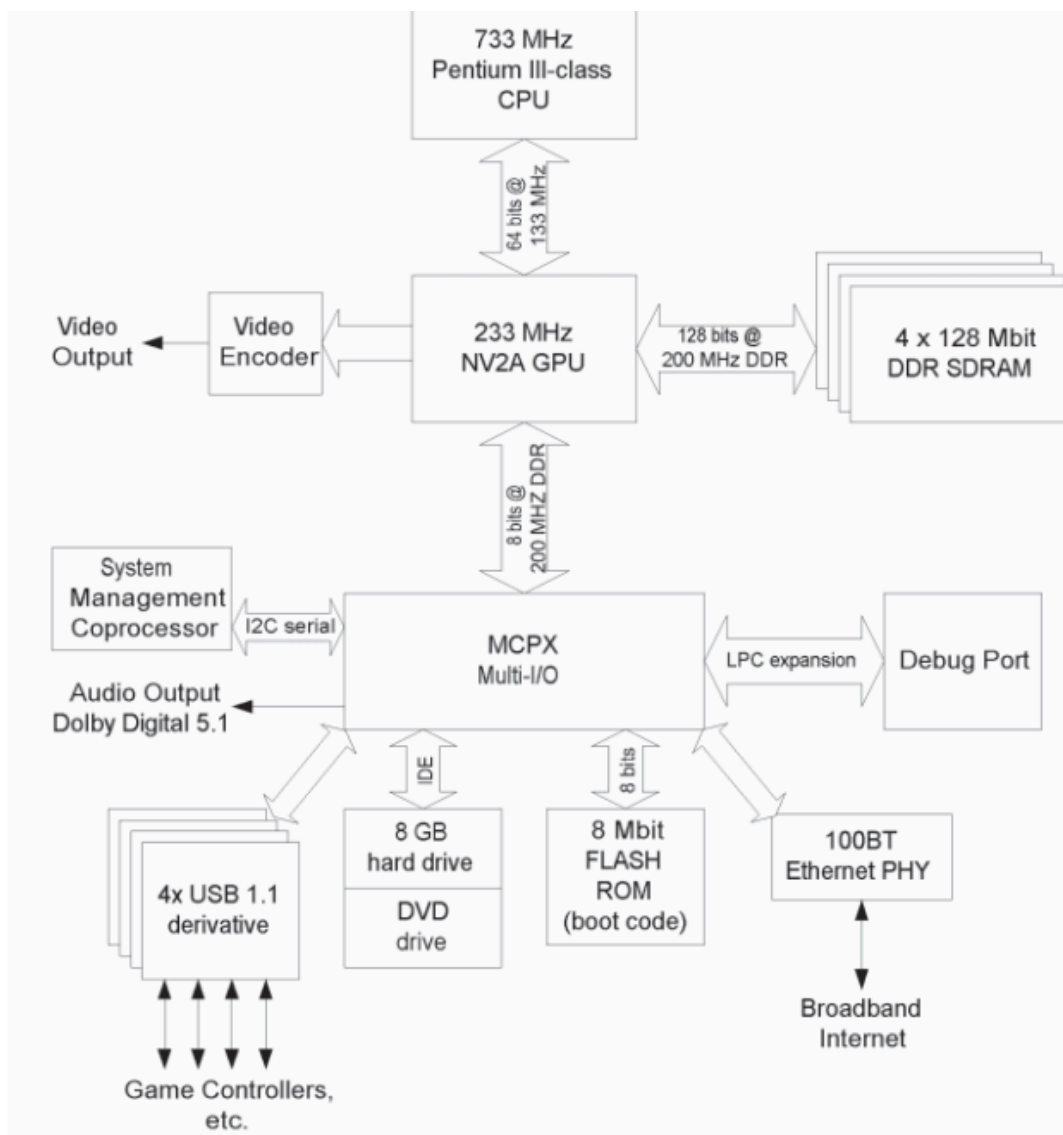


図 2-5: Xbox の高レベルアーキテクチャ図。

ノースブリッジロジックと GPU が 1 つのチップに統合されているため、CPU とグラフィックスプロセッサは共通のメモリバンクを共有できる。これを「ユニファイドメモリアーキテクチャ (UMA : Unified Memory Architecture)」と呼ぶ。従来の分離型ビデオ/メインメモリアーキテクチャと比較して、UMA は専用ビデオメモリを不要にするためコストが低い。ただし、メインプロセッサとグラフィックスプロセッサの間でメモリアクセスの競合が生じるため、特定の状況では UMA のパフォーマンスが低下する。この競合を軽減するために、システムメモリはしばしば複数のバンクに分割される。nForce IGP の場合、スイッチングネットワークを通じて GPU と CPU の両方が独立してアクセスできる 2 つのバンクにメモリを分割している。

GPUは「MCPX」と呼ばれる多機能チップにHyperTransportバスという高速で細い(ピン数の少ない)バスで接続されている。MCPXはサウスブリッジチップにXboxの周辺機器のほぼすべて——USBコントローラ、レガシーブートROMインターフェース、Dolbyデジタルオーディオプロセッサ、大容量ストレージIDEコントローラ、イーサネットコントローラ、システム管理機能のインターフェース——を統合している。

Xboxの主要ブロックの接続関係は図2-5に示されており、図2-6はこれらのブロックが実際のXboxマザーボード上のどこに位置するかを示している。

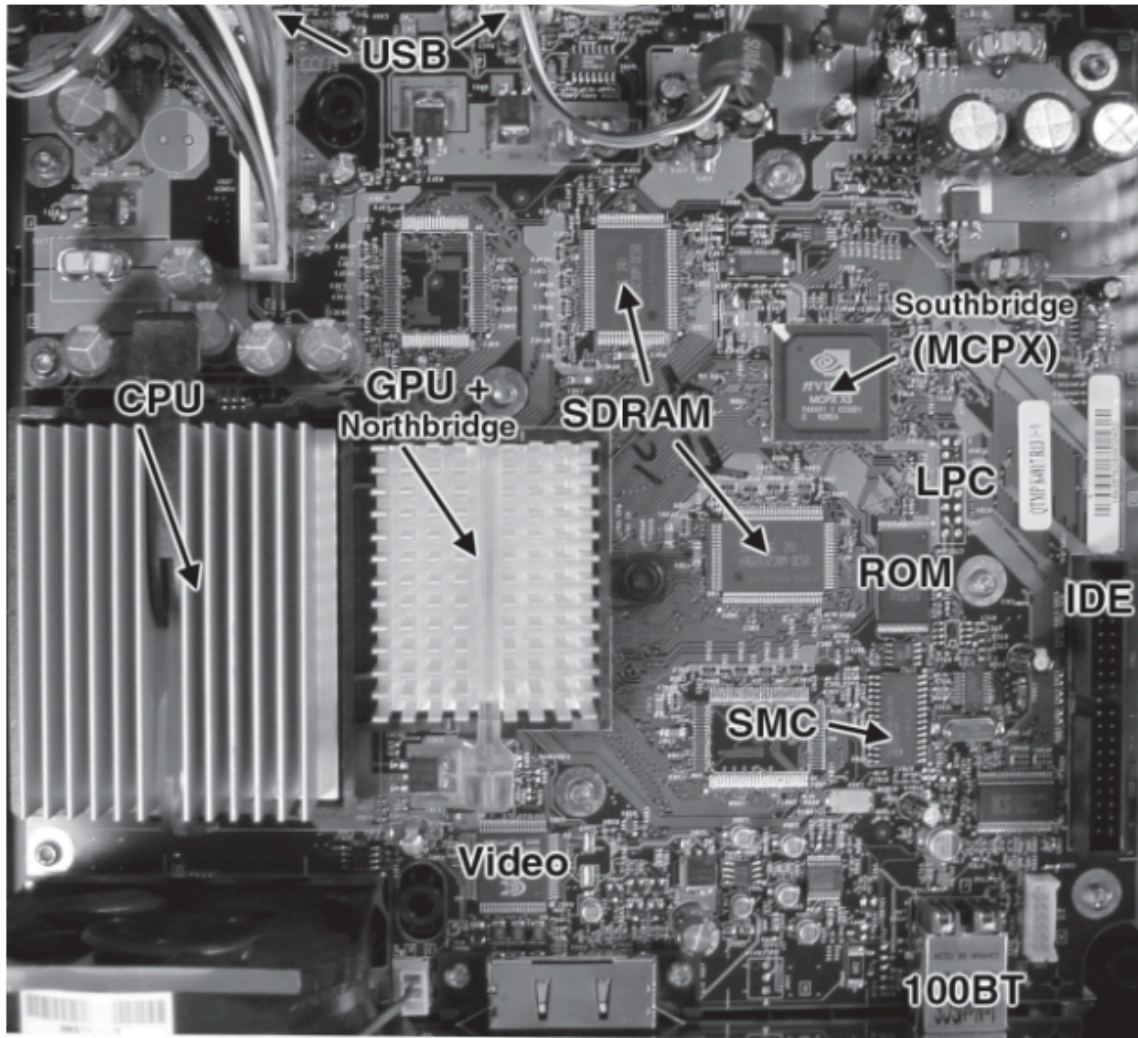


図 2-6: 主要部品にラベルを付けた Xbox マザーボードの写真。

機能の詳細

以下のセクションでは、Xbox アーキテクチャを構成する各部品の概要を簡単に説明する。特に Xbox のセキュリティ機構をリバースエンジニアリングするために必要な詳細に重点を置く。

CPU

CPU (Central Processing Unit、中央処理装置) は従来のコンピュータの計算の中核だ。CPU アーキテクチャというテーマだけで1冊の本に値するほどなので、Xbox をリバースエンジニアリングするための Xbox CPU の制御権を得る方法に絞って説明する。

CPU はメモリに格納された命令の並び——プログラム——を読んで、さまざまな計算を行ったり、データに基づいて判断を下したりする。命令はオペコード (opcode) と呼ばれる数値としてメモリに格納される。オペコードは引数としてオペランドを取る。プログラマーは低レベルのマシンコードを書く際、数百のオペコード番号を覚えなくて済むようにアルファベットのニーモニック (mnemonic) を使う。たとえば、バイト単位の減算命令のオペコードは 0010.1000 (2 進数) または 0x28 (16 進数) で、ニーモニックは「SUB」だ。必要な減算オペコードは減算データのソースと幅によって異なる。すべてのオペコードとオペランドの規則を追いつけるのは大変なので、ニーモニックとオペランドを命令番号に変換するプロセスはアセンブラ (assembler) と呼ばれるプログラムが担う。命令番号をニーモニックに変換するのはディスアセンブラ (disassembler) だ。なお、ほとんどのプログラムはアセンブリ言語では書かれておらず、C などの高水準言語が一般的に使われる。高水準言語はコンパイラ (compiler) を使ってマシン命令に変換される。マシン命令を高水準言語に変換する自動デコンパイルは難しいことがある。コンパイル——特に最適化されたコンパイル——の過程で、元のソースコードに含まれる高水準の構造情報の多くが失われるからだ。

プロセッサは命令ポインタ (IP: Instruction Pointer) と呼ばれるもので現在実行中の命令を追跡する。IP はコンテキストによってはプログラムカウンタ (PC: Program Counter) と呼ばれる。IP は通常、分岐命令に遭遇しない限り、プログラムを 1 命令ずつ進めていく。分岐命令は CPU 内部のデータを調べて新しい場所に分岐する機会をプログラムに与える。命令ポインタの動きを理解することは Xbox をリバースエンジニアリングする上で中心的な作業だ。IP を操作できるということは、Xbox が何をできて何ができないかを制御できることとほぼ同義だ。Xbox のソフトウェアアーキテクチャに実装されているセキュリティ機構は、コードを実行する前に暗号的に真正性を検証することで、IP が常に Microsoft 承認のコードのみを実行することを保証しようとしている。

コラム：2 進数と 16 進数

デジタル回路は 1 と 0 を使って数値を表現する。この 2 進数 (「2 進法」) 表記は、電気信号が数値を表現する方法を反映している。2 つの電圧レベルの範囲を使って一方または他方のロジック状態を定義している。2 つ以上の電圧レベルを使って情報を表現する電気システムを構築することも可能だが、電力と複雑さのコストを伴う。たとえば現代のモデムは複数の電圧レベルと位相情報を使って 1 つの時間単位で複数ビットのデータを表現している。

2 進数での数値の構成と算術は、おなじみの 10 進数 (「10 進法」) 表記と同じ規則に従う。10 進数では、桁があふれたときを記録するために 0 がプレースホルダーとして使われる。たとえば 9 より大きい 1 桁の数字はないため、9 に 1 を加えるとあふれが生じる。したがって 10 は最右の 10 進数の位が 1 回あふれたことを記録している。同様に 2 進数では、2 進法での最大の 1 桁の数字は 1 なので、1 に 1 を加えると 10 になる。

したがって 10 進法では、4 桁の 10 進数 $d_4d_3d_2d_1$ の値は以下のように分解できる：

$$d_4 \times 10^3 + d_3 \times 10^2 + d_2 \times 10^1 + d_1 \times 10^0 = d_4 \times 1000 + d_3 \times 100 + d_2 \times 10 + d_1 \times 1$$

同様に、4 桁の 2 進数 $b_4b_3b_2b_1$ は以下のように分解できる：

$$b_4 \times 2^3 + b_3 \times 2^2 + b_2 \times 2^1 + b_1 \times 2^0 = b_4 \times 8 + b_3 \times 4 + b_2 \times 2 + b_1 \times 1$$

たとえば、2 進数 $1010 = 1 \times 8 + 0 \times 4 + 1 \times 2 + 0 \times 1 = 10$ 進数の 10 となる。

2 進数で数値を追い続けるのはすぐに煩雑になる。10 進数の 968 を表すには 10 桁の 2 進数が必要だ。画面スペースを節約するために、2 進数は 8 進数または 16 進数に変換される。8 進数（「8 進法」）は初期のコンピュータ時代には一般的だったが、今は稀だ。16 進数（「16 進法」）が事実上の標準だ。16 進法には 16 の数字があるため、10 進数の 10～15 に対応する 16 進数は A から F の文字で表される。表 2-2 は最初の 16 個の正の整数の 2 進数・10 進数・16 進数の変換をまとめたものだ。

16 進数を 10 進数と区別するために、多くの人は C 言語の慣習を使う。0x[数値] は 16 進数を表し、[数値] は暗黙的に 10 進数だ。2 進数には C に類似した標準がないため、Verilog の標準 [桁数]'b[数値] を使う人もいる（[桁数] は 2 進数の桁数）。また 1 と 0 の文字列の後ろに「b」を付けた 1010.1100.1110b も 2 進数として使われる。なお、2 進数の桁を 4 桁ずつにグループ化するために「。」が使われており、2 進数を頭の中で 16 進数に変換するのを助けてくれる（= 0xACE）。

2 進	10 進	16 進	2 進	10 進	16 進
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	11	B
0100	4	4	1100	12	C
0101	5	5	1101	13	D
0110	6	6	1110	14	E
0111	7	7	1111	15	F

表 2-2: 2 進数・10 進数・16 進数変換表。

CPU の中核はレジスタファイルと呼ばれる非常に小さいが高速なメモリだ。1 プロセッサクロックサイクルに複数のデータを読み書きできる。レジスタファイルのデータは、算術論理演算装置、つまり ALU (Arithmetic Logic Unit) と呼ばれる演算実行ユニットに送られる。ALU が計算する機能はメモリからフェッチされた命令によって制御される。データが ALU で処理されると、レジスタファイルに書き戻すか、メモリに格納される。

ほぼすべての現代 CPU が持つ重要なパフォーマンス機能として、キャッシュ (cache) と呼ばれるメモリアクセスアクセラレータがある。キャッシュは CPU コアに近い将来に使用しそうなデータや命令のコピーを格納する小型高速メモリだ。キャッシュはレジスタ

ファイルより遅いがメインメモリより速く、レジスタファイルより多くのデータを格納するがメインメモリより少ない。

Xbox CPU のキャッシュについて知っておくべき重要な特徴は、ライトバックキャッシュ (writeback cache) だという点だ。ライトバックキャッシュでは、CPU 内部に格納されたデータのコピーがメインメモリの内容と同期していない状態になることがある。このタイミングの差は、外部メモリのトラフィックを観察するだけで CPU の実行をたどろうとする試みを複雑にする。また、キャッシュメモリはメモリバスを観察している者から中間的な計算結果を隠すためにセキュリティルーティンに活用されることもある。

ノースブリッジとサウスブリッジ

ノースブリッジ (northbridge) とサウスブリッジ (southbridge) という用語は PC アーキテクチャ固有の用語だ。ほぼすべての PC に搭載されている 2 つの基本的なサポートチップを指す。ノースブリッジチップは CPU をメインメモリや AGP・PCI などの高パフォーマンス拡張バスに接続する。サウスブリッジチップはノースブリッジチップからぶら下がるように接続され、一般的な PC に見られるすべての追加周辺機器——パラレル、シリアル、USB、マウス、キーボード、IDE コントローラ、オーディオコーデックなど——を含んでいる。CPU、ノースブリッジ、サウスブリッジという 3 つの主要モジュールに分割することで、PC 設計者はさまざまなメモリアーキテクチャを多様なプロセッサや周辺機器と組み合わせられる。

ノースブリッジとサウスブリッジのチップセット間の接続はチップセットによって異なる。Xbox の場合、HyperTransport と呼ばれる高パフォーマンスの細い並列バスが、ノースブリッジとサウスブリッジに相当する機能の接続に使われている。このバスは各方向 8 ビット幅しかないが、200 MHz でクロックされ各クロックエッジでデータがサンプリングされるため、実効ピーク転送レートは各方向 400 MB/秒だ。ノースブリッジチップはフロントサイドバス (FSB) で CPU に接続される。Xbox の FSB は AGTL+ロジックレベルを使用する 64 ビット 133 MHz バスだ。

リバースエンジニアリングにおいてチップ間の接続の種類を知って理解することは非常に重要だ。接続の種類によって、各部品間のデータを傍受する難しさが決まるからだ。比較的傍受が容易なバスである HyperTransport バスの詳細については、第 8 章「Xbox セキュリティのリバースエンジニアリング」で説明する。

Xbox のサウスブリッジは nVidia 設計の MCPX というチップで、nVidia nForce MCP (Multimedia and Communications Processor) の派生品だ。ノースブリッジチップも nVidia 設計で、NV2A GPU と呼ばれる。どちらも TSMC (Taiwan Semiconductor Manufacturing Corporation、台湾積体電路製造) によって製造された。NV2A は GPU と、ほとんどのノースブリッジチップに見られる従来のメモリ・拡張バスコントローラの両方を統合している。前述のように、グラフィックスプロセッサとノースブリッジを統合することで、システム設計者はグラフィックスメモリをメインメモリに統合でき、そのトレードオフとして幾分かのパフォーマンスが犠牲になる。

RAM

Xbox のマザーボードはメインメモリとして 64 MB の DDR SDRAM を使用している。DDR SDRAM とは Double Data-Rate Synchronous Dynamic Random Access Memory（倍速同期ダイナミックランダムアクセスメモリ）の略だ。同期化と DDR 技術を組み合わせることで、Xbox のメインメモリの合計帯域幅は 6.4 GB/秒を達成している。

RAM は基本的に CPU によってインデックスされる情報のテーブルだ。RAM 内の各位置にはアドレスと呼ばれる固有のインデックス番号があり、「ランダムアクセス」という名称が示すように、データアクセスの順序に制限がない。¹

「ダイナミック」という用語は、データの整合性を保つために常にリフレッシュしなければならない RAM に適用される。たとえば Xbox で使われている RAM はすべての場所を 1 秒間に約 30 回読み出して書き戻す必要がある。現代の DRAM チップにはこのプロセスを最適化する専用ハードウェアが組み込まれているから、パフォーマンスへのペナルティはそれほど大きくない。

「同期」という接頭語は、DRAM 内部でデータアクセスの手順が一連のステップに分割されることを意味する。各ステップは独立していて並列に実行できるため、複数のデータリクエストを同時並行で処理できる。クロックと呼ばれる外部タイミング信号が、DRAM 内部の各ステップを通じたデータアクセスリクエストの移動を同期させる。その結果、データアクセスリクエストはパイプの中を水が流れるように各ステップを通過する。このテクニックはパイプライン処理 (pipelining) とも呼ばれる。パイプライン処理によって複数のリクエストを同時に処理できるため、同期 DRAM はその前身よりも高い帯域幅スループットを持つ。ただし、SDRAM へのアクセスが最初に発行されてからデータが出力に現れるまでの時間——アクセスレイテンシ——はパイプライン処理によって改善されない。

「ダブルデータレート」という用語は、同期データが同期クロックに対してどのように転送されるかを指す。クロック波形は高低信号の繰り返しパターンからなる。従来のシステムではデータはクロック波形の低から高への遷移のときにのみ転送される。DDR システムでは低から高への遷移と高から低への遷移の両方で転送される。したがって同じクロック周波数で 2 倍のデータを転送できる。DDR SDRAM ベンダーが引用するパフォーマンスの目安 (DDR266 など) は転送レートを指すため、実際のクロック速度はその半分——この場合は 133 MHz となる。

ROM

すべてのコンピュータには、スタートアップ (ブート) プログラムを格納するための何らかの不揮発性メモリが必要だ。DDR SDRAM はこの用途には使えない。電源が切れるとすべてのデータが失われるからだ。現行バージョンの Xbox は、電源が切れても消えないデータを格納するためにフラッシュ ROM (FLASH ROM) を使用している。ROM は Read-Only Memory (読み出し専用メモリ) の略で、FLASH は電氣的に再プログラム可能な特定のスタイルの記憶素子を指す。PC ではフラッシュ ROM は便利で、エンドユーザーがブートコードのミスを修正するために再プログラムできる。しかし Xbox では、エ

エンドユーザーによるフラッシュ ROM のプログラミングは意図的に無効化されている。プログラミングに必要な書き込み信号は、Xbox マザーボードの背面の部品位置 R7R4 にあるジャンパを省略することで切断されている（「フラッシュ ROM プログラミングハードウェアの有効化」コラムを参照）。Xbox の場合、フラッシュ ROM の再プログラム可能性は主に開発・製造中の Microsoft の利便性のために利用されている。Microsoft がブートプログラムとカーネルをシリコンに刻み込む準備ができたと判断したら、数ヶ月以内に Xbox はより安価なハード配線の「マスク ROM」を使用するようになる可能性がかなりある。

コラム：フラッシュ ROM プログラミングハードウェアの有効化

Microsoft がインシステムのフラッシュ ROM プログラミングを禁止するために切断した信号にパッチを当てるのは比較的シンプルな手順だ。フラッシュ ROM の書き込み信号は、Xbox マザーボードの底面のセクター 7R に位置する 1 つの抵抗器（部品番号 R7R4）を省略することで切断された。2 つの銀色のパッドの間にワイヤーをはんだ付けするか、大量のはんだでパッドをブリッジするだけでよい。

ただし、このパッチによってハードウェアでフラッシュ ROM プログラミングが有効になっても、実際に再プログラミングを行うプログラムはまだ持っていない。そのようなプログラムを実行することは、Microsoft が設置した暗号的なソフトウェアセキュリティシステムのためにはるかに大きな課題だ。

ブート ROM にはシステム全体の初期化を担う重要なコードが含まれているから、コンピュータのリバースエンジニアリングにおいて極めて重要だ。Xbox の場合、ブートフラッシュ ROM はさらに重要な役割を果たす。厳格なソフトウェアセキュリティシステムの実装を部分的に担っているからだ。セキュリティシステムにおけるフラッシュ ROM の正確な役割については後で説明するが、今の段階で覚えておくべき重要なことは、フラッシュ ROM が Xbox のハードウェアの初期化を制御し、初期のオペレーティングシステムカーネルイメージも格納しているということだ。

その他

Xbox にはシステム管理コントローラ（SMC : System Management Controller）と呼ばれる小型の 8 ビットコプロセッサがある。SMC は RAM、ROM、プロセッサを 1 つのパッケージに収めた完全なミニコンピュータだ。SMC 内部のプロセッサは PIC（Peripheral Interface Controller）アーキテクチャを採用しており、もともと 1975 年頃にハーバード大学で開発され、General Instruments によって商業販売向けに適応されたものだ。Arizona Microchip Technology（現在は Microchip Technology、www.microchip.com）は 1985 年に PIC 製品ラインを買収し、以来販売し続けている。SMC は Xbox のセクター 7B にあり、指定記号は U7B2 だ。

SMC は Xbox 前面の電源ボタンを監視するため、CPU が電源オフのときでも動作していなければならない。そのため、Xbox がプラグインされているときはいつでも低電流の 3.3V 「スタンバイ」電源ラインがアクティブになっている。SMC はまた Xbox の電源ボタン周りのライトの制御と、DVD イジェクト機構の制御も担っている。さらに SMC には

CPU の健全性を監視してクラッシュした場合に CPU を再起動する機能がある。自分のオペレーティングシステムを Xbox で動作させたい場合は、SMC の監視機能を無効にする必要がある。SMC は I²C と呼ばれる 1 ビットシリアルインターフェースを通じて MCPX 経由で CPU と通信する。

Xbox のもう一つの重要な機能は LPC デバッグポートだ。LPC デバッグポートは 33 MHz で動作する 4 ビット幅のバスだ。LPC とは「Low Pin Count (低ピン数)」の略で、もともとはキーボード、シリアルポート、パラレルポート、ブート ROM などの多数の低速レガシーデバイスを、シンプルな中間変換チップを経由してサウスブリッジチップに接続する方法として考案された。デバッグポートはおそらく Microsoft のハードウェアメーカーによる製造テスト目的で Xbox に設けられている。Xbox が製造の最終段階に近づくと、LPC デバッグポートを使って Xbox マザーボードのテスト、診断、バーンインを行うブートプログラムを読み込む。LPC デバッグポートについては第 11 章で詳しく説明するが、今知っておくべきことは、LPC 準拠の ROM デバイスを接続してフラッシュ ROM のデータピン (D0) の 1 本を特定の電圧 (ゼロボルト) にショートさせることで、Xbox が LPC デバッグポートを通じて最初のブート ROM イメージを読み込むよう強制できるということだ。これは Xbox を自分のコードで起動させる最も簡単な方法の一つだ——Xbox を保護している秘密のブートコードを回避する方法を知っている場合に限るが。

パターンマッチング

Xbox アーキテクチャに慣れたところで、最も強力なリバースエンジニアリングツールの一つ——パターンマッチング——のための参考になる。部品の接続性、配置、形状を観察するだけでさまざまな部品の機能について根拠のある推測をできることが、一流のリバースエンジニアになるための第一歩だ。パターンマッチングの力を示すために、Xbox のマザーボードを PC のマザーボードおよびニンテンドーゲームキューブのマザーボードと比較する。

多くのパターンを学ぶことがパターンマッチングの上達には最善の方法だ。僕は購入するすべての機器を分解して、回路基板を熟読することで他の設計者が何を知っているかを「読む」ようにしている。すべての回路基板はその設計プロセスについての物語を語っており、何らかの意図された目的を持たない特異な回路の特徴に出会うことはほとんどない。

[!WARNING] 注意

電子機器を分解する際は、まず電源を抜いて、電源内の大型コンデンサの電荷が放出されるまで 1 分ほど待つこと。また、第 1 章で説明した適切な静電気対策を忘れずに！

比較：Xbox 対 PC

PC プラットフォームは非常によく文書化されているため、Xbox の PC への類似性はハッカーにとって大きな恩恵だ。Xbox のすべての部品には典型的な PC にアナログとなるものがあるため、高レベルの質問には類似した PC 部品について読むだけで答えられる。Xbox のマザーボードと PC の標準的なマザーボードとの類似点をより詳しく見ることに

は価値がある。さらに、本書の情報の多くは PC にも直接適用されるため、Xbox をハッキングして学んだことを多くの場面に応用できる。

Xbox の最も近い仲間は、nVidia nForce や Via Technology の ProSavageDDR のようなユニファイドメモリアーキテクチャを使用するチップセットに基づくシステムだ。前のセクションのアーキテクチャ図は、Xbox と nVidia nForce チップセットの nVidia ウェブサイト上の公開仕様を読むことで得られた。このセクションでは Xbox を Via Technology ProSavageDDR ベースの P4M266 マザーボードと比較する。Xbox を nVidia 以外のチップセットマザーボードと比較するのは、Xbox と PC との広範な類似性を強調するためだ。

図 2-7 は PC マザーボード、Via P4M266 の写真を示している。チップセットが異なるメーカーによって製造されているにもかかわらず、P4M266 と Xbox の類似性は顕著だ。

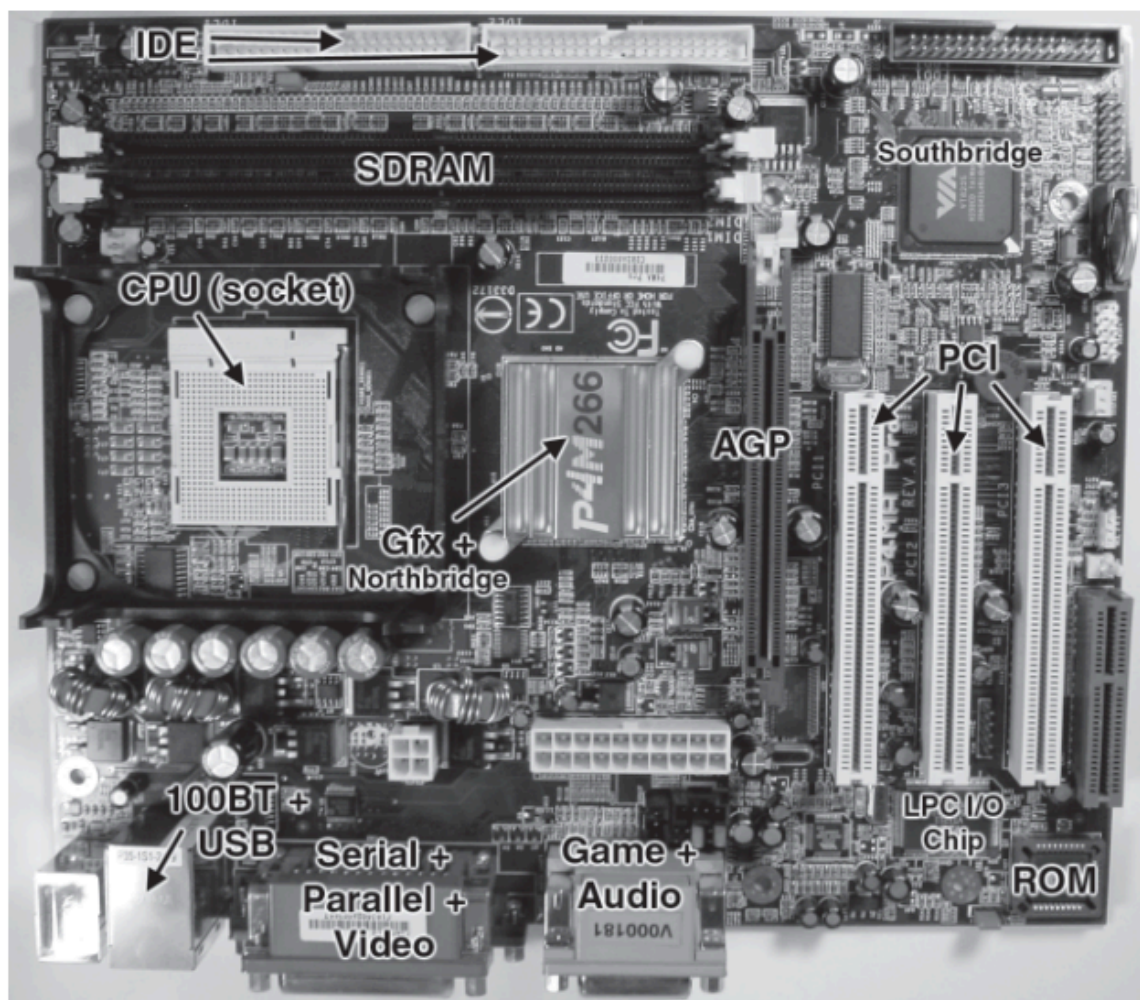


図 2-7: 統合グラフィックスを搭載した Via P4M266 マザーボード。

前のセクションで取り上げたほぼすべての内容がこの PC マザーボードにも適用される。主な違いはいくつかの雑多なポートとコネクタ、そして高パフォーマンスの PCI およ

び AGP 拡張ポートの存在だ。Via P4M266 にはまた、すべてのレガシー周辺機器が LPC マルチ I/O チップによって直接実現されているため、明示的な LPC デバッグコネクタがない。

対比：Xbox 対ゲームキューブ

ニンテンドーゲームキューブは Xbox と比較すると興味深い。ゲームキューブは Xbox と同じ目的——ゲーム——のために設計されたマシンだが、設計思想が大きく異なる。Xbox とゲームキューブはどちらも同じ大まかなアーキテクチャ——CPU、グラフィックスコプロセッサ、メモリ、サポートチップ——を使っているが、類似点はそこで終わる。ゲームキューブの設計は細部とコストへの徹底した注意を示している。ゲームキューブのマザーボードは小型でシンプルで、部品数は最小限に抑えられ、ヒートシンクと熱設計は非常にシンプルだ。ゲームキューブのマザーボードの配線パターンがほぼすっきりとした直線的なレイアウトになっているのは、ほぼすべての IC がゲームキューブ専用カスタム設計されていることの表れだ。その結果、ゲームキューブは Xbox よりもはるかに経済的に製造できるプラットフォームだ。

ゲームキューブの大まかな構成は、任天堂が提供する基本的なマーケティング情報から各チップの機能を推測することで認識できる。ゲームキューブのアーキテクチャのさらなる詳細は推測が難しい。標準的な PC に対応する部品を持たないカスタム部品が多数使われているからだ。マザーボードの配線パターンから、ボードの中央にある大型チップ「Flipper チップ」が PC 内の統合グラフィックスノースブリッジチップに相当するものと推測できる。これはほぼ正しい。重要な違いは、Flipper チップがメモリコントローラとグラフィックスコントローラを1つのパッケージに統合しているにもかかわらず、グラフィックス機能は依然として同じチップの内部に組み込まれた専用メモリを持つという点だ。この構成により、グラフィックスエンジンに非常に高性能なメモリが使用できる。トレードオフとして、チップ外部のメモリを使った場合よりもメモリが少し小さくなる。オンチップメモリの小ささは、極めて高速なチップ外メモリの使用によって部分的に補われている。

ゲームキューブは Xbox のような DDR SDRAM を使用せず、代わりに 1-T SRAM と呼ばれるものを使っている。1-T SRAM は SRAM（スタティック RAM）という非常に高速なタイプのメモリをエミュレートする DRAM メモリだ。SRAM は DRAM よりもランダムアクセスのレイテンシがはるかに低く、DRAM のようにすべてのメモリセルを1秒に30回リフレッシュする必要もない。DRAM が高速 SRAM として振る舞えるようにする実際の仕組みはかなり複雑で本書の範囲を超えているが、1-T SRAM メーカーのウェブサイト www.mosys.com で詳細を確認できる。

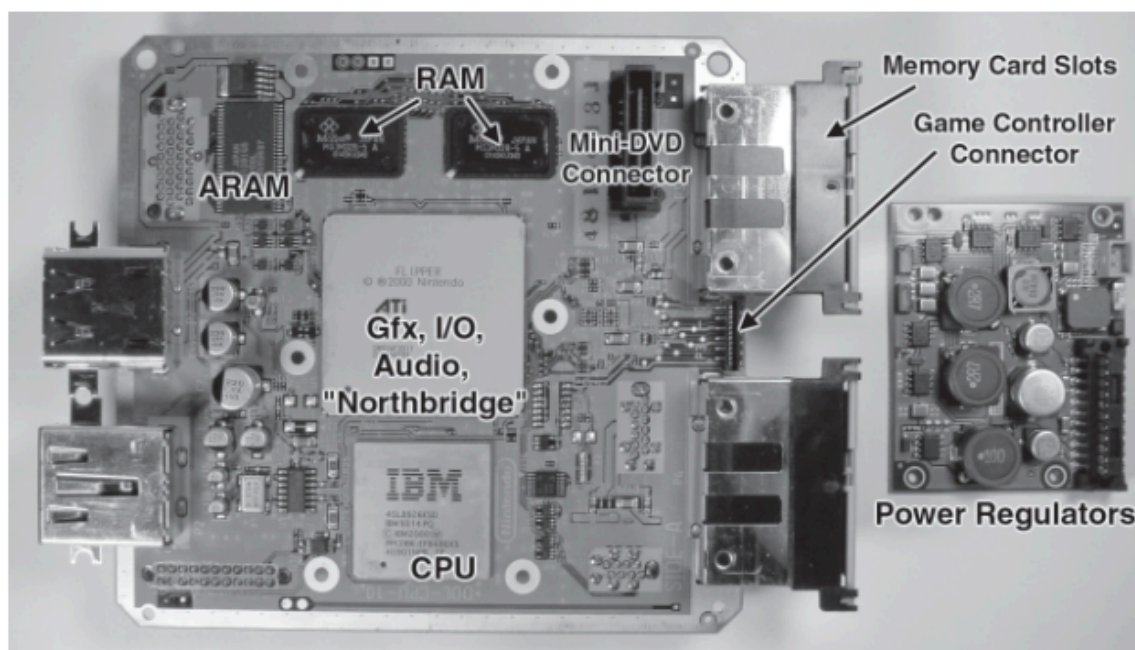


図2-8: ゲームキューブのマザーボードと電源レギュレータカード。マザーボードはXboxのマザーボードの約半分のサイズ。

ゲームキューブにはARAMと呼ばれるもう一つのメモリもある。1-T SRAM よりも遅く、高帯域幅のアクセスを必要としないオーディオサンプルなどを格納するために使われる。異種のメモリアーキテクチャを持つことで、ゲームキューブは各サブシステムからより一貫したパフォーマンスを引き出せる。これはフレームラグを最小限に抑える上で重要だ。そのトレードオフとして、ゲームキューブのプログラムがより難しくなる可能性があり、複数のメモリの管理が適切でないとパフォーマンスの問題が生じる可能性がある。

ゲームキューブと Xbox のもう一つの重要な違いは、ゲームキューブの Xbox よりもはるかに少ない消費電力だ。両コンソールとも電源コンセントに接続して使用するよう設計されているため消費電力は最初は重要に思えないかもしれないが、ゲームキューブの低い電力エンベロップにより、放熱部品が少なく電源も小型で済み、コスト削減につながる。図2-8には参考としてゲームキューブの電源レギュレータの写真が含まれている。電源レギュレータは Xbox の電源供給と Xbox マザーボード上のローカルスイッチングレギュレータを合わせたものに比べて体積が何分の一かにすぎない。

公平のために、ゲームキューブには小型の外部 AC-DC 変換器があるのに対して、Xbox は壁のコンセントから直接コンソールに電源を取ることに注意。また電子部品はアレニウス (Arrhenius) の法則が示すように、温度が高いほど急速に劣化する。たとえば、動作温度が摂氏 10 度上昇すると部品の故障率が約 2 倍になる。発熱が少なく、熱管理システムも Xbox と同等かそれ以上のゲームキューブは、長年にわたって Xbox より信頼性が高いはずだ。

最後に、ゲームキューブはあらゆるところで独自の I/O インターフェースを使っていることは興味深い。ゲームディスクのフォーマットはミニ DVD フォーマットで、DVD リーダーは独自のコネクタでマザーボードに接続されている。より小型の DVD メディアを使用することで任天堂はデータ読み出しのシーク遅延を減らし、ゲームのロード時間を短縮している。ゲームコントローラとメモ리카ードもまた独自のシグナリングフォーマットを使っている。ゲームキューブ内のすべてはおなじみの PC に多少似ているが、設計に変更なしに直接取り込まれたものは何もない。

ゲームキューブの製造性とコストを最適化することに加えて、ほぼ独自のチップと規格を使用することで、コンソールは Xbox よりもリバーズエンジニアリングがはるかに困難になる。たとえば図 2-8 を見るとゲームキューブには明らかな ROM チップがない。したがって、ゲームキューブのコードを調べ始めるだけでも、マザーボード上のいずれかのチップのどこかに隠された ROM を探し出して取り出さなければならない！これはセキュリティを難読化によって実現することが奏功する数少ない例の一つだ。たとえゲームキューブのセキュリティが全くなかったとしても、任天堂のカスタム DVD フォーマットにコードを書き込むコストと労力は、個人の愛好家にとって割に合わないものだ。

注

1. 実際には、SDRAM はパフォーマンス上の理由からメモリアクセスパターンにいくつかの制限がある（ページモードやバーストモードなど）。「ランダム」という名称は、データが厳格な順序規則に基づいてアクセスされる FIFO（First-In, First-Out、先入れ先出し）方式や LIFO（Last-In, First-Out、後入れ先出し）方式のメモリと区別するためのものだ。

第 3 章 青色 LED を取り付ける

Xbox のカバーを外したところで、いよいよ最初のプロジェクトに取りかかりよう。この章と次の章では、Xbox に対して自分で行える基本的な改造や修理をステップバイステップで説明する。ここで紹介するプロジェクトは、ハードウェアハッキングの経験がほとんどない読者を対象に設計・執筆している。より高度なトピックは後の章で扱う。

本章では、Xbox フロントパネルに通常取り付けられている緑色の発光ダイオード（LED：Light-Emitting Diode）を青色 LED に交換する方法を説明する。このプロジェクトでのはんだ付け作業は最小限で、作業の大半はフロントパネルと LED 回路アセンブリの取り外しに費やされる。では始めよう！

[!NOTE] 市販の Xbox には緑色/赤色の二色 LED が搭載されているが、赤色 LED はエラー表示にのみ使われる。本章の手順を実施すると、Xbox のフロントパネルインジケータは青色のみの LED になる。本来の代替品は青色/赤色の二色 LED（T-1 パッケージ、レンズ径 3 mm）だが、入手が困難なため本書の手順では使用しない。ここで紹介する手順から十分な基礎知識が得られるので、好みに応じて独自の LED を組み合わせて対応することも可能だ。

必要なもの

このプロジェクトを完了するために必要な機材は以下の通りだ：

- ・ 細先チップ付きの低ワットはんだごて
- ・ はんだ
- ・ フラックスとはんだごてチップクリーナー（任意）
- ・ 小型の平ドライバー
- ・ T-10 トルクスビットドライバー
- ・ 低電圧（3 ボルト）の青色 LED（T-1 パッケージ、3 mm）×2 個
- ・ はんだ付け中に部品を固定するためのマスキングテープ

LED の色は好みのものに代えてもよいが、約 3 ボルトで点灯し、T-1 スタイルのケースを持つものでなければならない。青色や白色の LED には 5 ボルト専用のものが多いので、購入時には定格電圧に注意すること。本書で使用する LED は Lumex SSL-LX3044USBC で、Digi-Key (www.digikey.com) から購入できる（部品番号：67-1747-ND）。予算を抑えたい場合、Digi-Key は米国郵便局のファーストクラスメールでの発送にも対応しているが、25 ドル未満の注文には 5 ドルの手数料が加算される点に注意。

Digi-Key の最低注文額が問題になる場合は、Mouser Electronics (www.mouser.com) でも青色 LED を扱っており、最低注文額の制限がない。例えば Kingbright の T-1 ケース・水クリアレンズ仕様の青色 LED がある——やや高電圧で明るいバージョンの Mouser ストック番号は 604-L7104PBC/H、低電圧・低輝度バージョンは 604-L7104QB/D だ。

エラー表示 LED の機能も維持したい場合は、二色 LED を使うこともできる。Xbox には共通カソード（common-cathode）の三端子 T-1 パッケージ二色 LED が必要だ。ただし、T-1 サイズの小型パッケージで青色素子入りの二色 LED は非常に入手しにくい。

フロントパネルの取り外し

Xbox のフロントパネルは ABS プラスチック製の成形部品で、4 本の T-10 トルクスネジと 3 つの成形摩擦ロック（friction lock）で固定されている。フロントパネルの電子回路は、金属製の電磁干渉シールド（EMI シールド）の穴を通る 9 芯の単一コネクタで Xbox マザーボードに接続されている。

第 1 章の手順に従って Xbox を開ける。ハードドライブと DVD ドライブを少し持ち上げて Xbox 後方に移動させ、Xbox マザーボードの前端が露出するようにする。ドライブのケーブルは外す必要はない。図 3-1 に、この手順が完了した Xbox の状態を示す。

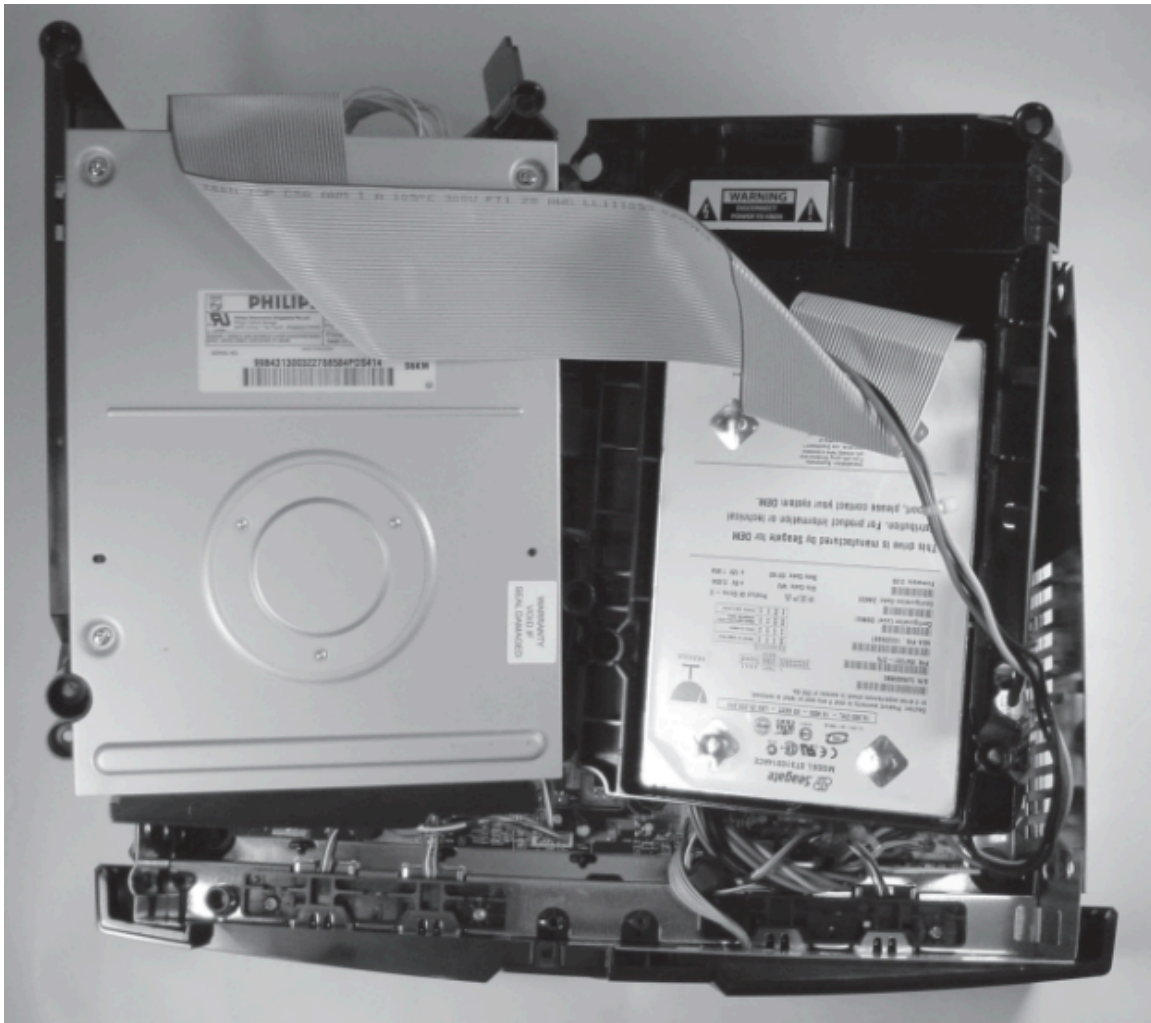


図 3-1: マザーボードの前端が露出するようにディスクドライブを移動した状態。

[!NOTE] 古い Xbox モデルには、Xbox 前部付近に垂直に取り付けられた PC ボードがある。この PC ボードはつかんでソケットから引き抜くことで取り外せる。フロントパネル中央の摩擦ロックを外す際に、先に垂直 PC ボードを取り外しておく作業しやすい場合がある。作業が終わったら必ず PC ボードを元に戻すこと！

フロントパネルアセンブリを固定している4本のネジを外す(図3-2参照)。次に、図3-3に示すようにフロントパネルのワイヤーコネクタを Xbox マザーボードから外す。コネクタは強く均一な力で引けば外れる。急に引き抜くとワイヤーが傷む可能性があるがあるので、ゆっくり引くこと。

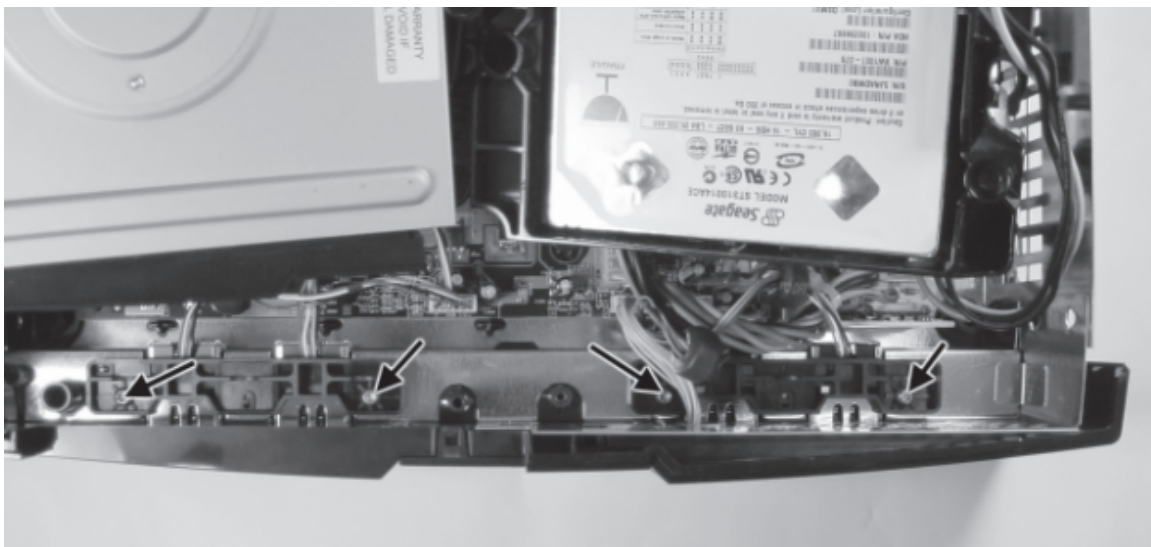


図 3-2: フロントパネルアセンブリの 4 本の固定ネジの位置。

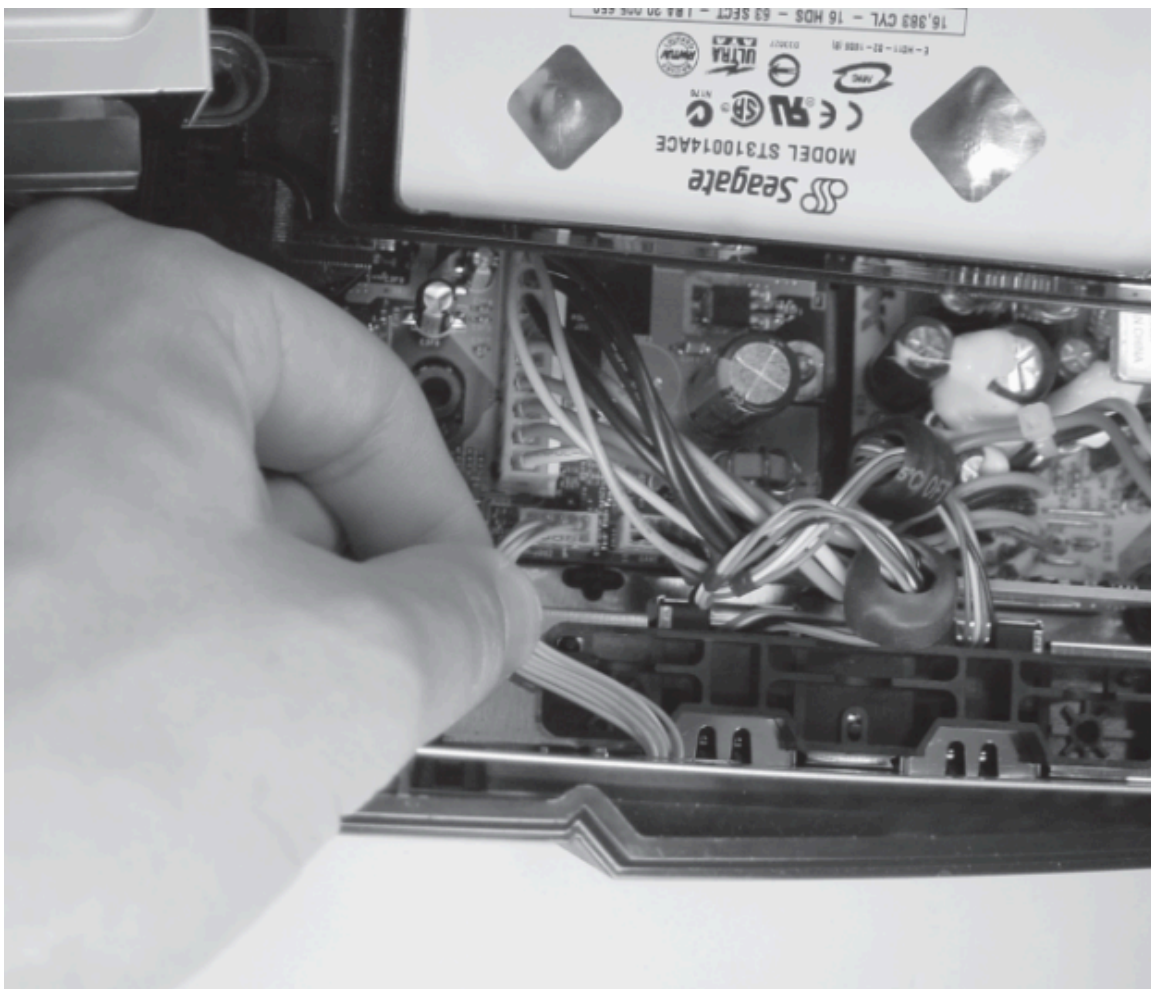


図 3-3: フロントパネルのワイヤーコネクタを Xbox マザーボードから取り外す。

次は少しコツが要る摩擦ロックの解除だ。摩擦ロックとはプラスチック製の爪で、部品同士を固定する。差し込みは簡単だが、取り外しは難しい形状になっており、解除するにはプラスチックを曲げるか押す必要がある。

フロントパネルは3つの摩擦ロックで固定されている——フロントパネル両端に1つずつと、金属 EMI シールドを貫通した中央に1つだ。まず、図 3-4 に示すように薄刃の平ドライバーを使って両端の摩擦ロックを緩める。このロックはかなり固いので、上部から少しずつ解除していく必要がある。パネルとケース本体の側面のすき間にドライバーの先を入れ、少しゆるむまでこじる。ドライバーを抜いたら、ケース下部でも同じ作業を繰り返す。何度か試してようやくロックが外れる場合もある。

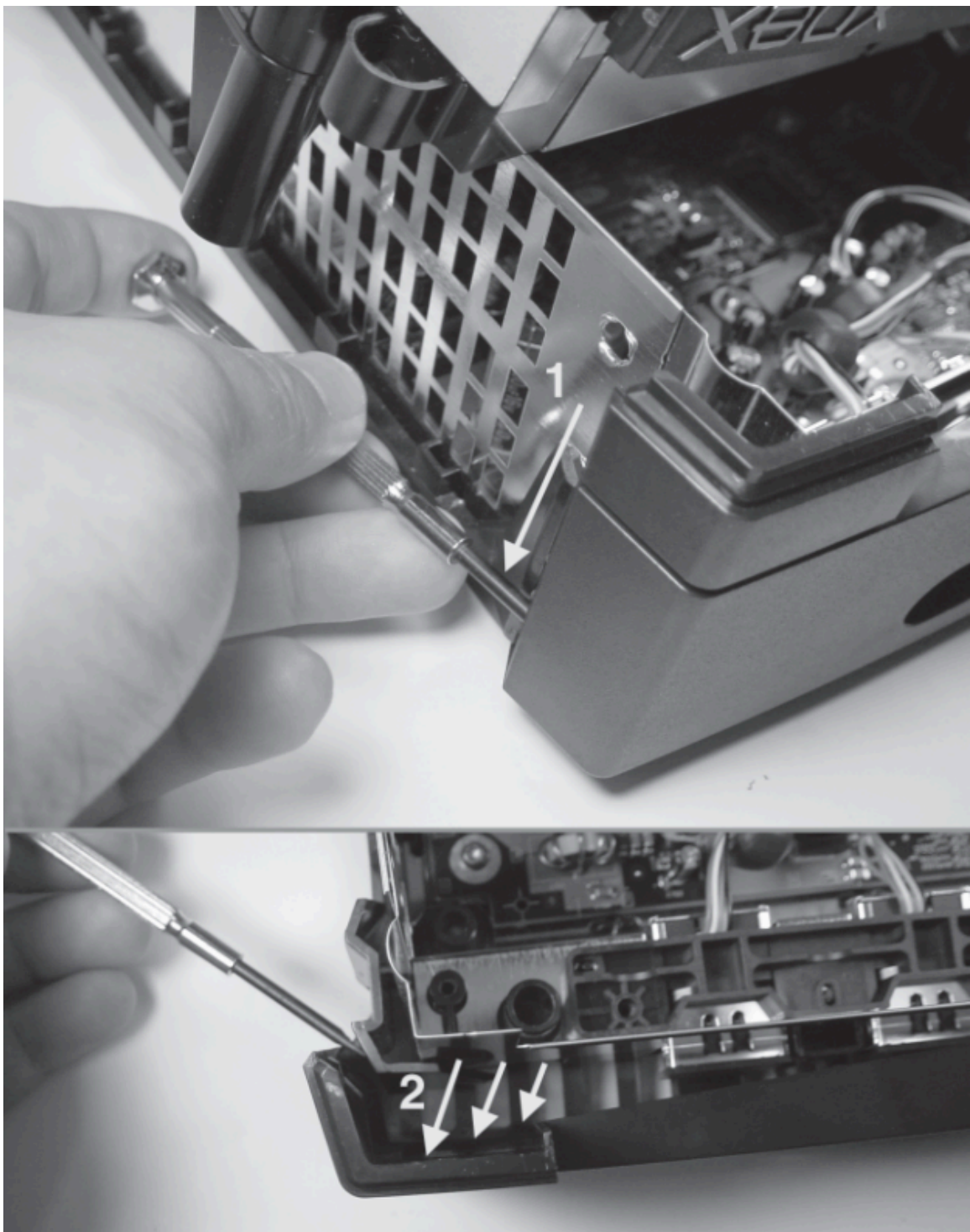


図3-4: 平ドライバーで端の摩擦ロックを緩める。(1) 上部から作業を始めて下へ進む ;(2) パネルの端が外れたらケースから外側に反らせる。

ケースに無理な力をかけないこと。プラスチックが割れたり傷ついたりする可能性がある。フロントパネルの端が外れたら、ケースから外側に反らせることができる。両端でこの作業を繰り返す。

両端が外れたら、図 3-5 に示す中央の摩擦ロックを上にはくとフロントパネルがポンと外れる。

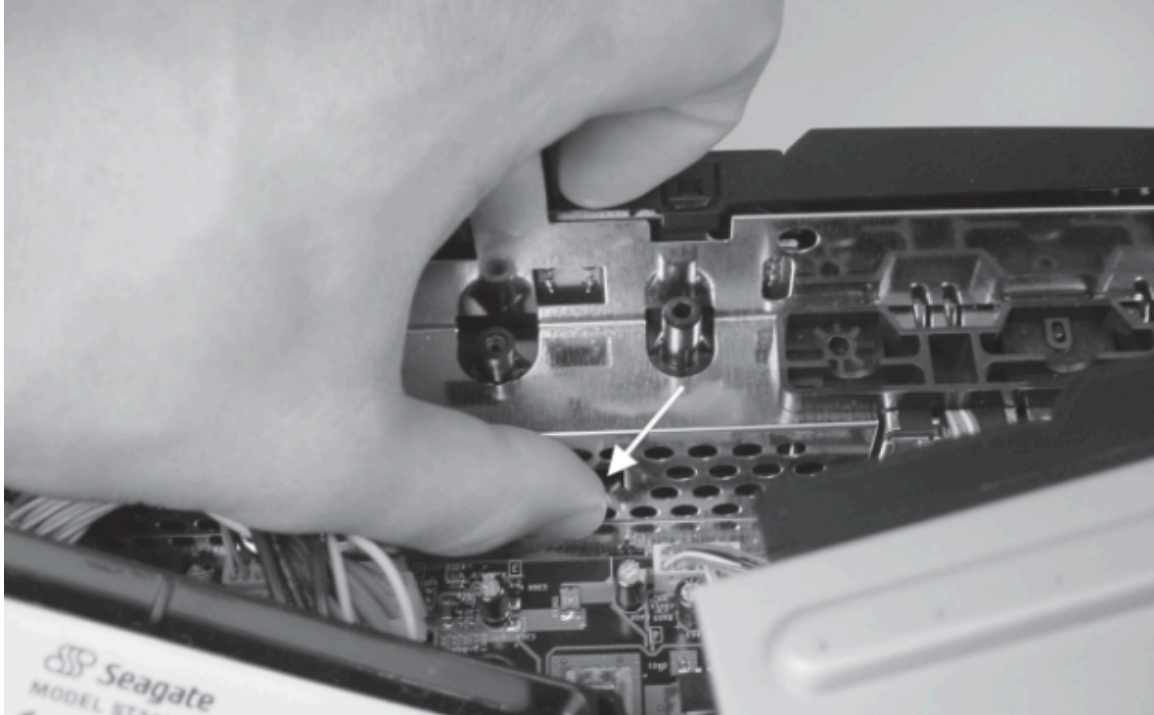


図 3-5: 親指で中央の摩擦ロックを押している様子。

フロントパネルが外れたら、フロントパネルのワイヤーコネクタを金属 EMI シールドの穴に通し、パネルの外側を下にして机に平らに置く。

フロントパネル回路基板の取り外し

Xbox のフロントパネルアセンブリには小さな回路基板が含まれており、図 3-6 に示す単一の摩擦ロッククリップで固定されている。指かドライバーで摩擦ロックを押し下げ、フロントパネルの回路基板アセンブリをホルダーから引き抜く。

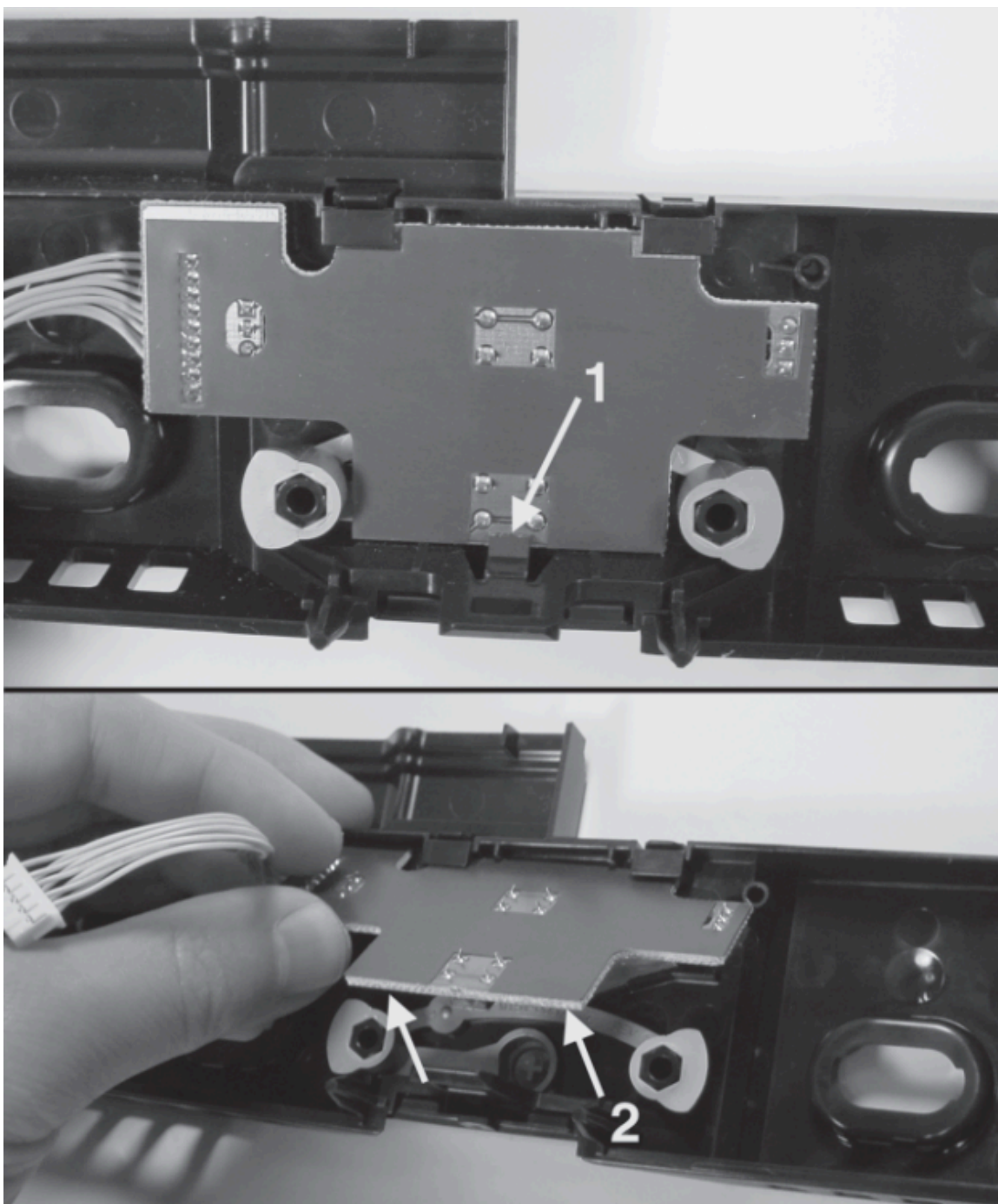


図 3-6: フロントパネルから基板アセンブリを取り出す。(1) 必要に応じてドライバーを使って摩擦ロックタブを押し下げ、(2) アセンブリをフロントパネルから持ち上げる。

回路基板アセンブリを緑色の面を下にして机に置くと、透明の LED が 2 つと平らなプッシュボタンスイッチが 2 つ見えるはずだ。

青色 LED の取り付け

回路基板アセンブリを取り出したら、いよいよ青色 LED を取り付ける。はんだ付けが必要なので、はんだごてのプラグを差し込んで加熱しておく。細先のチップを使い、使用前にチップクリーナーでチップを整えることが大切だ。機材が必要な場合は付録 A「ハッキング機器の入手先」を参照してほしい——ビデオゲーム 1 本よりわずかに高い程度の予算で揃えられる。

フラッシュカッターで既存の LED を両方とも切り取る。後で青色 LED の取り付けに使うので、LED から出ている金属の脚はできるだけ長く残しておくこと。図 3-7 に、作業が完了した後の回路基板の状態を示す。

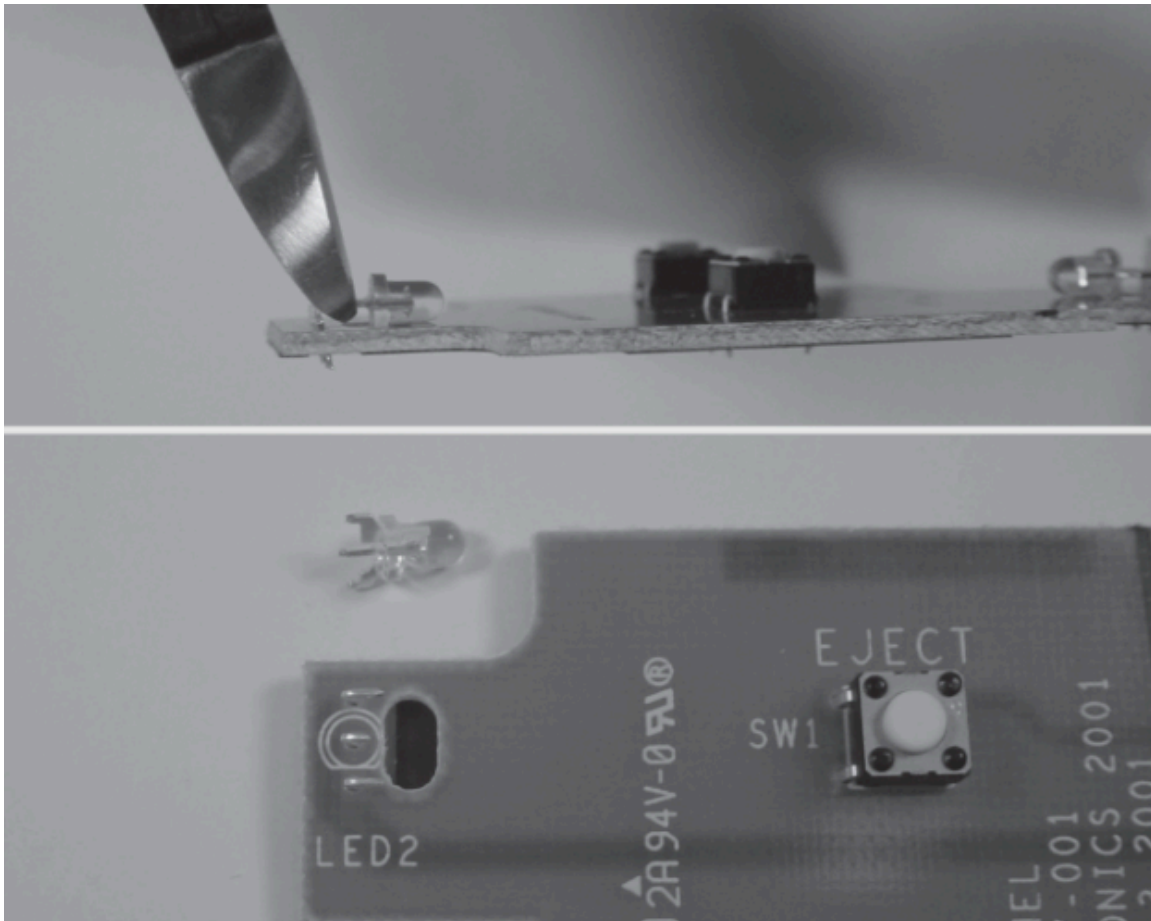


図 3-7: 既存の LED を回路基板アセンブリから切り取る。(上) LED をケースにできるだけ近いところで切る；(下) LED を取り外した状態。両方の LED でこの手順を行う。

はんだ付けをしやすいように、回路基板をマスキングテープで平らな面に固定する。青色 LED を、その脚が回路基板アセンブリ上の元の LED の金属スタブ（脚の残り）に触れる位置に置く。図 3-8 は、LED の極性の確認方法と回路基板上の正しい向きを示している（LED の極性の確認方法が分からない場合は、コラム「LED の解剖学」を参照してほ

しい)。はんだ付け中に LED が転がらないよう、マスキングテープで LED のレンズ部分を固定する。基板右側に取り付ける LED は、黄色のワイヤーコンネクタが邪魔になるため、脚を曲げるか切断する必要がある。

[!WARNING] LED の極性には十分注意すること。逆向きに取り付けると光らない。極性の確認方法が分からない場合は、コラム「LED の解剖学」を参照してほしい。

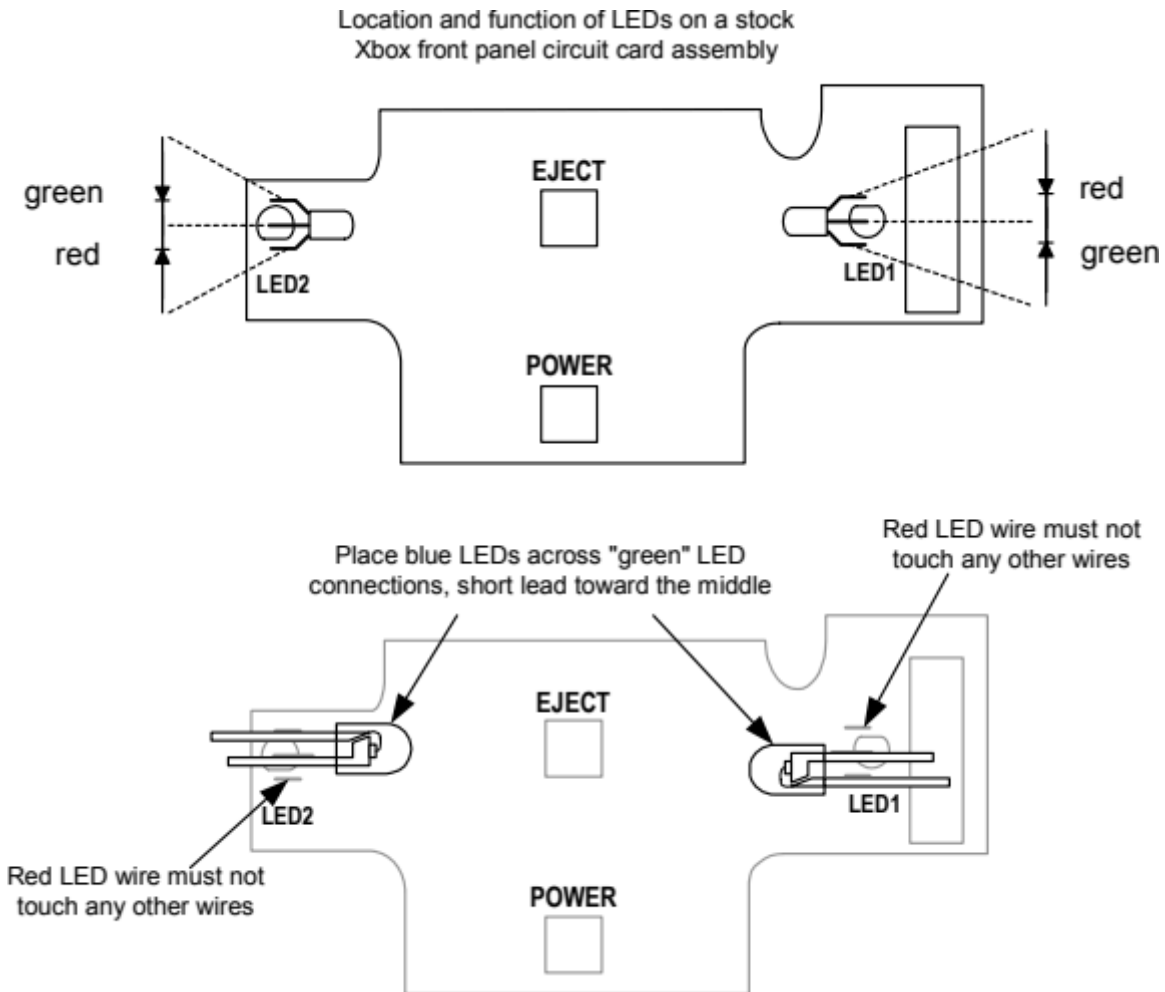


図 3-8: フロントパネル回路基板上での青色 LED の配置。回路基板の両側で LED の色の配置が反対（アンチシンメトリ）になっていることに注目。

図 3-8 は、標準 Xbox の LED の極性と機能も示している。挑戦心のある読者は、複数の LED や表面実装 LED パッケージを試して、より多くの色と機能を実現することに挑戦してみるとよい。LED の縁を先にサンドペーパーで削ることで、Xbox の T-1 パッケージよりわずかに大きい LED も取り付けることができる。

両方の LED の極性を再確認し、両方の LED の短い脚が元の LED の中央脚の残りに当たっていることを確認したら、LED をはんだ付けする。図 3-10 に、LED をはんだ付けし

ている様子を示す。はんだ付けの経験がない場合は、先に付録 B「はんだ付け技術」を読んでおくとよい。

はんだごてを使う前に、はんだ線を少し溶かしてチップが十分に熱くなっているか確認する。チップが十分に熱ければ、はんだ線は瞬時に溶けるはずだ。はんだごてが冷たすぎると良い接合部を形成できず、回路基板を傷めるリスクがある。

熱いはんだごてのチップを青色 LED の脚に当て、脚を回路基板アセンブリの金属スタブに押しつける。脚が熱くなっている間に、青色 LED の脚と金属スタブが接する点にはんだ線をわずかに当てる。溶けたはんだの表面張力によって、LED の脚と金属スタブにはんだが濡れ広がる。うまくいかない場合は、はんだごてを離してフラックスを少し接合部に塗り、やり直す。

金属スタブにはんだごてのチップを長く当てたままにしないこと——スタブを基板に固定しているはんだが溶けてしまう。スタブが自由にぐらつき始めたら、スタブと基板の接合はんだが溶けているサインだ。そうなったら、はんだごてのチップを基板に当てたままゆっくり引き離す。ゆっくり引くことで、スタブがはんだごてにくっついて基板から抜けてしまうのを防ぐ。スタブが冷えるのを待ち、正しい位置に曲げ直す。

コラム：LED の解剖学

LED は極性を持つ素子で、電流は一方向にしか流れない。逆向きに取り付けると動作しない。図 3-9 は LED の構造を示している。短い脚がある側（パッケージに小さな平らな部分がある側）をカソード（cathode、負極）と呼ぶ。LED が機能するためには、カソードをもう一方の脚——アノード（anode、正極）——より低い電位に接続しなければならない。

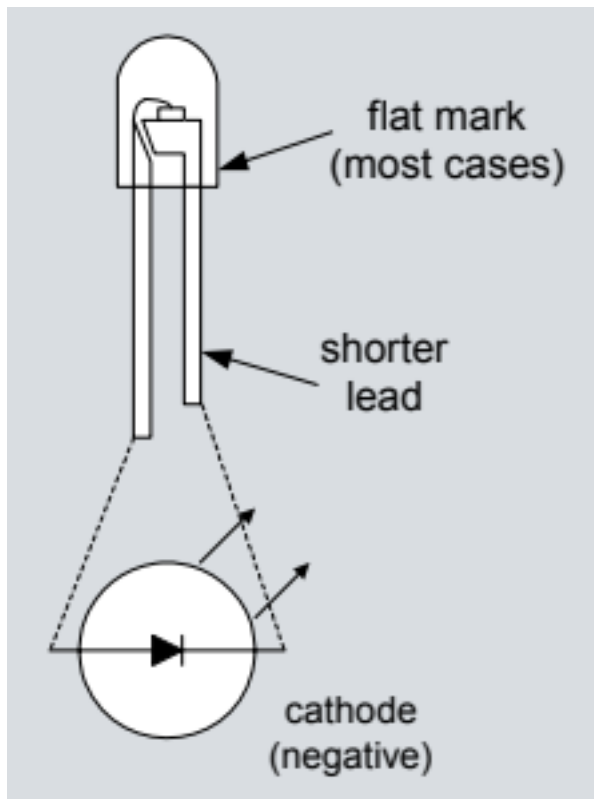


図 3-9: LED の構造。短い脚と平らな部分がカソード（負極）を示している。

LEDの種類によって点灯に必要な順方向電圧は異なる。赤色LEDは通常1.7ボルト、緑色LEDは約2.1ボルト、青色LEDは3.5ボルト以上を必要とする。初期の青色LEDは約5ボルトを必要としたが、技術の進歩によって電圧が下がり、電池駆動や低電圧電子機器にも組み込みやすくなった。このプロジェクトでLEDを選ぶ際は、必要な順方向電圧に注意すること。5ボルト仕様の青色LEDを取り付けると、Xboxのドライバが生成できる最大順方向電圧が約3ボルトのため、光量が非常に暗くなってしまう。



図 3-10: LED はんだ付けしている様子。基板と LED がマスキングテープで固定されていることに注目。

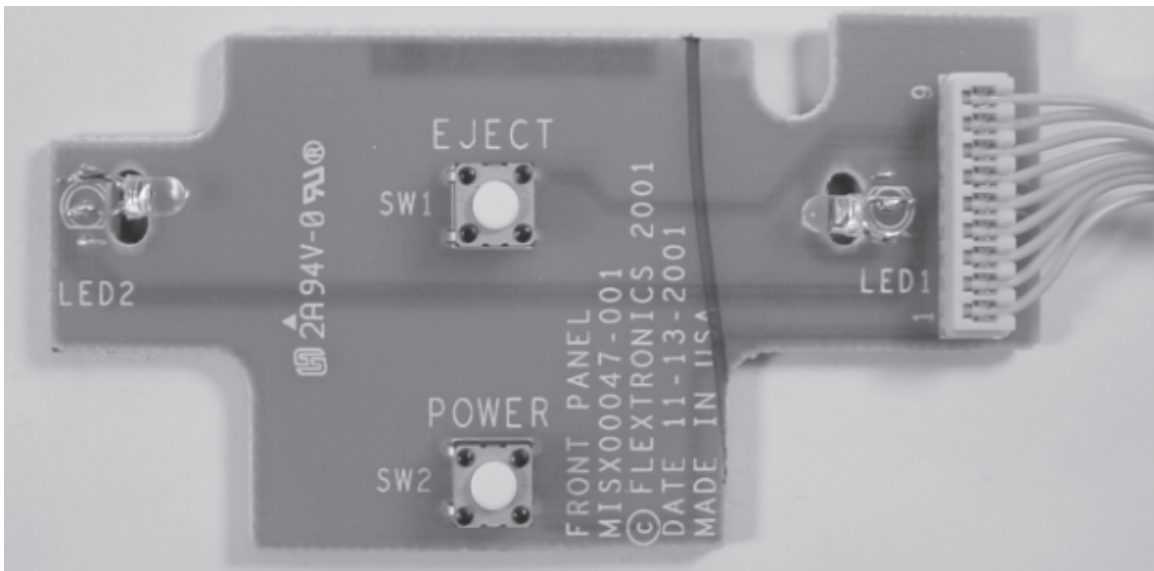


図 3-11: 完成した基板アセンブリ。

2 個の LED の計 4 か所のはんだ付けが完了したら、金属スタブを切らないよう注意しながら余分な LED の脚をできるだけ短く切り取る。完成した基板は図 3-11 のような状態になるはずだ。

フロントパネルの再組み立て

回路基板アセンブリの上端を保持クランプの下に合わせ、基板を摩擦ロックに押し込んでフロントパネルに戻す。

フロントパネルアセンブリ全体を Xbox に取り付ける。まず、ワイヤーコネクタを EMI シールドの元の楕円形の穴に通す。次にフロントパネルを Xbox に押し込めば、3つの摩擦ロックすべてがカチッとハマるはずだ。

フロントパネルのワイヤーコネクタを Xbox マザーボードに再接続する。コネクタは一方向にしか差し込めない形状になっているが、コネクタの成形に使われているプラスチックは柔らかいため、強引に押し込むと逆向きにも入ってしまい、取り返しのつかないダメージが生じる可能性がある。図 3-12 に示すように、基板側レセプタクルの欠けているピンの位置とコネクタヘッダの欠けているワイヤーの位置を合わせて、正しい向きを確認すること。

古い Xbox の場合は、垂直回路基板アセンブリをソケットに押し込んで再取り付けする。この PC ボードは Xbox ゲームコントローラのインターフェースカードなので、正しく取り付けることが必須だ。

ドライブをベイに戻し、電源ケーブルとリボンケーブルの接続が確実であることを確認する。Xbox のプラグを差し込むと、フロントパネルから青い光が放たれるはずだ。期待通りの結果が得られなくても焦らないこと——次の「デバッグ」セクションで、考えられる問題の対処法を説明している。

改造に満足したら、Xbox をオフにしてフロントパネルアセンブリの4本の固定ネジを取り付ける。ネジ穴にアクセスするためにドライブを再度取り外す必要がある。ドライブを戻して Xbox を再起動し、すべてが正常に動作することを確認したら、第1章の手順に従って Xbox を組み立て直す。

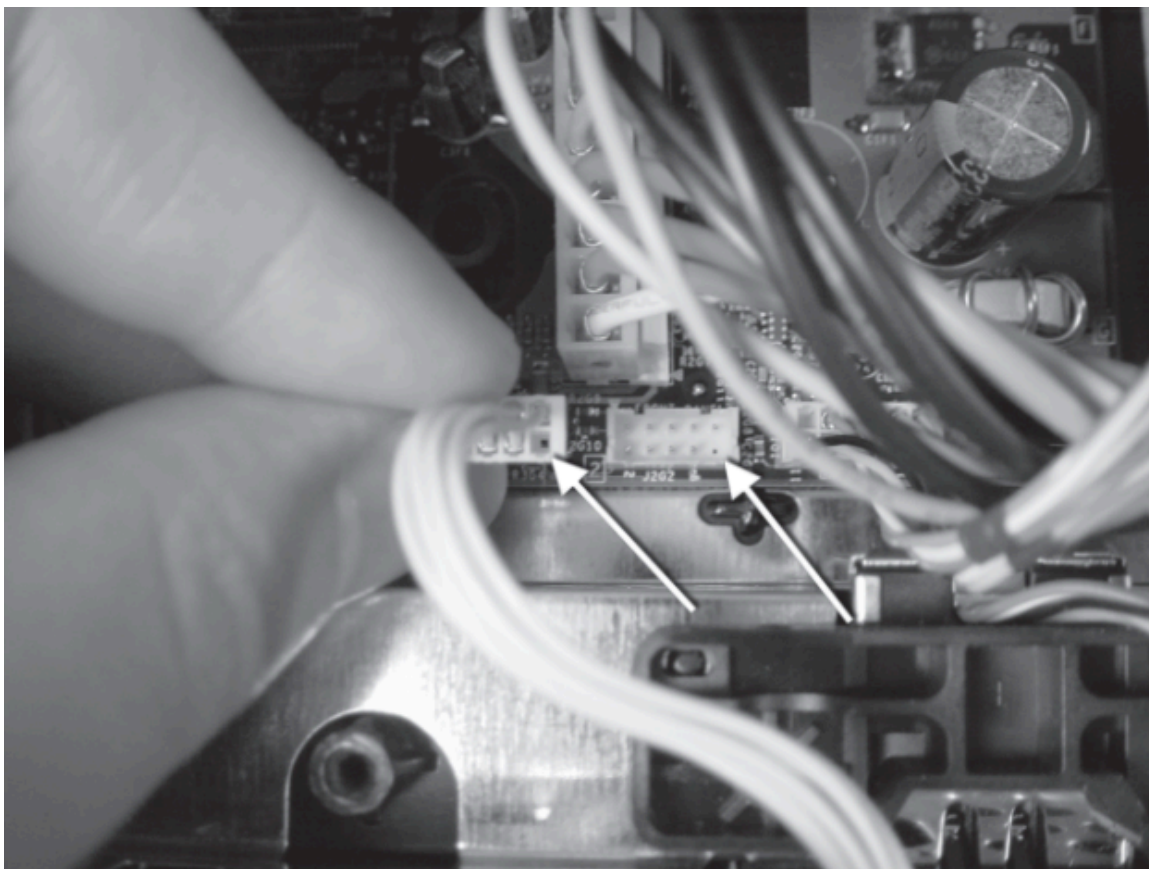


図3-12: フロントパネルワイヤーコネクタの向きを示す特徴。矢印は空きの極性確認ピンの位置を示している。

デバッグ

うまくいかないことは誰にでも起こる。僕の経験では、むしろ何か起きる方が普通だ。うまくいかないときに最も大切なことは、焦らないことだ！冷静を保ち、何が起きているかを観察し、不具合の原因を仮説立てて考えてみよう。参考として、表3-1によくある問題とその考えられる原因をまとめた。デバッグ技術のより詳しい説明は付録E「デバッグ：ヒントとコツ」を参照してほしい。

現象	考えられる原因
Xbox の電源が入らない	フロントパネルのワイヤーコネクタが正しく差し込まれていない。Xbox のプラグが差し込まれていない。フロントパネルのワイヤーコネクタまたは回路基板アセンブリが破損している。

Xbox の電源が入り正常に動作するが、LED が光らない、またはイジェクトボタン周辺の光の輪が半分しか点灯しない	1 個以上の LED が逆向きに取り付けられている。1 個以上の LED が緑色 LED の金属スタブではなく赤色 LED の金属スタブに取り付けられている。ビデオケーブルを抜いた状態で Xbox を起動することで確認できる——この状態では Xbox が赤色と緑色の LED 両方に点滅信号を送るため。
Xbox の電源が入り起動アニメーションが表示されるが、コンソールがサービスが必要だと示す	ハードドライブまたは DVD のコネクタが外れている。グレーのリボンケーブルヘッダが各ドライブに完全に差し込まれているか、各ドライブの電源コネクタが完全に差し込まれているかを確認すること。古い Xbox モデルの場合は、垂直のゲームコントローラインターフェースボードが正しく再取り付けされているかを確認すること。
Xbox の電源が入るが、ゲームコントローラが反応しない	古い Xbox モデルの場合は、垂直のゲームコントローラインターフェースボードが正しく再取り付けされているかを確認すること。
Xbox の電源が入るが、DVD がイジェクトされない	フロントパネルと Xbox マザーボード間のワイヤーコネクタが正しく差し込まれているかを確認すること。DVD の電源コネクタが正しく差し込まれているかを確認すること。

表 3-1: 青色 LED 取り付けのデバッグガイド。

第 4 章 USB アダプタを作る

ケーブル製作は、ハードウェアハッキングには欠かせない作業だ。ハードウェアへの改造や実験の多くで、既存のコネクタを目的に合わせるためのカスタムケーブルが必要になる。

本章では、Xbox 用 USB アダプタケーブルの製作方法を学ぶ。このケーブルは一般の小売店ではほとんど手に入らないが、Lik-Sang (www.lik-sang.com) のような一部のオンライン業者では取り扱っている。USB アダプタがあれば、標準的な USB ハブ、キーボード、マウスを Xbox に接続して Linux を動かすことができる。(本章は、丈夫なケーブルを製

作するための基本をチュートリアル形式で解説している。経験豊富なハードウェアハッカーは読み飛ばしても構わない。)

用意するもの

このプロジェクトには以下の材料が必要だ：

- Xbox ゲームコントローラーのブレイクアウェイケーブル交換品またはゲームコントローラー延長ケーブル (図 4-1 参照)
- USB Type A 延長コードまたは USB Type A メスソケット (図 4-2 に USB Type A メスコネクタを示す)
- はんだごて、はんだ、フラックス
- 斜めニッパーとワイヤーストリッパー
- 絶縁テープ
- (オプション) 3/8 インチの熱収縮チューブとホットグルー
- (オプション) ワニ口クリップ付きの「第三の手」ツール

本章の手順では、ブレイクアウェイ交換ケーブル (ゲームショップで入手可能) と USB 延長コードを使用する。別のコネクタを使いたい場合は、付録 F「Xbox ハードウェアリファレンス」に Xbox の各種コネクタのピン配置が載っているので参照してほしい。



図 4-1: (左) Xbox ゲームコントローラー延長ケーブル。(右) Xbox ブレイクアウェイケーブル。

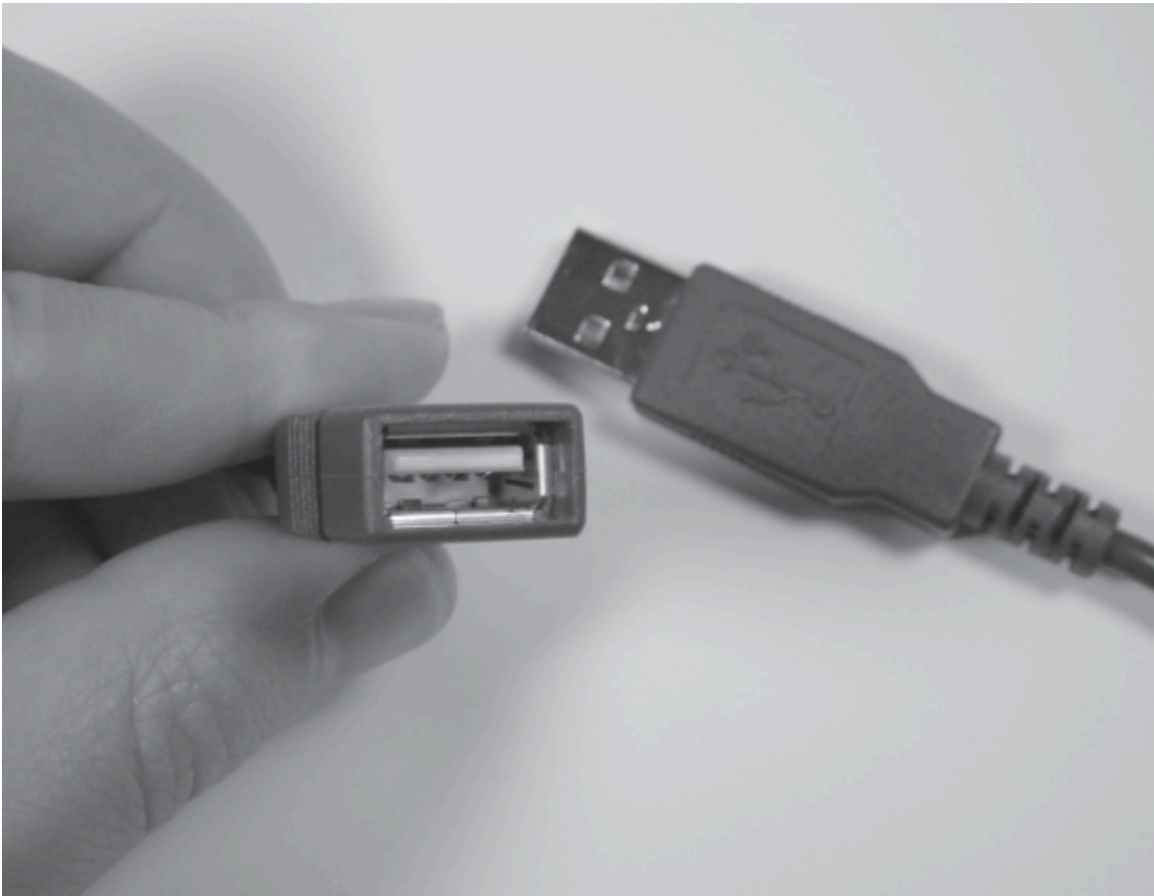


図 4-2: USB Type A メスコネクタ。

方針

Xbox の USB アダプタケーブルを作る基本的な考え方はシンプルだ。Xbox ブレークアウェイケーブルと USB 延長ケーブルをそれぞれ半分に切断し、適切な端同士をつなぐ。幸い、USB 準拠ケーブルには標準的な配線コードがある。赤が+5V 電源、黒がグラウンド (GND)、白が Data (-)、緑が Data (+) だ。つなぎ方は単純で、同じ色のワイヤー同士をつなぐだけでいい。(Xbox ケーブルには、ライトガン型ゲームインターフェース用にコンポジットビデオ同期信号のコピーを運ぶ黄色いワイヤーが1本余分についている。この黄色いワイヤーは無視しても問題ない。) 本章では、ワイヤーの接続とケーブルの封止を段階的に説明する。

[!WARNING] USB ケーブルメーカーの中には、標準的な USB 配線コードに従っていないものがある。最もよくある違反は白と緑のワイヤーの入れ替えで、まれに色コード自体が全く無視されている場合もある。導通テスターを使って、USB 延長ケーブルが実際に USB カラーコードに準拠しているかを必ず確認しよう。

実装

まず、Xbox ブレークアウェイケーブルを Xbox 側コネクタの端から約 5cm (2 インチ) のところで切断する。USB 延長ケーブルはメスコネクタの端近くで切断する。USB 延長ケーブルのオス側の半分と、Xbox ケーブルの小さいコネクタがついている半分は捨てていい。

次に、斜めニッパーで各ケーブルの外皮被覆に約 12 mm (1/2 インチ) の切り込みを入れる (図 4-3 の手順 1 参照)。被覆をめくると、金属の編組シールドと金属箔で保護されたワイヤーが現れる。編組と箔をめくって、余分な被覆とシールドを切り取る。赤・緑・白・黒のワイヤーの先端を、裸線が約 3 mm (1/8 インチ) 見える程度に剥く。(Xbox ブレークアウェイケーブルの黄色いワイヤーは剥かないこと。) 剥いた導線の端をフラックスに浸す (図 4-3 参照)。

アダプタケーブルの強度や安全性が気になる場合は、ケーブルの一方に約 32 mm (1-1/4 インチ) の熱収縮チューブをあらかじめ通しておく。(このチューブは後ではんだ接合部の上に被せ、ホットグルーで充填して丈夫な接続を作るために使う。) 熱収縮チューブによる補強は必須ではないが、施さないとケーブルの強度が落ち、繰り返しの曲げや引っ張りで断線しやすくなる。

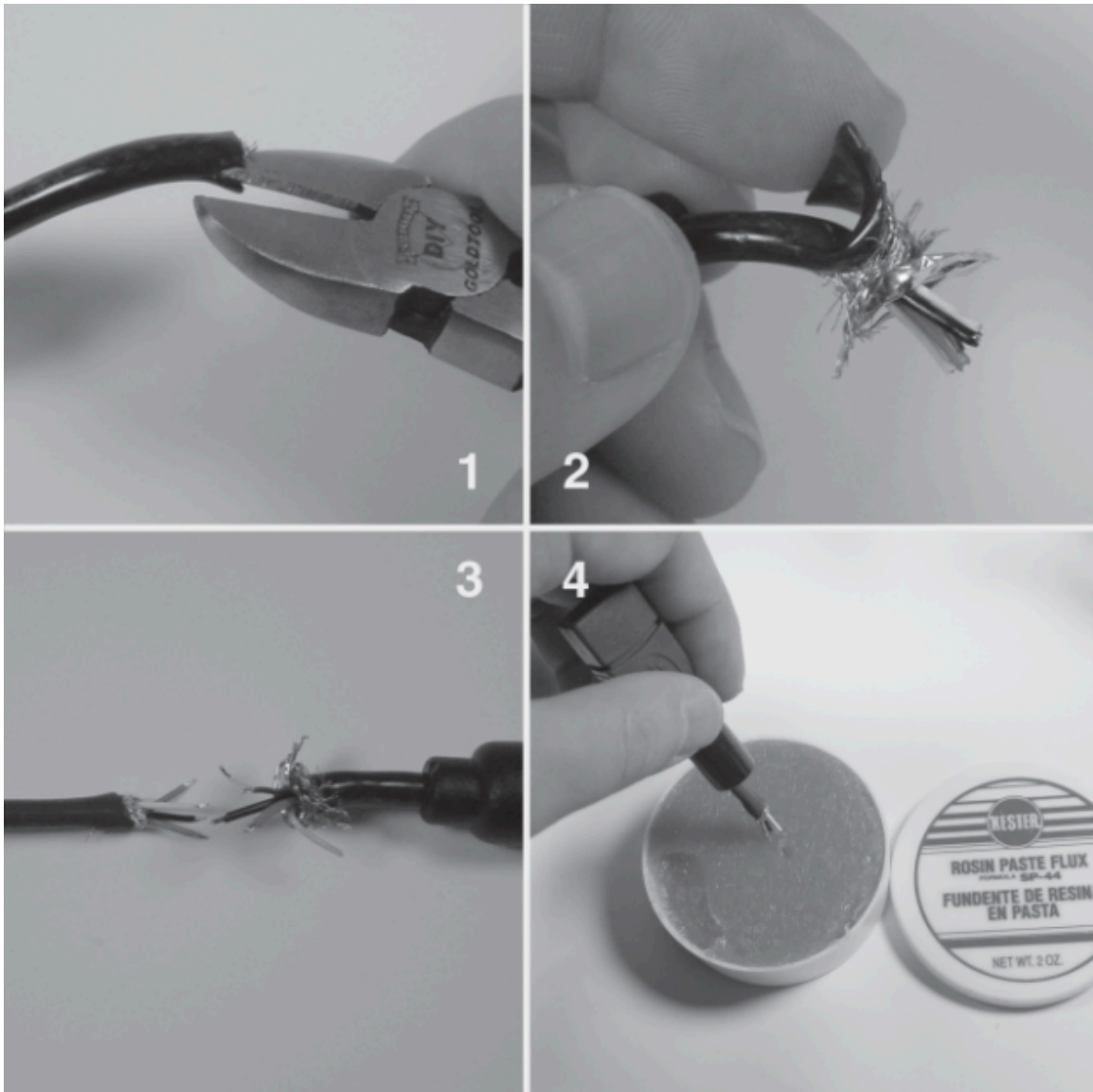


図 4-3: (1) ケーブル被覆に切り込みを入れ、(2) 被覆をめくって内部のワイヤーとシールドを露出させる。(3) 余分なシールドと被覆を切り取り、ワイヤー先端を約3mm剥く。(Xbox コネクタケーブル右側の黄色いワイヤーは剥かないことに注意。)(4) 最後に、剥いたワイヤーの先端をはんだフラックスに浸す。

次に、図 4-4 に示すように同じ色のワイヤー同士をはんだ付けする。はんだ付けの際は友人にケーブルを押さえてもらうか、ワニ口クリップ付きの「第三の手」ツール (Jameco 注文番号 26690) でケーブルを固定するといい。はんだ接合部はすべて滑らかで光沢があるよう仕上げる。各接合部を強く引っ張って、はんだ接続がしっかりしているか確認する。露出した接合部が互いに短絡しないよう、小さな絶縁テープを巻いておく。次のステップでは接合部を永久封入するので、その前にケーブルをテストしておくこと。

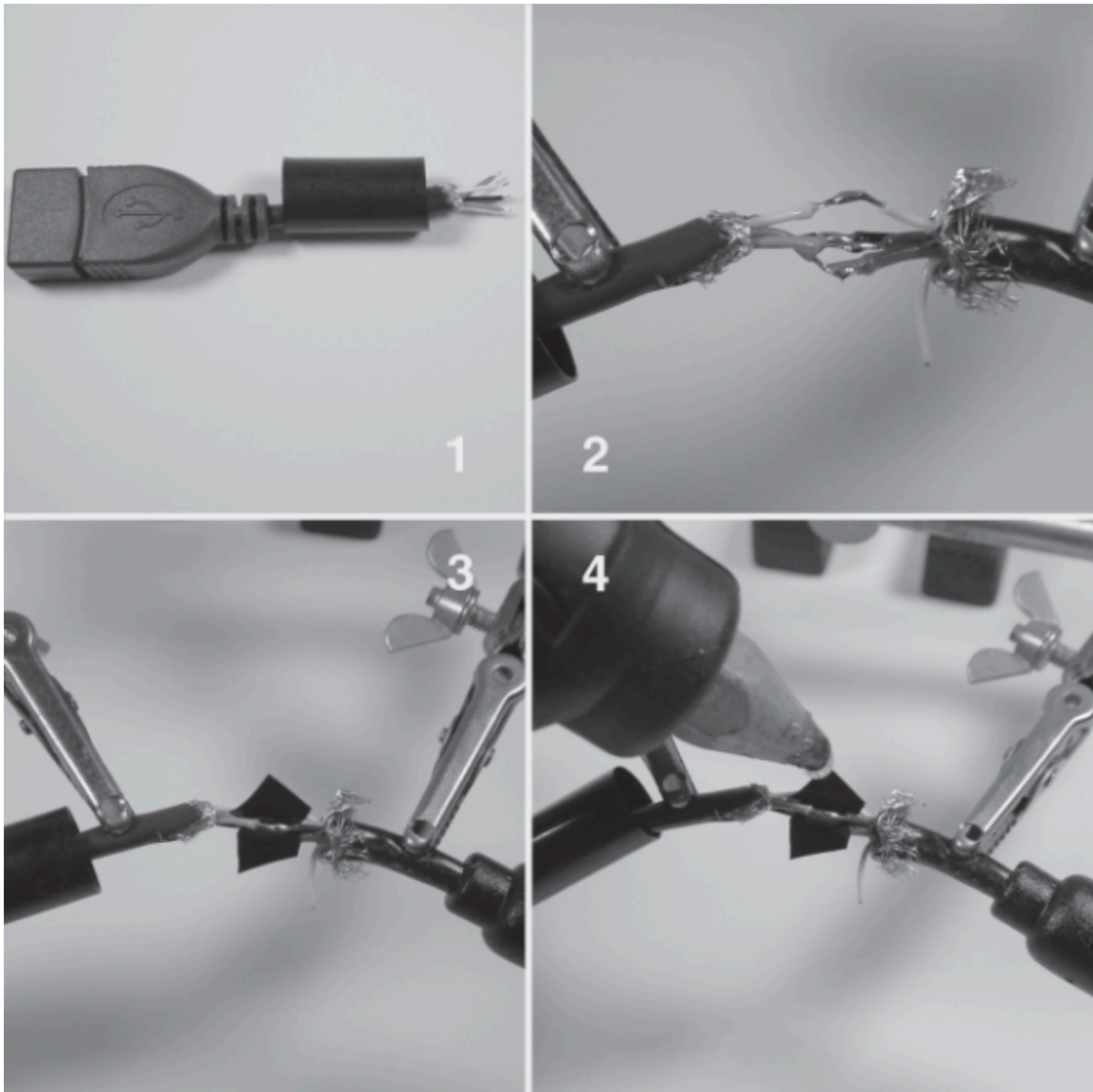


図 4-4: (1) はんだ付け前に熱収縮チューブをケーブルに通しておく。(2) 同じ色のワイヤー同士をはんだ付けする。(3) 短絡防止のため、ワイヤーを絶縁テープで巻く。ケーブルをテストする。(4) テスト済みのケーブルに少量のホットグルーを点付けし、次のステップに備えて絶縁テープとワイヤーを固定する。

ケーブルのテストが完了して問題ないことを確認したら、図 4-5 に示すように、機械的補強のためにはんだ接合部に丈夫なケーシングを施す。露出したはんだ接合部にホットグルーを少量点付けして、互いに短絡しないよう固定する。次に熱収縮チューブをはんだ接合部の上に被せ、チューブの両端からホットグルーを充填する。グルーの熱でチューブが収縮し、接合部を覆う固くて恒久的な外装が形成される。(ホットグルーは接合部のひずみ緩和としても機能するため、通常の使用状況では大半の負荷に耐えられる丈夫なケーブルになる。)

ホットグルーと熱収縮チューブがなくてもケーブルは完全に機能する。これらが手元がない場合は、各接合部に絶縁テープを丁寧に巻くことで応急的な機械的補強として代用できる。

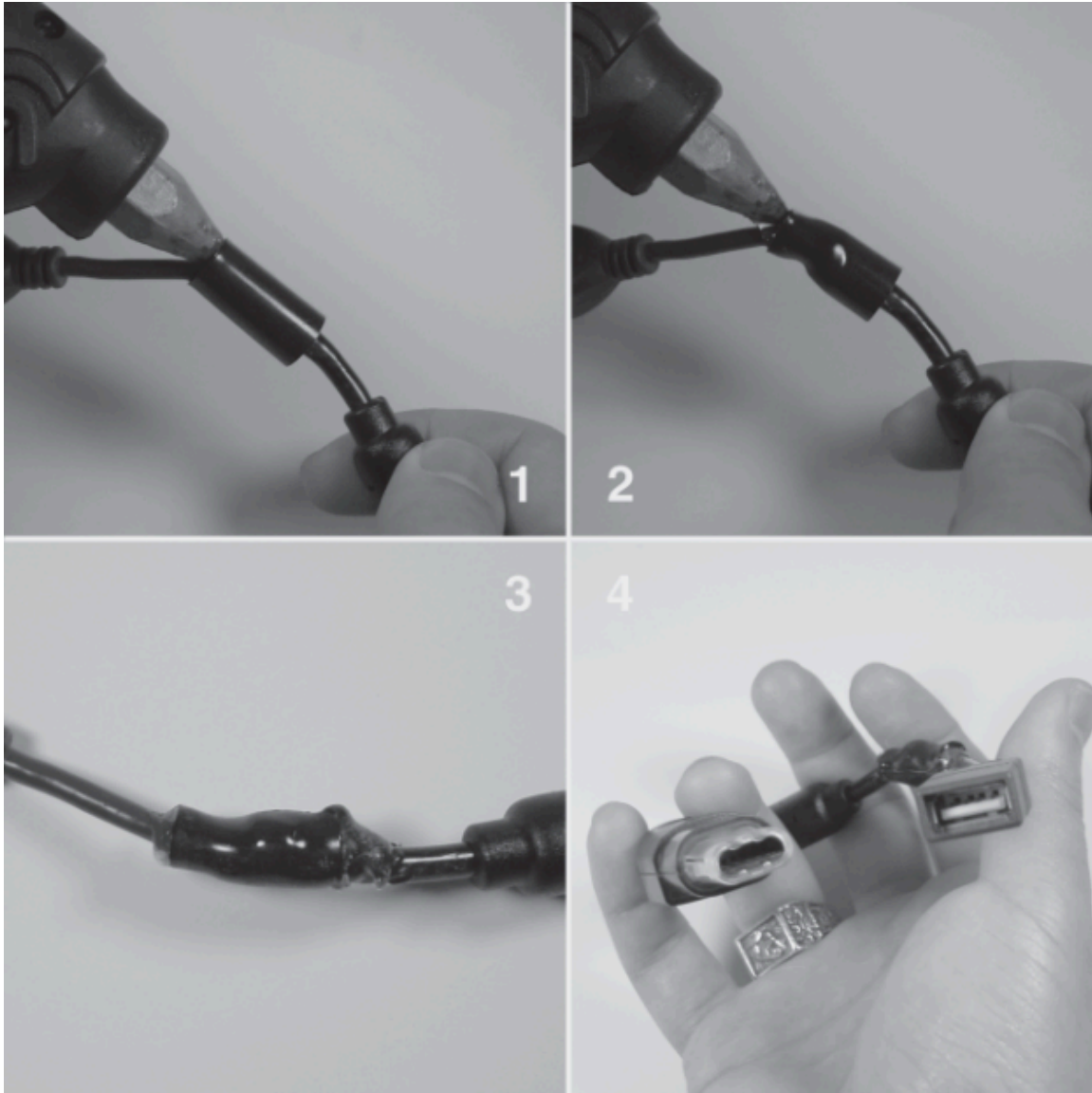


図 4-5: (1) 熱収縮チューブを接合部に被せ、ホットグルーを充填し始める。(2) ホットグルーの熱で熱収縮チューブが収縮する。(3) 完成品は大抵の扱いに耐える恒久的な外装で覆われる。(4) 完成した USB アダプタケーブル全体の写真。Xbox 側と USB 側のコネクタを示す。

Xbox ゲームポートへの USB アダプタが完成したら、第 11 章と第 12 章で Xbox に Linux をインストールするための手順を説明している。新しいアダプタを活用してほしい。

第5章 壊れた電源を交換する

残念ながら（そして驚くほどよくあることだが）、Xbox が3か月の保証期間を過ぎてから壊れた場合、修理の公式な手段はMicrosoftに修理代を払うことだけだ。最も簡単な修理でも100ドル以上、つまりコンソールの元の購入価格の約半額かかることもある。そのため、壊れた電源やハードドライブの直し方を尋ねるメールが大量に届いている。

残念ながら、壊れたハードドライブの交換にはXboxのセキュリティシステムを突破する必要がある。Xboxのマザーボードは特定のハードドライブに紐付けられた固有の鍵を使っているからだ。新しいハードドライブを取り付けるには、ハードドライブロックを書き換えるかバイパスできるモッドチップが必要になる。さらに、工場出荷時にインストールされたXboxのソフトウェアのコピーも必要だが、修理目的であっても配布・コピーすることは違法だ。そのため、Xboxのハードドライブ交換というトピックはデリケートすぎて本書では扱えない。ハードドライブ交換に関する数多くのFAQをオンラインで探してほしい。

一方、XboxのATX電源（Advanced Technology eXtended）は標準的なPCの電源と非常によく似ている。Xboxの交換用電源はいくつかのウェブサイトで購入できる——たとえばLlama.com (www.llama.com/xbox/Repairs/repairs.htm)、XboxRepair.com (www.xboxrepair.com)、Firefly-HK (www.firefly-hk.com) などだ。あるいは標準的なPCのATX電源から自分で作ることも可能だ。電源の故障は頻繁に起きるので、標準的なPC用ATX電源をXbox向けに改造する方法を紹介しよう。この手法にははんだ付けが不要だが、Xboxの電源を入れるときに余分なスイッチを操作する必要がある。付録C「PCBレイアウト入門」では、追加スイッチなしで済む簡単なプロジェクトを紹介している。

もちろん、Xboxの完全な交換用電源を購入して本章の手順の一部を参考にインストールする方が楽だ。しかし直接交換するより改造する方が学習効果は高い。ATX電源をXbox向けに改造することで、クリンプケーブルの作り方と電子理論の基礎、そしてXboxの仕組みが少し理解できる。

標準的なPCのATX電源をXboxに転用するもう一つの理由は、コンソールに余分な電力を供給するためだ。OEM (original equipment manufacturer、純正) のXbox電源はXboxの需要をкаろうじて満たす程度の電力しか出力しない。無改造のXboxに追加のドライブやファンを接続すると純正電源に過負荷がかかり、焼き切れる可能性がある。

なお、執筆時点で、Xboxの新しいハードウェアリビジョン（Xbox ハッキングコミュニティでは「v1.2」と呼ばれる）がリリースされており、本章で後述する独自のXbox電源コネクタの代わりに標準ATX電源コネクタが搭載されているようだ。改造作業に進む前に、自分のXboxの電源コネクタが本章で説明するコネクタと一致するか確認すること。本章が前提とするXbox電源コネクタは1列に12ピンが並ぶ形式だが、新しいコネクタは2列10ピンで合計20ピンある。また、最新のXboxハードウェアリビジョンはATX互換のコネクタを持つが、電氣的に標準ATX電源と互換があるとは限らない。新しいATX互換コネクタを持つXboxに標準ATX電源を接続する前に、電源コネクタの電圧を実測

して ATX 仕様書 (www.formfactors.org/developer/specs/atx/atx2_1.pdf) と比較することを強く勧める。

[!WARNING] 電源の交換作業では危険な電圧にさらされる可能性がある。電源を取り外す前には必ず Xbox をコンセントから抜き、蓄積された電荷が逃げるまで1分ほど待つこと。また、交換電源の取り付けが不適切だと、コンソールに恒久的な——場合によっては爆発的な——損傷を与える可能性がある。電源の放電が完了して電源がオフになっていることを確認した上で、コンソールをさらに損傷させるリスクを受け入れられる場合にのみ作業を行うこと。

壊れた電源の診断

Xbox がうまく On にならない場合は、まず問題を診断して故障箇所を特定しなければならない。故障がコンソール本体やコンセント側にある場合、電源を交換しても意味がない。次の診断手順に従って、Xbox の電源が本当に問題の原因かどうかを確かめよう。

1. 100W 以上の電球を使ってコンセントが機能しているか確認する。Xbox の負荷を正確に再現するために最低 100W が必要だ。
2. 電源コードにねじれや切れ目がないか目視で確認する。
3. 電源コードのプラグが Xbox の電源端子にしっかり挿さっているか確認する。これらのチェックを行っても Xbox が On にならない場合は次に進む。
4. Xbox 内部に焦げ跡や膨らんだコンデンサがないか目視で確認する。マザーボードに焦げ跡があればマザーボード自体を交換（つまり新しい Xbox を購入）する必要があるかもしれない。電源に焦げ跡があれば電源が損傷しており、交換作業に進んでよい。（電源故障によってマザーボードも損傷している可能性があるため、電源を交換しても Xbox が動作しないケースがある点は頭に入れておこう。）
5. メイン電源コネクタとフロントパネル基板コネクタがしっかり挿さっているか確認する（フロントパネル基板コネクタの位置は第3章「青色 LED を取り付ける」の図3-3を参照）。Xbox の電源スイッチはフロントパネル基板コネクタを通じてマザーボードに接続されている。
6. Xbox を Off のままコンセントに挿した状態で、ボルトメーターを使って 3.3V スタンバイ電圧 (3.3VSB) が仕様範囲内かどうか確認する。3.3VSB の測定は、電源コネクタのフロントパネル側の端から 6 本目のワイヤーとコネクタ上の黒いワイヤーのどれかにプローブを当てる。電源コネクタのワイヤーとコネクタの間の隙間にプローブを差し込むことで電圧を測定できる（マザーボード電源コネクタの内部には電源ワイヤーを囲む金属カラーがある）。3.3VSB の値が 3.14~3.47 ボルトの範囲外であれば電源の交換が必要だ。
7. Xbox の電源ボタンを押して On にしてみる（電源が壊れていればほとんど何も起きないはずだ）。電源から音がしたり煙が出たりした場合はすぐにコンセントを抜き、電源交換作業に進む。Xbox が完全に死んでいるようなら、電源から出ている各主要電圧を測定する。黄色いワイヤーは 11.4~12.6 ボルト、赤いワイヤーは 4.8~5.25 ボルト、オレンジのワイヤーは 3.14~3.47 ボルトの範囲にあるべきだ（これらの電圧はすべて黒

いワイヤーを基準とする)。また、Power OK シグナル (ピン 12、フロントパネルから最も遠いピン) の電圧が 3.1 ボルトを超えているかも確認すること。

リスト上のすべての項目が問題なければ、電源に故障はない可能性が高い。さらに調べるとすれば、電源スイッチの電氣的・機械的な問題 (スイッチのある基板の取り外し方は第 3 章を参照) やマザーボードの動作状況だ。ただし、電源故障の兆候が見られた場合は次を読み続けてほしい。

電源の交換

Xbox の ATX 電源を交換する全体的な方針は、標準的な PC 用 ATX 電源を Xbox 向けに改造することだ。必要な機材はこちらだ：

- 標準 ATX 電源。1U 電源は Xbox ケースの設置面積に収まるが、ケースを閉めるには少し高さがありすぎるかもしれない。
- (オプション) ATX マザーボード電源延長ケーブル。PC Power and Cooling (www.pcpowerandcooling.com) など多くのベンダーから購入できる。ATX 電源本体のケーブルの代わりに延長ケーブルを改造すれば、Xbox を手放す際に ATX 電源を標準 PC で再利用できる。
- クリンプツール。Molex 汎用クリンプツール (Digi-Key 部品番号 WM9999-ND) が強く推奨されるが、やや高価 (約 35 ドル) だ。Jameco 159265 のような安価なクリンプツールは 3 分の 1 ほどの価格で購入できるが、使い勝手が悪く、必要な接続強度を得るためにクリンプ端子にはんだを使う必要があるかもしれない。
- 12 ポジション・ピッチ 0.156 インチのコネクタハウジング 1 個 (Digi-Key 部品番号 WM2313-ND) または 6 ポジション・ピッチ 0.156 インチのスタッキングコネクタハウジング 2 個 (Jameco 部品番号 104731)。Xbox の ATX 電源コネクタ交換品として使用する。
- ピッチ 0.156 インチの電源コネクタ用クリンプ端子 13 個 (Digi-Key 部品番号 WM2313-ND、または Jameco 部品番号 78318)。
- DO-41 パッケージの 1N4001 またはそれ以上のシリコン整流ダイオード 2 個 (Digi-Key 部品番号 1N4001DICT-ND、または Jameco 部品番号 35975)。
- ワイヤーストリッパー。18 ゲージ線に対応できるものなら何でもよい。
- ワイヤカッター。斜めニッパーなら何でもよい。
- 絶縁テープ。

コラム：ダイオードで電圧を下げる

Xbox は +3.3V スタンバイ電源電圧を必要とするが、ATX 電源が出力するスタンバイ電圧は +5V だ。本来の解決策は +5V を正確に +3.3V に変換するレギュレータを使うことだが、このハックの目標ははんだ付けを最小限にして電源を交換することだ。

代替の解決策は、ダイオードを 2 個使って +5V を「ほぼ問題ない」+3.6V まで下げることだ。これができるのは、順方向に電流を流したダイオードの電圧降下が電流に対して対数的に比例するからだ。言い換えると、ほとんどの電流値において、ダイオードの両端の電圧はほぼ一

定だ。シリコンダイオードの順方向電圧降下はほぼ一様に約 0.7V で、2 個を直列につなぐと 1.4V 下げられる。

このハックで使う 1N4001 ダイオードは 1 アンペアしか流せないため、大電流が必要な用途にこのテクニックを使ってはいけない。幸い、Xbox のスタンバイ ATX 電源が引き出す電流はわずかなので、ダイオードが焼き切れる心配はない。

最後に一点、ダイオードの電圧降下は流れる電流量によってわずかに変動するため、精密に安定した電圧が必要な用途にはこのテクニックを使わないこと。Xbox の場合は電圧が少し高め側で動作しているが、この電源から動くデジタルロジックはその条件に十分耐えられる。

方針

標準 ATX 電源のインターフェースは Xbox の電源と非常によく似ている。Xbox には +3.3V、+5V、+12V、+3.3V スタンバイ電源、そして「Power OK」と「Power On」という 2 本の制御信号が必要だ。Power OK シグナルは電源出力が安定して適切に安定化されていることを示し、Power On シグナルは Xbox から電源の ON/OFF を制御する。標準 ATX 電源には Xbox を動かすのに十分な +3.3V、+5V、+12V 出力があり、Xbox マザーボードと ATX 互換の Power OK シグナルも備えている。ただし ATX 電源は +3.3V スタンバイではなく +5V スタンバイ電圧を生成し、Power On シグナルは Xbox とは極性が逆だ。この 2 つの非互換性は、はんだ付けなしに対処できる。

2 個のダイオードを直列につなぐことで +5V スタンバイ電圧を +3.6V 弱に下げる。ATX 電源への Power On シグナルはデフォルトで「On」になっているため、接続しないままにしておけば電源は常に On 状態を維持する。つまりコンソールがオフのときも電源は入り続ける。コンソールの電子回路に問題はないが、見た目には少し気になるかもしれない。付録 C「PCB レイアウト入門」では、これらの非互換性をより優雅に解決できるサンプル設計を紹介している。ただし付録の設計ははんだ付けと基板設計が必要になる。

手順

Xbox の ATX 電源を交換する手順は 2 段階だ：

1. 標準 ATX 電源ケーブルを Xbox 電源ケーブルに改造する
2. 古い電源を取り外して新しい電源を取り付ける

Xbox 電源ケーブルの製作

まず図 5-1 のように、ATX 電源ケーブルの既存のマザーボードコネクタを切り取る。ATX マザーボード電源延長ケーブルを使って改造する場合、将来 ATX 電源を標準 PC で再利用できるよう ATX 電源コネクタを保持できる。手順はどちらも同じだが、本章の写真では ATX マザーボード延長ケーブルを使用している。



図 5-1: ATX 電源ケーブルのコネクタを切り取る。

次に、図 5-2 に示すように、ATX ケーブルの以下 10 本のワイヤーにクリンプ端子を取り付ける：

- 黄 1 本
- 赤 3 本
- オレンジ 1 本
- 黒 4 本
- 灰 1 本

安価なクリンプツールを使っている場合は、十分な強度のクリンプ接続を作るのに苦労するかもしれない。その場合は、クリンプ端子をワイヤーにはんだ付けして接続を仕上げる。はんだ付けの際は十分に熱を加えないとはんだがワイヤーとクリンプ端子に十

分浸透しない。はんだを当てる前後それぞれ約5秒間、はんだごてを接合部に当てておくこと。

紫のワイヤー（+5V スタンバイワイヤー）には、ワイヤーの端とクリンプ端子の間にダイオードを2個直列につなぐ。図5-3の手順ではクリンプ端子の一部をダイオードの接続に使うため、はんだ付けは不要だ。（ダイオードは極性があることに注意：逆向きに取り付けると電流が流れない。ダイオードはカソード側——帯が描かれている端——がマザーボード向きになるよう取り付けること。）

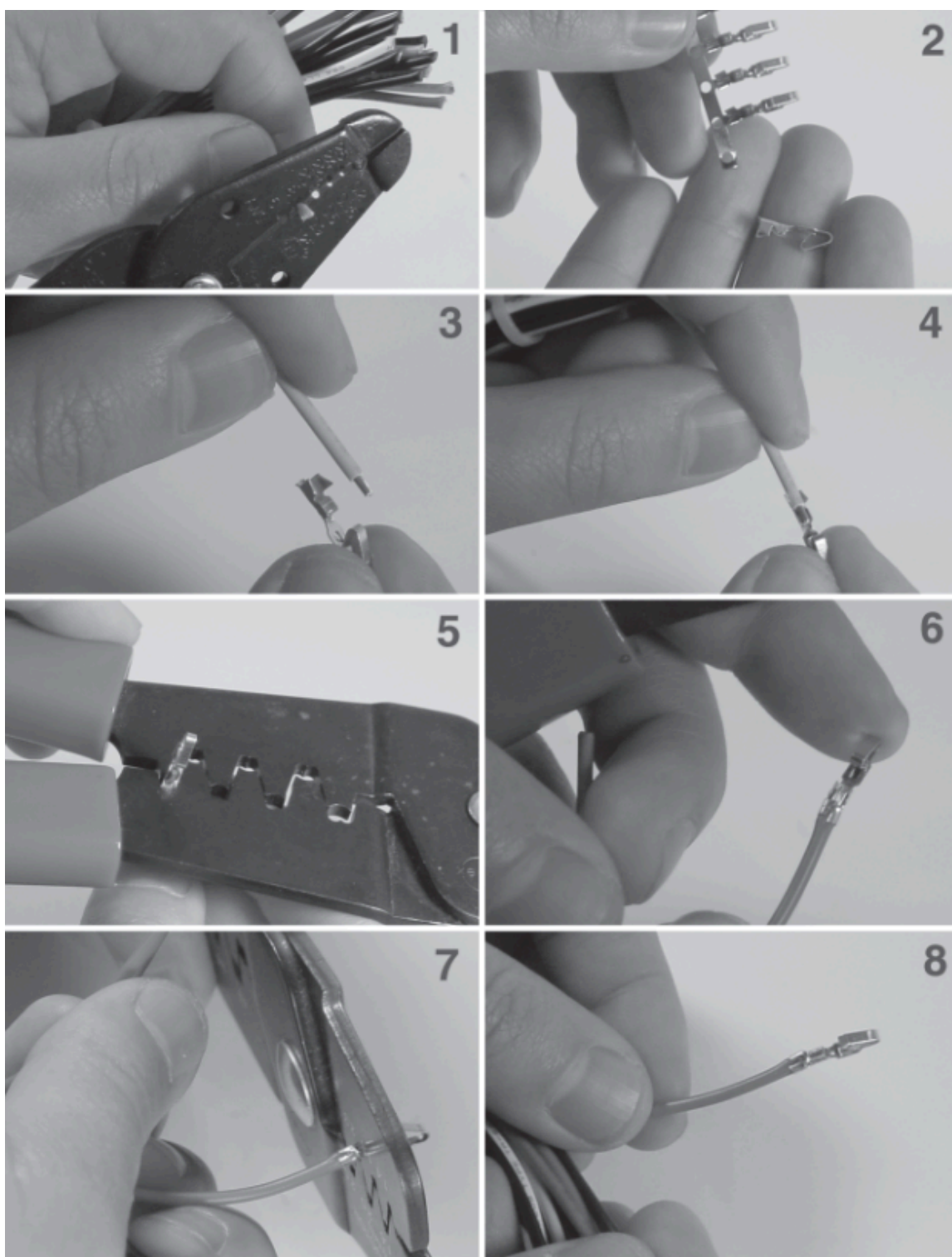


図5-2: ワイヤー端へのクリンプ端子取り付け手順。(1) ワイヤー端の被覆を約3mm (1/8インチ) 剥く。(2) 必要であればクリンプ端子を保持ストリップから外す。(3・4) 被覆

部分が長い方のクリンプフィンガーの間に約 1.5mm (1/16 インチ) 入るようワイヤーをクリンプ端子に差し込む。(5) クリンプフィンガーの被覆部分 (長い方のペア) をかしめる。(6) 被覆部分だけのかしめた状態。(7) クリンプフィンガーの導体部分のかしめる。(8) 完成したクリンプ端子。導体部のクリンプフィンガーは裸線にしっかりと折り畳まれていること。端子端を強く引っ張って接続強度を確認する。

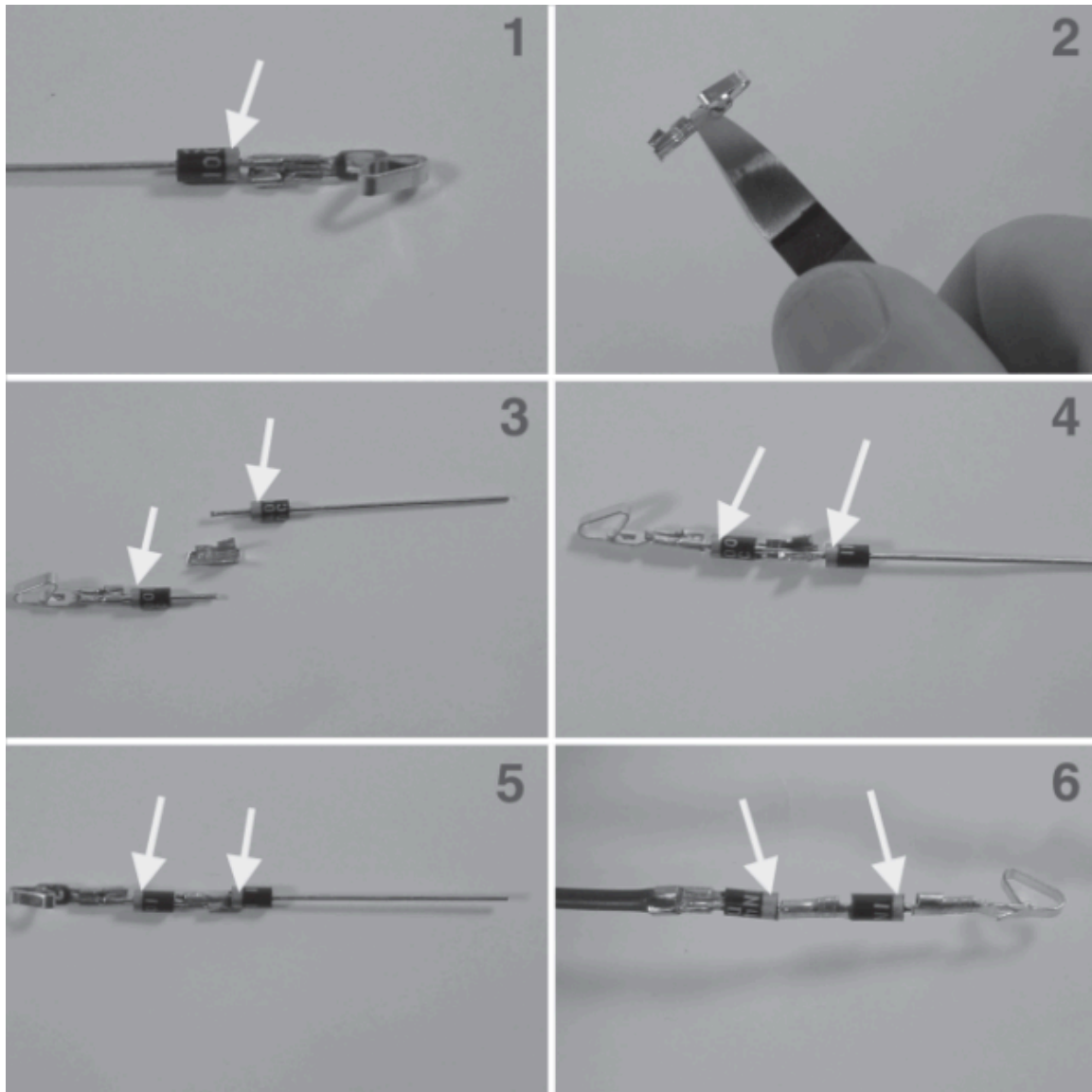


図 5-3: 紫ワイヤーの端とコネクタに差すクリンプ端子の間にダイオードを 2 個直列につなぐ。矢印はダイオード端部の極性ラインの正しい向きを示す。この作業でクリンプ端子を合計 3 個使用する。(1) クリンプ端子に極性帯がクリンプ端子側になるよう 1 個目のダイオードを取り付ける。(2) クリンプ端子を半分に切ってリーフ接触部を除去する。(3・4) 切断したクリンプ端子のクリンプ部にダイオードの極性に注意して配置する。(5) か

しめ後のダイオード。(6) 同様の手順で 2 個目の切断クリンプ端子を使い、紫ワイヤーの端にダイオードを接続する。

電源ケーブルの組み立てが完了したら、必ず作業内容を二重確認すること。ミスがあればコンソールに恒久的で修復不可能なダメージを与える可能性がある。図 5-6 は完成したコネクタアセンブリがどのようなものかを示している。利用可能であればケーブルタイを使って未使用ワイヤーをまとめておくと、邪魔にならず短絡や損傷を防げる。



図 5-4: 未使用のワイヤーとダイオードを絶縁テープで巻いて偶発的な短絡を防ぐ。

ピン	色
1	黄
2	赤
3	赤
4	赤
5	オレンジ

6	ダイオード+紫
7	黒
8	黒
9	黒
10	黒
11	空き
12	灰

表 5-1: ATX 電源ケーブルを Xbox 電源コネクタに接続するための配線表（ワイヤー挿入側から見て、極性リブが左側）。

最後に、仕上がったクリンプ端子を 0.156 インチコネクタハウジングに差し込む。完全かつ正しく挿入されると、クリンプ端子はハウジング内部でロックされる（図 5-5 参照）。ワイヤーは表 5-1 に示す順番で差し込むこと。

12 ポジションのコネクタハウジングを取り扱っていないベンダーもある。その場合は 6 ポジションのコネクタハウジングを 2 個使い、重ねる際の順序に細心の注意を払うこと。また、コネクタの極性リブに対するピン 1 の位置を確認すること。ピン 1 はワイヤー挿入側（上面図）から見て極性リブが左にあるときコネクタの一番上に位置する。コネクタを逆にして向きを間違えやすいので、既存の Xbox コネクタを参考にする。黄・赤・オレンジ・黒のワイヤーが両コネクタで並んで一致するはずだ。

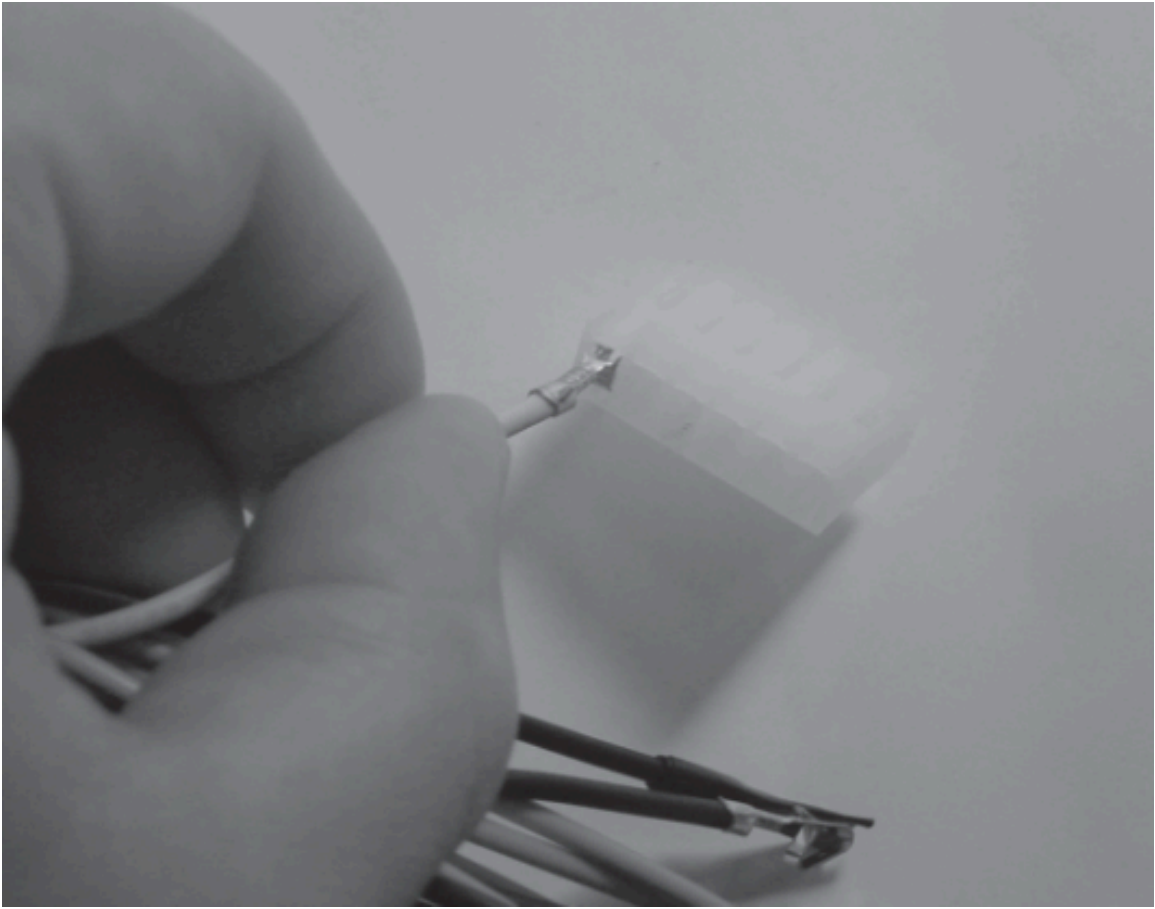


図 5-5: クリンプコネクタをコネクタヘッドに差し込む。

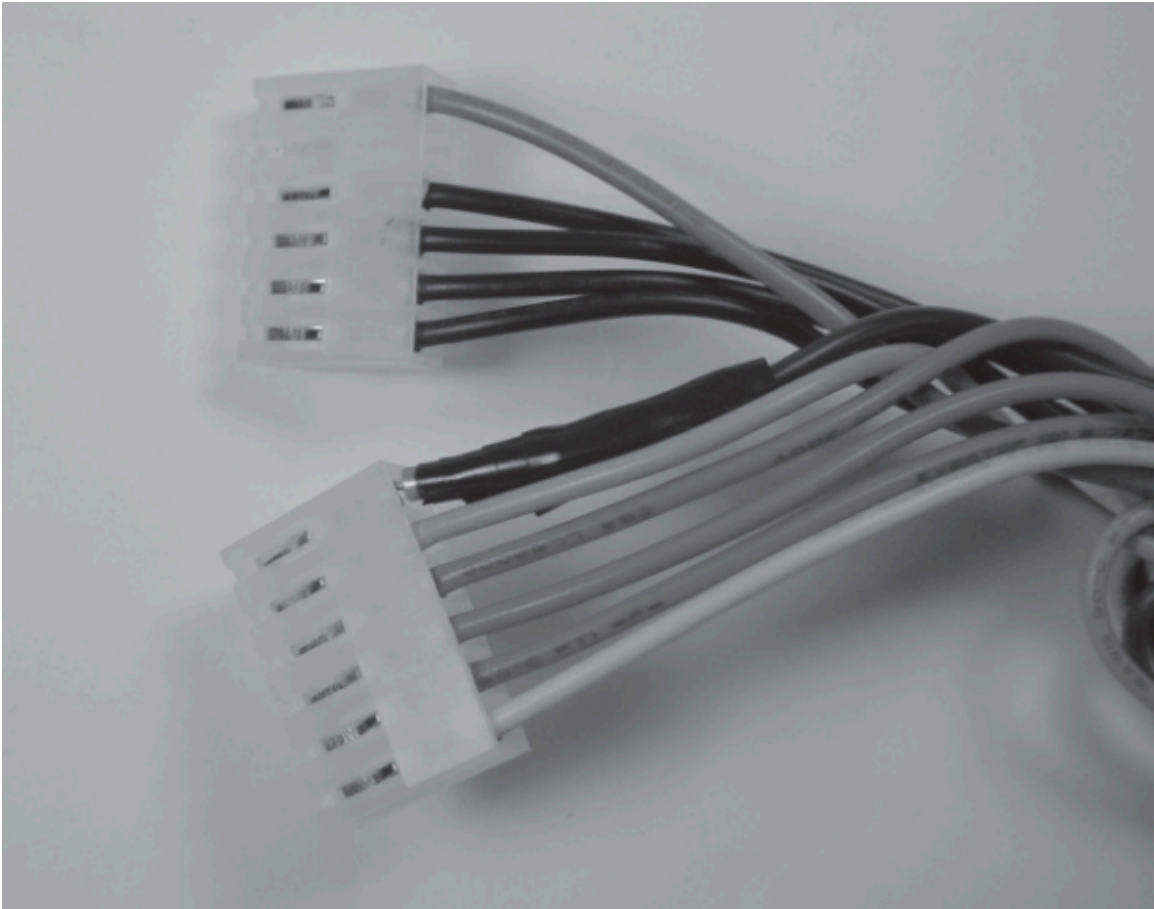


図 5-6: 完成したケーブルアセンブリ。

交換電源の取り付け

交換用電源の準備ができたなら、古い壊れた電源と入れ替える。まず第1章「保証を無効にする」の手順に従って Xbox ケースの上蓋を外し、ハードドライブの電源コネクタを抜いてハードドライブをケースから取り出す。ハードドライブに接続されているグレーの IDE リボンケーブルは外す必要はない。

[!NOTE] この時点で、Xbox のプラグをコンセントから必ず抜いてあること、そして電源に蓄積された電荷が逃げるまで少なくとも 1 分間置いてあることを確認すること。コンセントに挿したままや、コンセントを抜いてすぐに作業すると非常に危険だ。電荷が逃げる前に電源の金属部分に素手で触れると、重傷または死に至る電撃を受ける可能性がある。

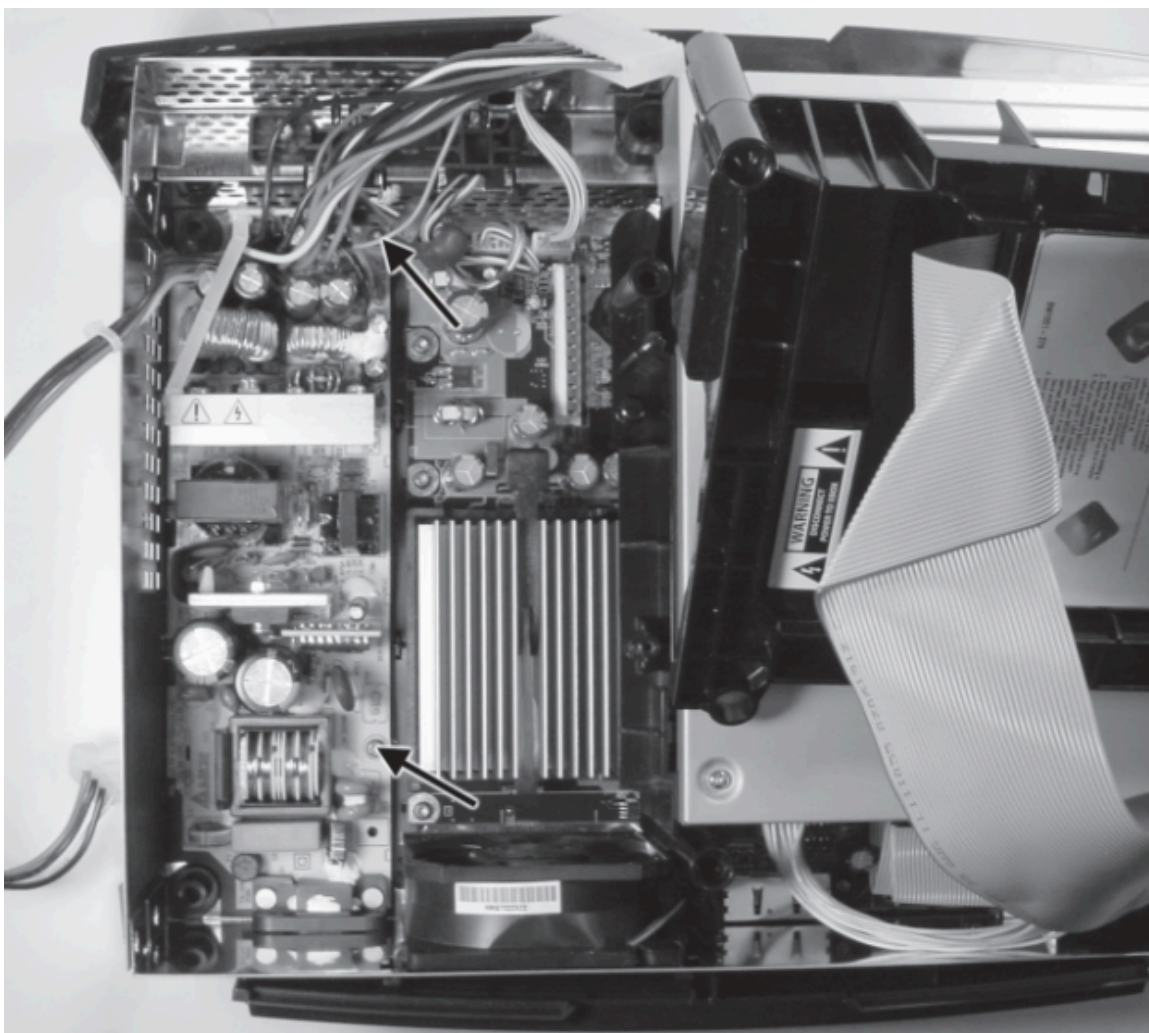


図 5-7: 電源を固定する 2 本のネジの位置。

1. 電源ワイヤーの束をまとめて掴み、もう片方の手で Xbox を押さえながらケーブルを力強く引いて Xbox の電源コネクタを抜く。取り外しの際はケースやヒートシンクの金属の鋭いエッジに注意すること。
2. Xbox の電源を固定している 2 本の T-10 トルクスネジを外す (図 5-7 参照)。
3. Xbox のフロント側に近い端を先に持ち上げて、電源を Xbox ケースから取り出す。
4. 作成したケーブルと既存の Xbox 電源ケーブルを並べて、極性リブを合わせながら比較確認する。赤・黄・オレンジ・黒のワイヤーが 2 つのコネクタで一致するはずだ (この確認でケーブルの配線ミスによる Xbox へのダメージを防げる)。
5. ATX 電源ケーブルアセンブリを Xbox の電源コネクタに接続する。黄のワイヤーが Xbox のフロント側に最も近いピンに来るようにする。電源ケーブルを 1~2 ピンずれた位置に挿してしまう可能性があることを忘れずに。電源コネクタに空きピンが見えたら 1 ピンずれて挿さっている。この状態は Xbox のハードウェアや電源に恒久的なダメージを与える可能性があるので必ず確認すること。

6. Xbox のハードドライブを ATX 電源のディスクドライブ電源コネクタの 1 つに接続する。ハードドライブのコネクタは標準 PC のディスクドライブ電源コネクタと同じなので改造は不要だ。
7. ワイヤーがケースに短絡していないか、冷却ファンの羽根に引っかかっていないかを確認する。これで Xbox の電源を入れる準備が整った。

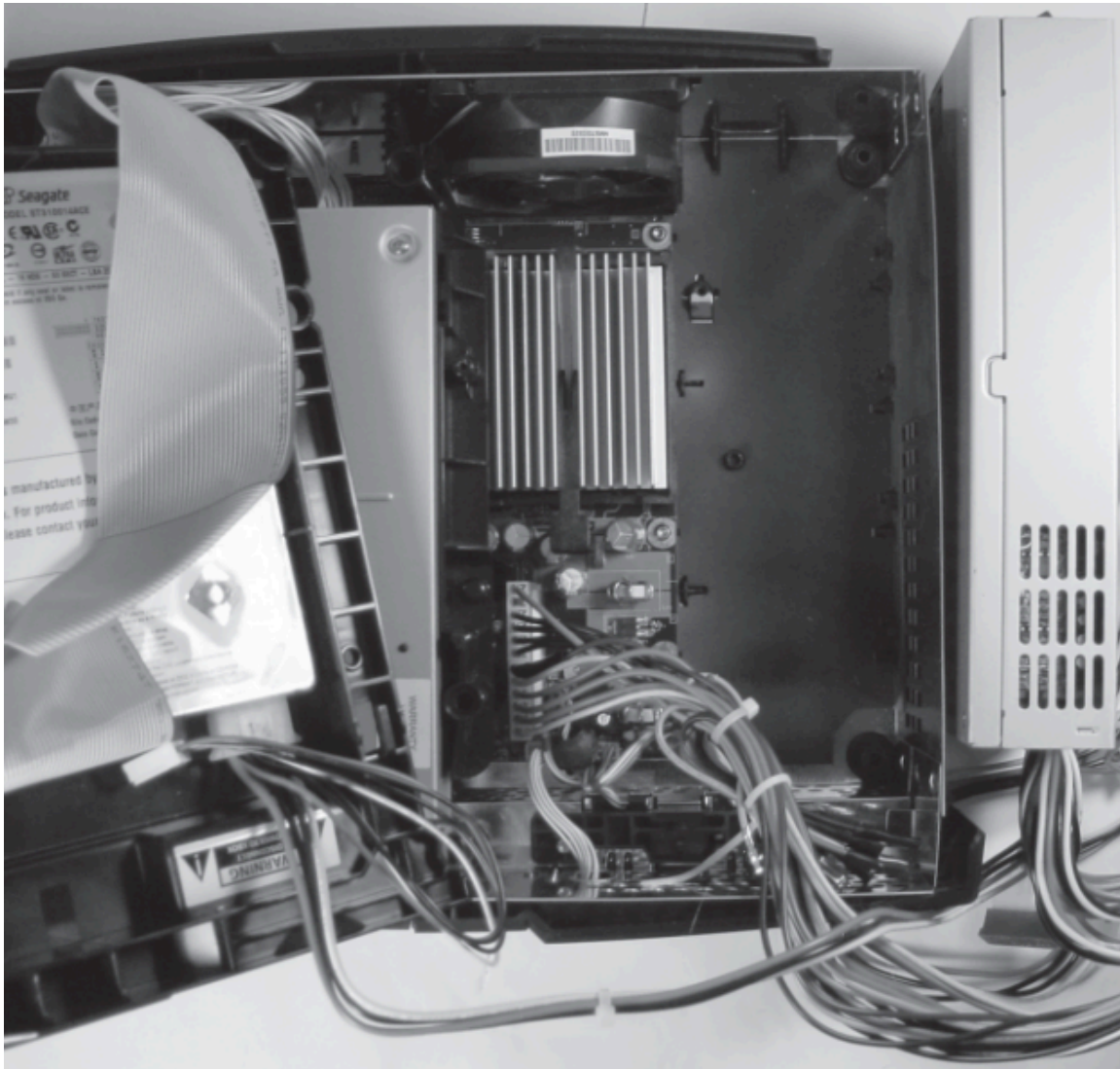


図 5-8: 改造した ATX 電源延長ケーブルを使って Xbox に接続した 1U スリム型 ATX 電源。

交換電源での起動

ほとんどの ATX 電源には電源スイッチが付いている。まずスイッチをオフにし、ATX 電源のプラグをコンセントに挿し、スイッチをオンにする。この時点で ATX 電源は Xbox に電力を供給し始める——Xbox のシステムコントローラーはボックスが Off だと認識して

いても、だ。その結果、Xbox 内部の冷却ファンが回り始めるはずだ。次に Xbox フロントの電源スイッチを押す。これで Xbox が正常に起動するはずだ。うまくいったら、おめでとう！

[!WARNING] Xbox のバージョンによっては GPU にヒートシンクファンが付いていないものがある。マザーボード上のチップに直接ファンが取り付けられていなければ、そのタイプの Xbox だ。GPU のヒートシンクファンがない Xbox は特定の条件下で過熱しやすく、長期安定動作のためにはドライブをマザーボードの上に設置した状態で動かす必要がある。ディスクドライブの裏面がエアダクトシステムを形成し、Xbox のメインケースファンから GPU ヒートシンクに向けて風を通していている。(ATX 電源ケーブルがあるとディスクドライブをケースにぴったり収められなくなるが、エアフローのダクト機能への影響は大きな問題ではない。)

Xbox の電源を切りたい場合は、ATX 電源のスイッチを切るだけでいい。あるいは Xbox フロントの電源スイッチで先に Xbox をオフにしてから、ATX 電源をオフにしてもよい。

デバッグのヒント

電源を交換後に Xbox が正常に起動しない場合は、本章冒頭の「壊れた電源の診断」のチェックリストを使ってケーブルをテストしよう。最もよくある問題は、クリンプ接続の不良やダイオードの取り付けミスだ。クリンプ接続の不良は、Xbox は On になるが頻繁にクラッシュするという断続的な動作不良につながることもある。

何らかの理由で Xbox が起動シーケンスの途中で止まる場合は、第 3 章「青色 LED を取り付ける」の「デバッグ」セクションにある表 3-1 を参照してほしい。可能性のある問題とその原因の一覧が載っている。付録 E「デバッグ：ヒントとコツ」には、デバッグの技法と方法論についてより詳しい解説がある。

Xbox は正常に動作するが時々クラッシュする場合は、GPU のヒートシンクファンがない Xbox かもしれない。問題の詳細は前のページの注意を参照してほしい。この問題を解決するには、追加のファンを取り付けるか、紙とテープで既存のダクトシステムを改善する必要があるかもしれない。

第 6 章 最高の Xbox ゲーム：セキュリティハッキング

Xbox のハードウェアを改造・調整する次の段階は、Xbox ハードウェアを完全に支配することだ。残念ながら、ハードウェアを支配するのは思ったほど簡単ではない。Xbox の設計者たちは、高度なソフトウェア攻撃はもちろん、大半のシンプルなハードウェア攻撃からもコンソールを守るために、セキュリティに多大な労力を注いだ。Xbox のセキュリティ機構は、デジタル著作権管理 (DRM) アーキテクチャの産物だ。

[!NOTE] 原則として、自作プログラムを実行するといった「フェアユース」の目的でハードウェアを使うことは、違法ではないはずだ。しかし、フェアユース、保護されたハードウェア、そして比較的新しい著作権保護回避法との関係はいまだ不明確だ。第 12 章「注意せよハッカーよ」では、ハッキングの法的問題をより詳しく取り上げている。

Xbox のセキュリティ対策を回避する方法はいくつもある。本章と第 8 章「Xbox セキュリティのリバースエンジニアリング Xbox セキュリティ」では、Xbox のセキュリティシステムを解析した冒険譚を語る。成功だけでなく、遭遇した失敗についても書く——そこから学べることもあるからだ。第 9 章「裏口から忍び込む」では、他のハッカーたちが Xbox セキュリティを回避するために取ったアプローチを解説する。第 7 章「セキュリティ入門」では、第 8 章と第 9 章を理解するための背景知識を提供している。

妄想的な設計との最初の出会い

2000 年春に Xbox が発表されたとき、ハードウェア愛好家コミュニティは沸き立った。その興奮の理由はゲームとしての可能性だけではない——300 ドルという手頃な価格で、ネットワーク対応の x86 アーキテクチャ高性能 PC として使える可能性があったからだ。発売数か月後の値下げにより、Xbox の価格はその後 200 ドルを下回るまで下がった。Xbox のアーキテクチャが x86 PC に非常に近いということは、膨大な既存アプリケーションと技術ノウハウが、理論上はコンソールに簡単に移植できることを意味していた。

僕が初めて Xbox の中身を見たのは 2001 年 11 月下旬、当時の彼女（今は婚約者）がクリスマス前のプレゼントとしてくれたときだ。すぐに作業に取りかかった。Xbox ハードウェアを支配するためには、まずブート ROM を取り出して中身を解析するのが最初のステップだ。第 2 章の Xbox アーキテクチャの解説を思い出してほしい——Xbox のブート ROM には、Xbox の動作環境を確立するためのコードがすべて含まれている。

ROM を読み出す

Xbox で使われている ROM は、電氣的に消去・書き換えが可能なフラッシュ ROM (FLASH ROM) だ。フラッシュ ROM には複数のパッケージ形式があるが、Xbox が採用しているのはもっとも一般的なパッケージの一つ、TSOP (Thin Small Outline Package、薄型小アウトラインパッケージ) だ。Xbox マザーボードの上面、セクター U7 に搭載されており、回路記号は U7D1 だ。TSOP パッケージは非常に見分けやすい——長方形の形状で、ピンがパッケージの短辺にのみ並ぶ、数少ないチップパッケージの一つだからだ。他のほとんどのパッケージはピンを長辺や全辺に配置して接続性を最大化しているが、フラッシュ ROM はシリコン面積あたりの I/O 要件が比較的少ない。ベースの型番「29F080」をウェブ検索で確認すれば、これが確かに 8Mbit のフラッシュだとわかる。

フラッシュ ROM の内容を読み出す（ハッカー流に言えば「スナーフ」する）技術はいくつかある。はんだ付け不要の方法は、フラッシュ ROM に挟み込むテストクリップを購入し、Xbox 本体には電源を入れずに、テストクリップ経由で ROM に電源を供給・制御しながら中身を読み出すことだ。この目的に適したテストクリップは Emulation Technology から購入できる (www.emulationtechnology.com)。(テストクリップによるオーバーライド方式にはいくつかの問題がある。最大の問題は、テストクリップ経由で電源を受けていないフラッシュ ROM 周辺のチップを恒久的に損傷させてしまう可能性だ。ただし Xbox の場合は問題が生じないようで、この方式を試みたハッカーたちは成功を収めている。1 僕が最初にこの方法を選ばなかったのは、マザーボードを壊したくなかったのと、この方法に必要な 300 ドルのテストクリップを買う余裕がなかったためだ。)

もう一つの巧みな方法は、フラッシュ ROM 周辺のテストポイントにワイヤーをはんだ付けして、Xbox が ROM の内容を読み取る様子を盗み聞きすることだ。盗み聞きの方法は二種類ある。一つは ROM とインターフェースできるカスタム基板にワイヤーを接続すること、もう一つはロジックアナライザ（デジタル信号を多チャンネルでキャプチャする計測器）を使い、Xbox CPU が ROM にアクセスする際のデータをキャプチャすることだ。後者の方法でも成功を収めたハッカーがいたし、実際にこの手法の過程で Xbox のブートシーケンスにいくつかのバックドアが発見されている。2 僕がこの方法も選ばなかったのは、最初の Xbox を入手した時点でロジックアナライザを持っていなかったこと、そして大量のワイヤーをはんだ付けする作業が非常に面倒で難しく、ミスも起きやすいからだ。

僕がとったのはもっとオーソドックスな方法だ——フラッシュ ROM を外して、ROM リーダーに差し込む。さらにマザーボードにソケットを取り付けた。そうすれば今後の ROM の着脱やプログラミングが素早く確実にできる。

細かいピンの完全性を保ちながらフラッシュ ROM を取り外すのは、適切な道具があれば簡単で、そうでなければほぼ不可能だ。コツは、フラッシュ ROM のピン全体を同時に加熱することだ。均一に加熱できれば、フラッシュ ROM はマザーボードからすっと外れる。標準的な鉛筆型のはんだごてではピン全体を同時に加熱できない。この作業に適した道具は、図 6-1 に示すような「トング」または「ピンセット」型のはんだごてだ。このタイプのはんだごては二つのヒーターエレメントを持ち、チップの両側を同時に加熱できる。さらに、チップ全体の長さを一度に加熱できる幅広のパドル型チップを備えている必要がある。

this approach as I did not want to risk damaging the motherboard, and because I could not afford the \$300 test clip required for the job.)

Another ingenious approach is to solder wires to the test points around the FLASH ROM to eavesdrop on the Xbox as it reads the ROM's contents. Eavesdropping can be accomplished by either connecting the wires to a custom board that can interface with the ROM, or by using a logic analyzer to capture the data as it is accessed by the Xbox CPU. The latter approach was used with success as well, and in fact some back doors in the Xbox boot sequence were discovered as a consequence of

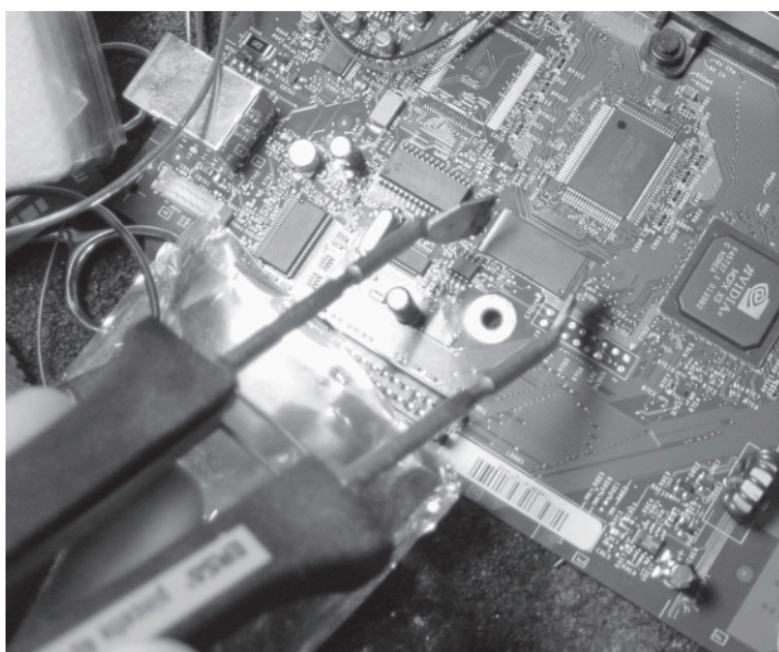


Figure 6-1: Removing the Xbox FLASH ROM with a tweezer-style soldering iron.

this methodology.² I chose not to use this approach either, as I did not have a logic analyzer when I got my first Xbox, and because soldering all the wires down can be very tedious, difficult, and error-prone. My approach was more traditional: just remove the FLASH ROM and drop it into a ROM reader. I also placed a socket on the motherboard, so that future removal and programming of the ROM would be very quick and reliable.

² Visor has written up his experiences with the logic analyzer snooping approach at <http://www.xboxhacker.net/visor/aXventure1.txt>

図 6-1: ピンセット型はんだごてで Xbox のフラッシュ ROM を取り外す。

こうした機能を持つはんだごては相当な値段（数百ドル）がするが、あらゆる場面で役立つ価値ある投資だ。僕は Ersat SMT Unit 60A を使っているが、展示会の会場で大きく値引きして購入したもので、学位取得中にこなした基板組み立て案件で元はすぐ取れた。Xytronic 製の手頃な価格のはんだごてが Jameco（品番 168410）から約 70 ドルで購入できるが、僕自身は使ったことがないので品質は保証できない。もう一つのリーズナブルで簡単な方法は、付録 B「はんだ付け技術」で説明している脱はんだ合金を使う方法だ。（ROM 用のソケット 3 は比較的安価で 20 ドル未満だが、取り付けには安定した手と光学ルーペ類が必要だ。）

ROM を外してピンを清掃・点検したら、ROM リーダーで内容を読み出せる。ROM リーダーは購入できるが、自分で作るのも良い学習体験だ。僕が自作した ROM プログラムについては、ウェブサイト (<http://www.xenatera.com/bunnie>) で紹介している。オリジナルの Flashburner4 プログラムはシンプルな設計で、第 2 バージョン 5 より理解・製作しやすいが、機能は劣る。

ただし、ROM をできるだけ早く読み出すことが目的なら、素直に ROM リーダーを購入する方がいい。良い ROM リーダーは、本気のハードウェアハッカーの工具箱に欠かせない一品だ。Needham's Electronics (<http://www.needhams.com>) は幅広い価格帯に対応した優れた ROM プログラム／リーダーを展開している。

Microsoft との邂逅

ROM の内容を取り出したら、次のステップは仲間のハッカーたちと共有して解析することだ——本当にそうだろうか？ ROM の内容をウェブサイトに掲載してから 12 時間も経たないうちに、Microsoft のエンジニアから Microsoft の著作権コンテンツをサイトから削除するよう丁寧に依頼する電話が来た。もちろんすぐにコンテンツをサイトから削除した。最初から公開しなければよかったのだ。

Microsoft とのこの最初の接触は、Xbox のリバースエンジニアリングは普通の家電製品のリバースエンジニアリングとは訳が違うという厳しい警告だった。リバースエンジニアリングのある側面を守る法律も存在するが、知的財産 (IP: Intellectual Property) 所有者を守る膨大な著作権法も存在する。Microsoft の権利を尊重しながら Xbox を共同でリバースエンジニアリングするのは、法的な地雷原を歩くようなものだ。

一方では、Microsoft は利益を期待して製品に投資しリスクを取る権利がある。しかし収益性は法律では保証されていない。たとえば Microsoft がやっているように、ソフトウェアの売上で差額を回収する目的でコンソールを大幅な赤字で販売する（「ロスリーダー」戦略）のはリスクの高い賭けであり、最終的に Microsoft が利益を得ることは法律によって保証されていない。一方でハッカーである僕たちには、自分が稼いだお金で購入したハードウェアをいじる権利（「フェアユース」）がある。Microsoft が実質的に PC を大幅割引で僕たちに売りたいというなら、それは別に構わない。Xbox におけるマイクロソフトの損失を補填するのに十分なゲーム（10 本前後以上）を購入するかどうかは、完全に Microsoft のビジネス・マーケティング戦略の問題だ。

僕の見方では、Microsoft の大きな損失対収益比はこの業界では少々異例だ。ソニーと任天堂はコンソールハードウェアのコストでほぼ損益分岐点を維持している。携帯電話会社も Xbox 並みの赤字で端末を販売することが多いが、端末コストを回収するためにユーザーとの契約を義務付けており、契約解除には違約金が発生する。これは Xbox Live のビジネスモデルへのマイクロソフトの自信の表れかもしれない。

こうした問題のちょうど中間に位置するのが、暗号技術を使った著作権保護機構とフェアユースの権利の相互作用だ。Xbox はコピープロテクションとコンソールの利用ポリシーを強制するために暗号技術を幅広く活用していることがわかった。そこで登場するのが、比較的新しく判例も少ない 1998 年デジタルミレニアム著作権法 (DMCA : Digital Millennium Copyright Act) だ。確立した判例が少なく、法律の文言の間にはグレーゾーンが多い。ハッカーとして自分が直面しうるリスクを自分で評価しなければならない。第 12 章「注意せよハッカーよ」では、新時代におけるハッキングの法的問題をより詳しく探っている。

ROM の中身を解析する

Microsoft に跳ね返されたが、ROM の中身は手元にある。そのまま解析を進めた。ブート ROM にはハードウェア初期化手順と、オペレーティングシステムをロードする命令が含まれており、ひょっとすると OS コード本体もあると予想されるだろう。しかし、どこから始めればいいのか？

ROM 中のプログラムは毛糸の玉のようなものだ。糸の端を見つけさえすれば、あとは時間と根気さえあれば必ず核心まで解きほぐせる。

幸い、Xbox の Pentium プロセッサの起動ポイントは Intel によって詳細に文書化されている。電源を入れると、プロセッサはリセットベクタと呼ばれる決まったアドレスからコードを実行し始める。このリセットベクタはアドレス 0xFFFF.FFF0——メモリの最上部付近だ。このアドレスのデータを 16 進数で見てみよう：

```
0xFFFF.FFF0  EBC6 8BFF 1800 D8FF FFFF 80C2 04B0 02EE
```

最初の 2 バイト「EBC6」は、0xFFFF.FFB8 番地へのジャンプ命令だ。最初のバイト「EB」は「次の命令からの相対変位による短い相対ジャンプ」という特定のオペコードで、2 バイト目の「C6」がジャンプの 8 ビット符号付きオフセットだ。つまりプロセッサが最初に行うことは、別の場所へジャンプすることだ——リセットベクタにはメモリの上端から落ちるまでに 16 バイトしか余裕がないため、どのブートプログラムも必ずこの処理を行う。このコードはリセットベクタとして典型的なもののなので、教育目的でここに転載しても問題ない。

次のコードブロックは、プロセッサの GDT (Global Descriptor Table、グローバルディスクリプタテーブル) と IDT (Interrupt Descriptor Table、割り込みディスクリプタテーブル) を初期化する処理だ。GDT と IDT はプロセッサのメモリ管理方式と割り込み処理方式を設定する。これらのレジスタの役割を正確に理解する必要はないが、興味があれば Intel のデベロッパーウェブサイト (<http://developer.intel.com>) で公開され

ている「IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide」で詳しく説明されている。

これらのレジスタの設定後、プロセッサはプロテクトモードに移行して 0xFFFF.FE00——メモリの最上部から正確に 512 バイト下の領域——にジャンプする。ここから話が面白くなってくる。セグメントレジスタを設定する短いコードのあと、ジャムテーブルインタープリタ (Xbox コミュニティでは「X-Code インタープリタ」とも呼ばれる) と呼ばれるプログラムが実行される。ジャムテーブルとは、ハードウェア初期化の文脈で使われる、読み取り・書き込み・単純な条件分岐のオペコードを含む値のテーブルを指す業界用語だ。一般的な PC の初期化には数百の操作が必要で、ジャムテーブルはコア初期化コードを肥大させることなくこの複雑さに対処するのに役立つ。ジャムテーブルを使うと、インストールされているメモリの種類や量といったユーザー設定可能なハードウェアパラメータにも対応できる、より柔軟な初期化が可能になる。Xbox の場合、ジャムテーブルインタープリタはフラッシュ ROM の下端付近からジャムテーブルのオペコードを取得し始める。(ジャムテーブルインタープリタが実装するオペコードは非常に強力で、ジャムテーブルのオペコードを使って Xbox 内の任意の場所へのデータの読み書きができることを覚えておいてほしい。)

ジャムテーブルインタープリタが終端オペコードを実行すると、プロセッサは MTRR (Memory Type Range Registers、各メモリ領域のキャッシュ可能性を宣言するレジスタ群) をクリアし、0xFFFF.A000 から始まる 16KB のメモリ領域の復号を開始する。このメモリ領域の復号に使われている暗号は、微妙な違いはあるが RC-4 暗号アルゴリズムによく似ている。リスト 6-1 に、Data Rescue 社の IDA Pro というツールの助けを借りてリバースエンジニアリングした C コードを示す(このツールについては次の章で詳しく説明する)。

```
// key initialization routine
unsigned char K[256]; // 0xFFFFC80 in flash
unsigned char S[256]; // 0x10000 in SDRAM
for( i = 0; i < 256; i++ ) {
    S[i] = i;
}
j = 0;
for( i = 0; i < 256; i++ ) {
    // RC-4 would do j = (j + K[i] + S[i]) % 256
    j = (j + K[i] + S[j]) % 256;
    // swap S[i], S[j]
    temp = S[i];
    S[i] = S[j];
    S[j] = temp;
}

// decryption routine
unsigned char cipherText[16384]; // 0xFFFFA000 in FLASH
unsigned char plainText[16384]; // 0x400000 in SDRAM
for( index = 0x4000, i = 0, k = 0; index > 0; index-- ) {
```

```

// xbox version
t = (S[i] ^ cipherText[k]) % 256;
plainText[k] = t;
// swap( S[i], S[t] );
temp = S[i];
S[i] = S[t];
S[t] = temp;
i = (i + 1) % 256;
k++;
}

```

リスト 6-1: フラッシュ ROM に見つかったダミー暗号の逆コンパイル。

この暗号で復号されたデータは、復号プロセスの最後に実行されるコードのブロックの
はずなのだが、ここで何かがひどくおかしいことが判明する——復号されたコードはガ
ベージだ。まったく機能しない。

コラム：メモリアドレスデコーディングのトリック

メモリの特定領域を物理的な実態とは異なる見せ方にする技術がいくつかある。Xbox のブー
トシーケンス解析に関係する二つのトリックは、エイリアシング (aliasing) とオーバーレイ
ング (overlaying) だ。

メモリのエイリアシングとは、二つのアドレスが同じメモリ位置を参照することで、通常はい
くつかのアドレスビットを無視することで実現する。例で示そう。3 ビットのアドレスを使う
システムを考える。3 ビットシステムでアドレス指定できるのは $2^3=8$ か所だ :000、001、010、
011、100、101、110、111。ここで 4 か所のメモリがあるとする。4 か所を識別するには 2 ビッ
トで十分だ :00、01、10、11。この 4 か所のメモリに 3 ビットアドレス方式でアクセスする場
合、アドレスビットの 1 ビットを無視しなければならない。最上位ビットを無視すれば、アド
レス 000 と 100 はどちらもメモリの位置 00 にマッピングされる。つまり位置 00 は、アドレ
ス 000 と 100 にエイリアスされているということだ。

メモリオーバーレイングは、帯域外の情報を使って異なるメモリバンクを切り替える技術だ。
秘密のメモリバンクを持ちたいとしよう。そのために、CPU と公開メモリ・秘密メモリの間に
セレクタを挿入する。このセレクタは、図 6-2 に示すように、秘密メモリまたは公開メモリの
いずれかのデータを CPU に提示することができる。結果として、アドレスセレクタを制御す
るプログラムが、秘密ブロックへのアクセス権も制御することになる。コンピュータが秘密メ
モリバンクのコードから起動した場合、秘密コード領域のプログラムはこの仕組みを使って、
公開メモリ内のプログラムを実行する前にセレクタを公開メモリに切り替えることで、自分
自身を隠すことができる。

Memory Address Decoding Tricks

A number of tricks exist that can be used to make regions of memory appear different in some way than their physical representation would indicate. The two tricks relevant to the analysis of the Xbox boot sequence are *aliasing* and *overlaying*.

Memory locations are aliased when two addresses refer to the same memory location, typically accomplished by ignoring a few address bits. To illustrate aliasing, consider a system using a 3-bit address. There are only $2^3 = 8$ unique locations addressable in a 3-bit system: 000, 001, 010, 011, 100, 101, 110, and 111. Now suppose that you have a memory with four locations; it requires only two bits to distinguish each of the four locations: 00, 01, 10, and 11. If you use our 3-bit address scheme to talk to this four-location memory, one of the address bits must be ignored. If the highest bit is ignored, then address 000 and 100 will both map to location 00 in memory. In other words, location 00 is *aliased* to addresses 000 and 100.

Memory overlaying is a technique where out-of-band information is used to select between different banks of memory. Let's suppose that we wish to have a bank of secret memory. To do this, we insert a selector between our public and secret memories, and the CPU. This selector can choose to present the CPU with data from either the secret memory or the public memory, as indicated in Figure 6-2. As a result, the program that controls the address selector also controls who has access to the secret block. If the computer starts up running code located in the secret bank of memory, a program in the secret code region can use this mechanism to hide itself by setting the selector to point at public memory before running programs located in public memory.

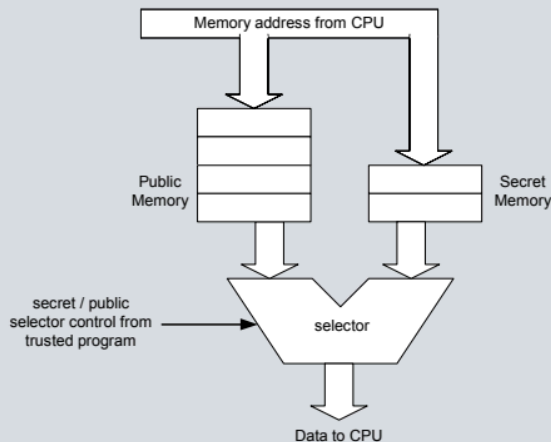


Figure 6-2: Memory overlaying to hide secret regions.

図 6-2: 秘密領域を隠すためのメモリアーバーレイ。

さらに、ジャムテーブルのオペコードも壊れているように見える。この現象は問題に取り組む他のハッカーたちによっても確認されており、コード変換エラーではないことが証明された。明らかに Xbox には表面からは見えないものがある。

この奇妙な動作を説明しようとする仮説と噂が出始めた。人気の高かった仮説には次のようなものがあった：

- **アドレス線またはデータ線のスクランブル。** どこかでアドレス線やデータ線が何らかの 1 対 1 マッピング関数によって反転または並び替えられている。スクランブル関数は初期化手順の一部としてチップセットにプログラムされている可能性があり、初期ブートブロックはプレーンテキストとして読めるが、残りのデータはスクランブルされているかもしれない。
- **二次暗号プロセッサ。** Xbox の別のプロセッサが実際の Xbox 初期化を担当しており、ROM のブートコードはダミーだ。
- **プロセッサ内蔵のブートコード。** プロセッサがプロセッサダイ上のコードブロックによって実際に初期化されており、ROM のブートコードはダミーだ。
- **チップセット内蔵のブートコード。** プロセッサは標準の Pentium と同じように動作するが、チップセットに ROM のダミーコードを上書きするブートコードが含まれている。

これらの仮説のほぼすべてについて、証明または反証する唯一の方法はハードウェア上で実験を行うことだ。たとえば、SMC (System Management Controller、システム管理コントローラ——Xbox がコンセントに挿さっている間は常にオンの 8 ビット自己完結型プロセッサ) がマシンのセキュアブートシーケンスに関与していないことを確認するため、ハッカーたちは SMC の全ピンの波形トレースをキャプチャし、SMC が初期化において重要な役割を果たす場合に予想されるイベントシーケンスと照合して分析した。

仲間のハッカーからの重要な観察は、0xFFFF.FFF0 のリセットベクタコードを変更しても Xbox が完全に正常に起動するというものだった。0xFFFF.FFF0 でプロセッサが最初に実行する命令が壊れていれば、マシンがクラッシュするはずだと誰もが考えるだろう。ところがマシンは完全に正常に動作した。この観察は、フラッシュ ROM の様々な部分を意図的に壊す実験によって検証された。結果は、フラッシュ ROM の驚くほど広い領域を壊しても Xbox の起動には何の影響もないというものだった。特に、0xFFFF.FE00 から 0xFFFF.FFFF までのブート初期化シーケンス全体をヌルにしても Xbox は問題なく起動した。

この発見だけで、フラッシュ ROM のブートブロックがダミーだという仮説が強く支持された。しかし疑問は残った——本物のブートコードはどこに保存されているのか？ 選択肢は三つ：二次暗号プロセッサの中、プロセッサの中、そしてチップセットの中だ。二次暗号プロセッサの仮説は、マザーボード上に起動時に暗号プロセッサの役割を果たすのに十分なパワーやアクティビティを持つチップがないという理由で却下された。

ブートブロックをプロセッサに格納する選択肢も、チップセットに格納するよりも可能性が低いと判断された。この分析の根拠はチップ製造のコスト経済学に基づいている。Pentium III プロセッサは多くの手作りブロックを持つ非常に複雑なデバイスであり、セキュアブートブロックを含むようにシリコンを修正するには、カスタムシリコンの製造に必要なマスクだけで約 25 万ドルの先行投資に加え、相当なエンジニアリングリソースが必要になる。さらに、Microsoft は当初 Xbox に AMD プロセッサを選んでしたが、土壇場で Intel に切り替えたという噂もあった。プロセッサコアにカスタムブロックが統合されていたとすれば、Microsoft は CPU ベンダーをそう簡単には切り替えられなかったはずだ。一方、nVidia のチップセットはシリコンコンパイラを使ってモジュール設計されているため、セキュアブートブロックのような機能を追加するのは技術的にずっと容易だ。さらに、Xbox のチップセットは Microsoft 専用で作られた nForce のカスタムバージョンで、Intel の FSB（フロントサイドバス、Front Side Bus）専用調整されている。そのため、セキュアブートブロック追加のコストは、そのようなプロジェクトにすでに割り当てられているエンジニアリングリソースとマスクセットに組み込める。

チップセット内の秘密 ROM オーバーレイに本物のブートコードがあるという仮説のもとで、残された課題は、コードがどのチップ（ノースブリッジかサウスブリッジ）に格納されているかを特定し、この秘密 ROM を抽出する方法を見つけることだった。秘密 ROM を抽出するための戦略がいくつか浮上した：

- Pentium の JTAG (Joint Test Action Group) 「バウンダリスキャン」機能を使って初期ブートコードのキャプチャを試みる。JTAG は専用のシリアルポートを通じてチップ上の全ピンの状態を読み書きできる診断バスで、非常に強力な汎用性の高いデバッグツールだ。
- プロセッサの FSB (フロントサイドバス) をプローブして、プロセッサに入るブートコードのキャプチャを試みる。

of a crypto processor. Storing the boot block in the processor was also deemed a less likely option than storing the boot block in the chipset.

The rationale for this analysis is based on the economics of building chips. A Pentium III processor is very complex with many handcrafted blocks, and modifying the silicon to include a secure boot block would require significant engineering resources as well as an upfront investment of about a quarter of a million dollars just for the masks needed to produce the custom silicon. In addition, it was rumored that Microsoft had originally chosen an AMD processor for the Xbox, and switched to Intel at the last minute. If custom blocks were integrated into the processor core, Microsoft could not have switched between CPU vendors so easily. On the other hand, nVidia's chipsets are designed modularly using silicon compilers, so it is technically easier to add warts like a secure boot block. Furthermore, the chipset in the Xbox is a custom build of the nForce made just for Microsoft, tailored specifically for an Intel front-side bus (FSB). As a result, the cost of adding a secure boot block could be rolled into the engineering resources and the mask sets already allocated to such a project.

Operating under the theory that the real boot code is located in a secret ROM overlay in the chipset, the challenges remaining were to determine in which chip (Northbridge or Southbridge) the code was stored, and how to extract this secret ROM. A few strategies for extracting the secret ROM presented themselves:

- **Use the JTAG “boundary scan” feature** on the Pentium to try to capture the initial boot code. JTAG is a diagnostic bus that allows you to read and set the state of every pin on a chip through a special serial port. It is a very powerful and versatile debugging tool.
- **Probe the processor FSB (Front Side Bus)** to try and capture the boot code as it enters the processor.

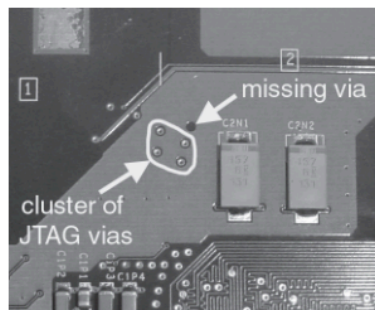


Figure 6-3: Missing JTAG via. Note how the filled in copper region (lighter area) has a hole where a via used to be. This is the result of a last-minute change to the board layout without recalculation of the fill regions.

図 6-3: 欠落した JTAG のビア。銅充填領域（明るい部分）に元々ビアがあった穴があることに注目。ボードレイアウトを土壇場で変更した際に充填領域の再計算をしなかった結果だ。

- ・ メモリスニファを取り付けて、メモリへの書き込み時に復号済みデータストリームのキャプチャを試みる。
- ・ 顕微鏡を使ってチップ表面から秘密ブートエリアの内容を読み出す。
- ・ サウスブリッジとノースブリッジチップ間のバスをプローブして、チップセットがプロセッサに送るブートコードのキャプチャを試みる。これはブートデータがサウスブリッジのどこかに格納されている場合にのみ有効だ。

これらの仮説はどれも簡単に検証できるものではなかった。フラッシュ ROM イメージの暗号解析に行き詰まった挫折したハッカーたちが次々と諦め、Xbox ハッキングへの取り組みは徐々に停滞した。僕も諦める組の一人になっていただろう（何しろ、あと数か月で博士論文を書き上げなければならなかった）——決意を固めたハッカーコミュニティが励ましをくれていなければ。

2001 年 12 月のクリスマス休暇中、IRC チャンネルとウェブフォーラムを通じてハッカー仲間と連絡を取り合い続けた。世界中の様々な境遇のハッカーたちが Xbox ハッキングの IRC チャンネルに集まっており、技術的なことも個人的なことも、彼らから学び、彼らと語り合うのが楽しかった。

1 月は全部博士論文 6 の執筆に費やして Xbox ハッキングは避けようと心に決めていたが、Xbox が採用した複雑で謎めいたセキュリティには抗えなかった。時間が経つにつれ、IRC チャンネルにたむろする少数のハードコアハッカーグループにハードウェアの専門家が重要だということがますます明らかになってきた。1 月末までには、Xbox セキュリティについての情報が面白すぎて無視できなくなっていた。

2 台目の Xbox を購入し、ホットエアガンで主要部品をすべて取り外し始めた。Xbox を分解することにはいくつかの目的があった。一つ目は、チップを取り外すことで Xbox 上の配線と接続がすべて露出し、マルチメーターの導通テストモードを使ってチップ間の接続を簡単に追えるようになることだ。二つ目は、興味深いチップをすべて熱い酸の槽に浸けてプラスチックの封止材を除去し、顕微鏡で解析できるようになることだ。最後に、Xbox を買って完全に分解したことで、動作する Xbox をプローブしたり改造したりする際の心理的な余裕が生まれた。（リバースエンジニアリングは庭仕事に似ている。手と膝を汚さないようにしながら庭を耕すのは難しい。だったら割り切って土の中に転がり込んだ方がいい。）

Xbox の分解の結果、Microsoft がハードウェアハッカーからコンソールを守るために講じた対策のいくつかが明らかになった。まず Pentium CPU の JTAG 接続を確認した。JTAG の信号はすべて——TRST# 信号を除いて——プロセッサ近くのタップしやすい抵抗器に便利にルーティングされていた。TRST# は JTAG インターフェースの初期化において重要な役割を果たす。興味深いことに、TRST# はアクセスしにくい場所の内部グラウンドプレーンに接続されており、JTAG メカニズムを恒久的に無効化していた。Xbox マザー

ボードをさらに調べると、TRST# 信号が土壇場で取り除かれた痕跡が見つかった。(欠落したビアの最大の証拠は、図 6-3 に示すように、JTAG 信号専用のビアクラスター近くの電源配線に、ビアとしてちょうどいい大きさの穴が開いていることだ。)

秘密 ROM を抽出するための JTAG アプローチへのさらなる打撃は、Intel の JTAG スキャンコードが独自仕様 (プロプライエタリ) だということだ。秘密ブートデータを抽出するのに使えるレベルまでそのコードをリバースエンジニアリングすることは、それだけで一つの大きなプロジェクトになる。

JTAG アプローチを断念し、次に試みたのは CPU、GPU、そして MCPX (Xbox のサウスブリッジチップ) のパッケージを剥がしてベアダイを顕微鏡で観察し、ROM の候補となる構造を探すことだった。パッケージの取り外し (「デキャップ」) は、チップを熱した発煙硫酸に浸けることで行った。(家庭でのこの方法はお勧めしない。一度、毒性と腐食性のある液体を全身にこぼしたことがあるが、幸い僕の肌ではなく保護具が犠牲になった。発煙硫酸は燃えさかる炎よりも速く有機物を溶かす。) 同じく非常に毒性と危険性の高い発煙硝酸も使える。僕自身は試していないが、報告によると発煙硝酸はエポキシの封止材を除去するのにより効果的で、特に選択的なパッケージ除去が必要な場面では有用だという。

従来の可視光顕微鏡を使った手動検査は多少の望みを与えてくれたが、この技術は光の物理学によって限界がある。光の最短波長は 450nm (青色に相当) であるため、最高の可視光顕微鏡技術でも 150nm のトランジスタを解像することはできない。秘密コードが、従来のアレイ ROM 構造——光学顕微鏡で識別できる上層の金属層に、1 か 0 を定義する金属線がエッチングされている構造——を使ってチップに格納されていることを期待していた。ハードワイヤード ROM 構造を使う動機はコストだ。フラッシュ ROM とヒューズベースの PROM はシステムコストを大幅に増加させる余分な製造工程が必要だが、上層の金属層を使うことは設計者のリスク管理の観点から動機付けられる。上層の金属層は最も粗い層 (光学顕微鏡で解像できるほど粗い) であり、ROM コードにバグがあった場合に変更するのに最もコストが低い層だ。また、初期の動作確認 (ブリングアップ) 段階では、FIB (Focused Ion Beam、集束イオンビーム) 装置と呼ばれるチップ修正機を使って上層をカット・ジャンパするのが最も容易だ。残念ながら、顕微鏡でチップをざっと確認したところ、そのような構造は見つからなかった。

この時点で秘密 ROM を抽出するために残された唯一の選択肢は、動作中の Xbox ハードウェアをプローブして、Xbox プロセッサへのコードのロード中にそれをキャプチャすることだった。サウスブリッジチップとフラッシュ ROM の上流でコードを盗み聞きするということは、フロントサイドバス、ノースブリッジ-サウスブリッジ間バス、またはメインメモリバスのいずれかをプローブすることを意味した。これらのプローブアプローチの実行上のトレードオフについては、次章 (第 7 章) で基本的なセキュリティの概念を簡単に紹介した後、第 8 章で議論する。

注

1. Andy Green によるテストクリップアプローチについてはこちらのページに詳しい記録がある：<http://www.warmcat.com/milksop/milksop.html>

2. Visor によるロジックアナライザアプローチについてはこちらにまとめられている：
<http://www.xboxhacker.net/visor/aXventure1.txt>
 3. Emulation Technologies (<http://www.emulation.com>) が幅広い種類のソケットを提供している。Xbox のフラッシュ ROM 用の具体的な型番は S-TS-SM-040-A だ。
 4. <http://www.xenatera.com/bunnie/proj/flashburn/fb.html>
 5. <http://www.xenatera.com/bunnie/proj/fb2/>
 6. スーパーコンピュータアーキテクチャ、データとスレッドの移動、フォールトトレランス、高速低遅延ネットワーク、または大規模マルチスレッドマシンに興味があれば、こちらで博士論文を読むことができる：<http://www.xenatera.com/bunnie/phdthesis.pdf>
-

第7章 セキュリティ入門

前の章で体験したように、Xbox をハックするには、ハードウェアとファームウェアのハッキングに加えて、セキュリティハッキングの知識も必要だ。本章では、まず洗練されたセキュリティを家庭用ゲーム機というありふれた製品に組み込む動機を探り、Xbox のセキュリティメカニズムを理解・評価するために必要な基本原理とアルゴリズムを解説する。

セキュリティって本当に必要なの？

ビデオゲームコンソールは、大多数の人にとってはただのおもちゃだ——低価格の家庭用電子機器にすぎない。それなのに、なぜ Microsoft はシステムをこれほど手厚く守ろうとしたのだろうか？ セキュリティハッキングのゲームでは、守る側の動機を理解することが、突くべき弱点を見つける鍵になることが多い。

暗号技術それ自体はセキュリティではない。セキュリティを実現するための手段にすぎない。本当のセキュリティは、エンドユーザーを含めたシステムアーキテクチャ全体を対象にする。Kevin Mitnick が最近の Slashdot インタビューで語ったように、「……セキュリティとは柵から買ってこられる製品ではなく、ポリシー、人、プロセス、テクノロジーで構成されるものだ」¹。セキュリティとは本質的に社会的な概念だと僕は思っている。実際の生活でも、窓を開けたまま玄関のドアに鍵をかけておけば、人々はわざわざ窓から入ったり錠前を壊したりしない——どちらも技術的には難しくないのに。施錠されたドアと開いた窓が機能するのは、ドアの鍵が主にシンボリックな意味を持つからだ。侵入者は意識的な違反行為に踏み出す必要があり、それだけで犯罪者と善良な人々を分けるのに十分だ。Sony の PlayStation はこのフロントドアロック型セキュリティの好例だ。

ゲームのコピープロテクションに使われているメカニズムはシンプルで、暗号技術は一切使われておらず、安価なハードウェア改造で簡単に回避できる。それでも、販売数のデータを見れば、PlayStation ゲームを購入する習慣は廃れていない——フロントドアロックはちゃんと機能しているのだ。

Microsoft もフロントドアロックのバリエーションを Xbox に採用している。Xbox のゲームソフトは、（今のところ）コピー不可能な DVD-9 フォーマット——片面 2 層のメディア規格——で配布されている。一方、ユーザーが書き込める DVD は常に DVD-5 フォーマット、つまり片面 1 層だ。一方の層に書き込む際にもう一方の層の記録面を傷めずに済む書き込みシステムを作ることが難しいため、DVD-9 対応バーナーが近い将来登場する可能性は低い。セキュリティデータを DVD-9 ディスクの 2 層に分散させ、ゲームの実行ファイルを DVD-9 メディアからのみ起動できるよう要求することで、Microsoft はかなり効果的なフロントドアロックを実現している。DVD-9 フォーマットを要件として課すことで、潜在的なゲームコピーの試みをハードウェア改造が必要な領域に追い込んでいるわけだ。

では、なぜ Microsoft は Xbox にこれほど複雑なセキュリティスキームを投資するリスクを冒したのだろうか？ Xbox セキュリティシステムの残りの部分——セキュアブートセクタ、署名された実行ファイル、トラスト関係、暗号化・認証されたネットワークプロトコル——の主目的は、著作権侵害対策ではない可能性がある。

動機の一つとして、Xbox コンソールをゲーム以外の用途に使えないようにすることが考えられる。Xbox コンソールはほぼ 100% 普通の PC という特異な立ち位置にある。GameCube や PlayStation 2 とは違い、適切な BIOS さえあれば Xbox でそのまま動きそうなソフトウェアが山ほど存在する。さらに厄介なことに、Microsoft はコンソールのハードウェアで競合他社よりはるかに大きな損失を出している。最新の小売価格 199 ドルを前提にすると、その損失は 1 台あたり 200 ドルに達するという試算もある。だから、補助金を投じた GNU/Linux ボックスを売っているだけにならないよう努めたわけだ。しかし、これさえもおそらく Microsoft の主目的ではない。Xbox は RAM が 64MB、キーボードもマウスも付属せず、当時の基準でもプロセッサが比較的遅い。2002 年末に Walmart で 200 ドルで買えた Microtel PC より魅力的とはいいがたい。さらに Microsoft には潤沢な資金がある。Xbox が Sony の PlayStation 2 を超える市場支配を確立できれば、数十億ドルの先行投資損失は——手元に積み上がる約 400 億ドルの現金と比べれば——相対的に小さい。だから、Xbox セキュリティの重要な使命は、コンソールの代替用途を防いだり海賊版を阻止したりすることではないのかもしれない。

おそらく Xbox の複雑なセキュリティの本当の理由は、Microsoft のオンラインゲームサービスである Xbox Live の成功を保証することにある。Microsoft のマーケティングとプレスリリースは、ハードウェア販売の起爆剤として Xbox Live の成功に賭けていることを示している。さらに、Xbox Live はサブスクリプションサービスで、ローンチから 1 年後にはユーザーが月額料金を払う必要がある。Xbox Live でユーザーを虜にできれば、ハードウェアでかなりの損失を出しても、一気に Xbox ビジネスは十分な収益性を持つように見えてくる。「オンラインゲームのディズニールンド」とも形容される Xbox Live の目標は、公平で洗練されたゲーム体験を提供することだ。そのサービス価値の核心は「チーターがいない」ことだ。誰もズルをしていないと保証するには、ユーザーを Xbox Live が管理するレジストリに対して認証させ、ゲームの状態を安全かつ改ざん不可能に保つ必要がある。さらに、ゲームソフトにはパッチが当たっていないことも求められる。もっと

切実なのは、ほんの一握りのチーターが大勢のユーザーのゲーム体験を台無しにするという事実だ。そうなると、DVD-9 フォーマットが提供するフロントドアセキュリティだけでは不十分に見えてくる。コンピュータにそこそこ詳しい20代の男性ゲーマーが何百万人も存在する状況で、彼らの倫理観と誠実さに事業の成功を賭けるのは分が悪い。ハードウェアは信頼できなければならず、ネットワーク接続は安全でなければならず、実行ファイルは署名・封印されていなければならない。

ハードウェアは信頼できなければならない——この一文は繰り返す価値がある。信頼できないユーザーを相手にするとき、クライアントとの信頼関係を築く唯一の方法は、すべてのハードウェアの内部に「信頼の種」が存在することだ。だからこそ Microsoft は、すべてのクライアントに改ざん耐性（耐タンパー性）のあるハードウェアを組み込み、アテステーション——特定のデータ（例えばプレイヤーの識別情報やゲーム状態）が、確かに汚染されていないソフトウェアとハードウェアから生成されたことを証明する機能——を可能にする必要があった。耐タンパーハードウェアはアテステーション機能を直接実装する必要はないが、少なくとも認証前にシステムが信頼できる状態にあることを保証しなければならない。

ハードウェアの信頼性を確保する方法はいくつかある。最も力技なのは、ハードウェア全体を物理的に保護することだ。現金自動預払機（ATM）は物理的に安全なハードウェアの典型例だ。厚い鉄板で密封し、侵入センサーを張り巡らせた ATM を物理的に破って改造するのは難しい。ただ、効果的ではあっても、ビデオゲームコンソールに適用するには現実的でなく、コストも高すぎる。

より経済的な解決策は、システムの残りの部分を「計測」できる小さな耐タンパーハードウェアを使うことだ。この種の計測には通常、暗号学的ハッシュ関数が使われる。すべての信頼計測値が期待値と一致すれば、システム全体が信頼できると結論づけられるかもしれない。

「かもしれない」と言ったのは、このスキームが依然として中間者攻撃（man-in-the-middle attack）に対して脆弱だからだ。中間者攻撃とは、攻撃者が2者間でやり取りされる情報を自由に改ざん・制御できる攻撃の総称だ。計測クエリに対して偽の正常データを送り込まれてしまう。この中間者攻撃の弱点があるため、システム計測に極めて高度な耐タンパーモジュールを使うことに意味はない。パッケージ化されたシリコンチップ1個で十分だ——プリント基板上を流れる計測データを傍受・偽装する方が、チップのエポキシパッケージを物理的に破って回路を改造するよりも一般的に簡単だからだ。

信頼計測システムは「一度だけ計測する」アプローチで実装できる。プロセッサのコールドブートシーケンスから始めて、実行前にすべてのコードを信頼性について計測する。プロセッサが一切の信頼されないコードを実行しないなら、信頼できないものは何もないはずだ。このスキームに必要な耐タンパーハードウェアはごくシンプルでよい——コールドブートコード、つまり「信頼の種」を格納する耐タンパー ROM だ。計測と検証のプロセスで使われる暗号方式は、通常ハッシュと公開鍵暗号の組み合わせだ。公開鍵暗号がこの用途に好まれるのは、正当なコードセグメントを生成するために必要な秘密鍵が、ハー

ドウェアベンダーだけが知るシークレットだからだ。このスキームも多種多様な中間者攻撃、純粋な暗号解析攻撃、システム実装上の攻撃に対して脆弱であることには変わらない。

暗号技術入門

暗号 (cipher) (名詞) : 1 a: ゼロ b: 重さ・価値・影響力が皆無の存在 : 無名の人。2 a: 意味を隠すためにテキストを変換する方法——「CODE」の第2定義と対比せよ 2

暗号はそれ自体では何のセキュリティも提供しない。正確に言えば、暗号がセキュリティを提供できるのは、鍵が安全で、アルゴリズムが強固で、システムにバックドアがない場合だけだ。強力な暗号で暗号化されたCD-ROMを手渡され、スーパーコンピュータと共に密室に閉じ込められたとしよう——おそらく太陽が超新星爆発を起こす前にCD-ROMを復号することはできない。一方、CD-ROMを暗号化しているマシンの動作を観察・調査できる状況なら、暗号化などほぼ無意味だ。キーボードを盗み聞きして暗号化鍵を入手できる。あるいは、コンピュータのメモリをダンプして、鍵を知らなくても平文を得られる。

Xboxもこれによく似た後者の状況に置かれている。最終的には、Xboxは正規ディスクに収録されたプログラムにアクセスして実行しなければならない。さらに、XboxのPentium CPUは、正規の命令と不正な命令を区別できない。おまけに、ユーザーはXboxのハードウェアを自由に調査・改造できる。だから、Xboxが強力な暗号を使っている、鍵のセキュリティには疑問符が付き、システムへのバックドアが存在する可能性もある。

本節では、Xboxで使われている暗号アルゴリズムの種類を簡単に説明する。実装上の観点や実際の運用上の問題に焦点を当てる。Xboxのセキュリティシステムに対する攻撃手法を理解・評価するために、これらのアルゴリズムを知っておく必要がある。

[!NOTE] 暗号技術の理論的な側面については、僕の能力と本書の範囲を超えているため扱わない。暗号に興味のある読者は、Bruce Schneierの名著Applied Cryptography(John Wiley & Sons)をぜひ参照してほしい——僕の暗号知識のほとんどはその本から来ている。暗号に精通している読者は、本章の残りの部分をざっと眺めるかスキップしても構わない。

暗号アルゴリズムの種類

Xboxで使われている主要な暗号アルゴリズムの種類は以下のとおりだ。

- ・ ハッシュ (Hash)
- ・ 共通鍵暗号 (Symmetric cipher)
- ・ 公開鍵暗号 (Public key cipher)

ハッシュにはいくつかの種類がある。暗号学的なハッシュは、大量のデータを「要約」あるいは「ダイジェスト」するために使われる。要約結果は固定長の数値で、通常100～200ビット程度。一方、元のデータはほぼ任意のサイズで構わない。ハッシュの最も重要な特性は、一方向計算だということだ。つまり、ハッシュの計算は簡単だが、同一のハッシュダイジェストを持つデータ列を見つけたり、ハッシュから元データを逆算したりするこ

とは非常に難しい（「非常に難しい」の意味については、コラム「とてつもなく難しい問題」を参照）。

同一のハッシュ値を生成する2つのデータ列を見つけることへのハッシュの耐性を「衝突耐性（collision resistance）」と呼ぶ。一般に、優れた n ビットのハッシュが衝突を引き起こすには、約 $2^{n/2}$ 個のランダムなデータ列をハッシュして比較する必要がある。ハッシュは計算が非常に高速で衝突耐性も高いため、大量のセキュアなデータ領域でビットが変化していないかを検出するために頻繁に使われる。多くの用途では、メッセージ全体を暗号化する計算コストをかける代わりに、暗号化されたメッセージのハッシュだけを付加する方式で十分だ。

共通鍵暗号は、暗号化と復号の鍵が互いに簡単に導出できるアルゴリズムだ。多くの場合、暗号化鍵と復号鍵は同じだ。共通鍵暗号は、鍵スケジュール（key schedule）と暗号関数で処理されたデータを組み合わせる混合関数（mixing function）を使う。この混合は、ブロック暗号のように1ブロックのデータに対して複数回繰り返されることもあれば、ストリーム暗号のように1回だけ行われることもある。共通鍵暗号の基本的な演算はどれも計算上シンプルなので、大量データを暗号化する際の優先的な方法として使われる。

混合関数の典型的な例は、XOR、剰余加算、剰余乗算だ。最もシンプルな XOR 演算には「任意の数とそれ自身を XOR するとゼロになる」という性質がある。XOR 演算は記号 $\oplus$ で表され、通常の算術の性質（交換法則、結合法則、分配法則など）をすべて持つ。

$$(A \oplus B) \oplus B = A \oplus (B \oplus B) = A \oplus 0 = A$$

つまり、 A がメッセージで B が鍵なら、 $(A \oplus B)$ が暗号文となり、再び B と XOR するだけで平文に戻せる。

鍵スケジュールは、比較的短い鍵を取り、その情報を長いビット列に展開するアルゴリズムだ。鍵スケジュールは、鍵データをより大きなデータブロックに拡散させ、暗号文と鍵の関係を不明瞭にするために使われる。

コラム：とてつもなく難しい問題

暗号関数はすべて、すべての演算子が与えられれば結果を計算するのは簡単だが、結果だけが与えられた場合に演算子を求めることが非常に難しい数学的アルゴリズムに基づいている。暗号関数のセキュリティとは、まさにこの「結果だけから演算子を求めることの難しさ」そのものだ。「非常に難しい」とはどういうことか、少し考えてみよう。

共通鍵暗号の AES（Advanced Encryption Standard）を例に取ろう。128 ビット鍵を使い、差分暗号解析や線形暗号解析など現在知られているすべての解析的暗号解析攻撃に対して強固だとされている。「解析 X に対して強固」とは、ブルートフォースで鍵や平文を回復するために必要な演算回数以上の演算が解析 X でも必要になる、という意味だ。ブルートフォース探索とは、非常に高速なコンピュータで 2^{128} 通りすべての鍵を試して元のデータを回復しようとするのだ。

今日広く使われている暗号アルゴリズムのほとんどは、すべての既知の暗号解析技術に対して強固なので、重要な指標はブルートフォース攻撃の強度だ。

少し古い 56 ビット暗号の DES (Data Encryption Standard) は「とてつもなく難しい問題」とは言えない。FPGA (Field Programmable Gate Array、現場でプログラム可能なゲートアレイ) を使ったマシンを構築すれば、約 222 鍵/秒/ドル (222 は約 400 万) というコスト効率で鍵を解読できる。この数字はムーアの法則に従い時間とともに増加する。現在、1 週間ほど待つ覚悟があれば、高級車 1 台分の価格で DES 鍵を回復できる。銀行が DES でアカウントデータを暗号化していないことを祈るばかりだ！

DES の後継である AES は、128・192・256 ビット鍵が使えるが、どれも総当たり攻撃には事実上不可能なほど大きい。NIST (米国国立標準技術研究所) が公開した AES Q&A (<http://csrc.nist.gov/encryption/aes/aesfact.html>) によると、ブルートフォースで 1 秒に 1 つの DES 鍵を回復できるほど強力なマシンが 128 ビットの AES 鍵を回復するには、149 兆年かかる。256 ビット鍵に対するブルートフォース攻撃の強度について Bruce Schneier は Applied Cryptography の中で、使用エネルギーの観点から興味深い分析を行っている。熱力学的に理想的なコンピュータを使っても、2192 まで数えるだけで太陽の年間エネルギー出力の 32 倍以上のエネルギーが必要だという。(ただし、以上はすべて最も効率的な攻撃がブルートフォースであるという前提に立つ。誰かがアルゴリズムの弱点を発見し、はるかに効率的な攻撃手法を編み出す可能性はゼロではない。暗号の強度を徐々に削るような新しい解析技術は常に生まれ続けている。)

一方、公開鍵暗号は、素数の乗算や剰余べき乗など、逆算が難しいさまざまな数学的演算に基づいている。その結果、多くの公開鍵暗号では鍵空間が疎になるため、同等のセキュリティを持つ共通鍵暗号より長い鍵が必要になる。例えば、RSA 公開鍵暗号の鍵長は通常数千ビットだ。

RSA 公開鍵長と共通鍵暗号の鍵長の正確な対応関係は不明だ。RSA のセキュリティは大きな素数の積を因数分解することの難しさだと考えられているが、アルゴリズムにはまだ発見されていない攻撃が存在する可能性もある。大きな素数の積を因数分解する実質的な難しさは、コンピュータ技術の進歩 (ムーアの法則) だけでなく、二次ふるい法や一般数体ふるい法の発明・改良といった整数論の進歩によっても低下する。

1999 年 8 月、研究者グループが数体ふるい法を使い、因数分解の実行時間のセットアップを含め、暦上 7.4 か月で 512 ビットの素数を因数分解した (<http://www.rsasecurity.com/rsalabs/challenges/factoring/rsa155.html>)。さらに、量子コンピュータのような新技術は多項式時間での素数因数分解を可能にすると期待されている。ただし、興味深い素数を因数分解するのに十分な量子コンピュータを実際に構築できるかどうかはまだ議論中なので、息をひそめて待つ必要はない。

RSA Security 社は現在、ほとんどの企業用途には 1024 ビット、「極めて貴重な鍵」には 2048 ビットの鍵長を推奨している (<http://www.rsasecurity.com/rsalabs/faq/3-1-5.html>)。Bruce Schneier は Applied Cryptography 第 2 版で、2304 ビットの公開鍵は 128 ビット共通鍵と同等のセキュリティを提供し、1792 ビットの公開鍵は約 112 ビット共通鍵に対応すると推定している。

Xbox のセキュリティスキームを読み進める中で、これらの基本的な指針を念頭に置いておくといい——ブルートフォースでセキュリティを破ることがいかに難しいかを理解するために。ハッキングフォーラムには「なぜこれらの鍵に対して分散鍵探索プロジェクトを始めないのか？」という書き込みが繰り返し投稿される。これで答えがわかっただろう。

一般的な共通鍵ブロック暗号の暗号関数は、精巧に設計された換字・置換・圧縮・拡張の組み合わせとして実装される。これらの関数は平文を「混乱」させ「拡散」させるためのものだ。暗号関数のわずかな変化も、通常は暗号のセキュリティに大きな影響を与える。

共通鍵暗号では暗号化鍵と復号鍵が密接に関係しているため、特定のセキュリティ用途での利用が難しくなる。例えば、暗号化された文書をメーリングリストに配布したい場合、受信者全員が事実上僕の暗号化鍵を知らなければ文書を読めない。また、初めて連絡する相手と通信を開始するのも難しい——どこかの時点で鍵を相手に送る必要があり、その送信を傍受した第三者が鍵を盗んで、その後のすべてのメッセージを読んだり、偽造したり、改ざんしたりできてしまう。

公開鍵暗号は、暗号化と復号に異なる鍵を使うアルゴリズムだ。この理由から非対称暗号 (asymmetric cipher) とも呼ばれる。公開鍵暗号の最大の利点は、一方の鍵を秘密にできることだ。これにより、信頼できないユーザーとデータをやり取りしても、そのユーザーが他の保護されたコンテンツを偽造したり読んだりできないようにできる。公開鍵アルゴリズムの欠点は、一般的に計算が複雑なため共通鍵暗号より遅いことと、同等のセキュリティを得るために長い鍵が必要なことだ。そのため、大量データのやり取りが必要な場合、公開鍵暗号は通常、高速な共通鍵暗号のための鍵を暗号化するために使われ、バルクデータ自体は共通鍵暗号で暗号化される。この共通鍵暗号の鍵はランザクションごとにユニークにできるため、「セッション鍵 (session key)」と呼ばれることが多い。

SHA-1 ハッシュ

SHA-1 (Secure Hash Algorithm 1) は、米国政府が FIPS (Federal Information Processing Standard) 刊行物 180-1 (<http://www.itl.nist.gov/fipspubs/fip180-1.htm>) で推奨している安全なハッシュアルゴリズムだ。NSA (米国国家安全保障局) が考案し、Ronald L. Rivest の MD4 メッセージダイジェストアルゴリズムをベースにした SHA-1 は、264 ビット未満の任意の長さのメッセージを処理して 160 ビットの出力を生成する。

SHA-1 ハッシュアルゴリズムは確定的な 160 ビットのシード状態から始まる。この状態は 512 ビットのメッセージデータブロックと 4 ラウンドかけて混合される。各ラウンドは一連の非線形関数、ローテーション、シフト、XOR で構成される。あるラウンドの結果が次のラウンドの計算のシードになる。一般に、同一のハッシュ値 (つまりハッシュの衝突) を持つ 2 つのメッセージを見つけるには、280 個のランダムなメッセージを生成してハッシュし「同時に」比較する必要がある。この「同一ハッシュを持つ 2 つのランダムなメッセージを見つける」ことは「誕生日攻撃 (birthday attack)」と呼ばれ、「誕生日のパラドックス」という確率論的現象にちなんでいる——23 人がいる部屋で 2 人が同じ誕生日を共有する確率は 50% を超える、という現象だ。一方、特定のメッセージと同じハッシュ値を持つメッセージを見つけるには、2160 個のランダムなメッセージを生成・比較する必要がある。このように、ハッシュ関数の強度はその使われ方に大きく依存する。

TEA

TEA (Tiny Encryption Algorithm、タイニー暗号化アルゴリズム) は、ケンブリッジ大学コンピュータ研究所の David Wheeler と Roger Needham が開発した。(開発者の TEA

のウェブページは <http://vader.brad.ac.uk/tea/tea.shtml> にある。本節の内容の多くはそのページから引いている。）

```
void encipher(unsigned long *const v, unsigned long *const w,
              const unsigned long *const k)
{
    register unsigned long
        y=v[0], z=v[1], sum=0, delta=0x9E3779B9,
        a=k[0], b=k[1], c=k[2], d=k[3], n=32;
    while(n-->0) {
        sum += delta;
        y += (z << 4)+a ^ z+sum ^ (z >> 5)+b;
        z += (y << 4)+c ^ y+sum ^ (y >> 5)+d;
    }
    w[0]=y; w[1]=z;
}
```

リスト 7-1：ANSI C による TEA アルゴリズム 3

その名のとおり、TEA はリアルタイムデータストリームの暗号化や、プロセッサ性能とストレージが限られた組み込みアプリケーションに適したコンパクトで高速な暗号アルゴリズムだ。128 ビット鍵を使い、64 ビットのデータを 1 回の処理単位として扱い、32 ラウンドそれぞれでシフト、XOR、加算だけを使う。（リスト 7-1 と図 7-2 に示したアルゴリズムは、32 ビット汎用プロセッサへの実装に最適化されている。）

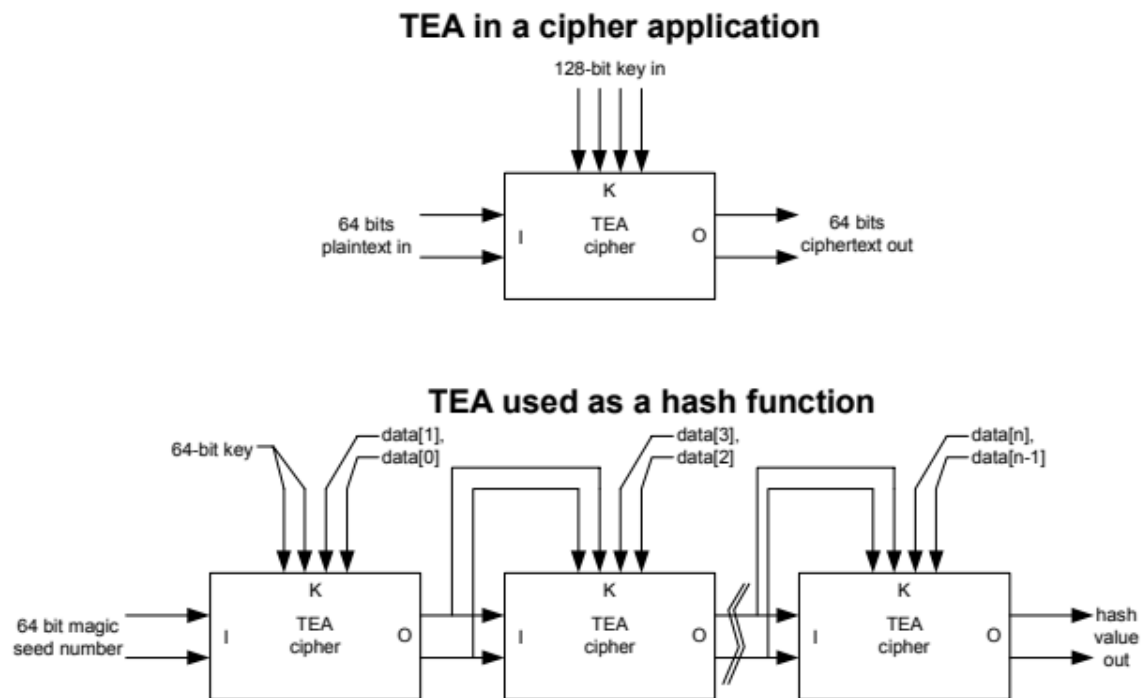


図 1: Figure 7-1：TEA 暗号の使用シナリオ

図 7-1 (上) : TEA 暗号の使用シナリオ。TEA を暗号として使う場合 (128 ビット鍵で 64 ビットブロックを暗号化) と、ハッシュ関数として使う場合 (前の出力を次の TEA 演算の鍵入力として連鎖させる) を示している。

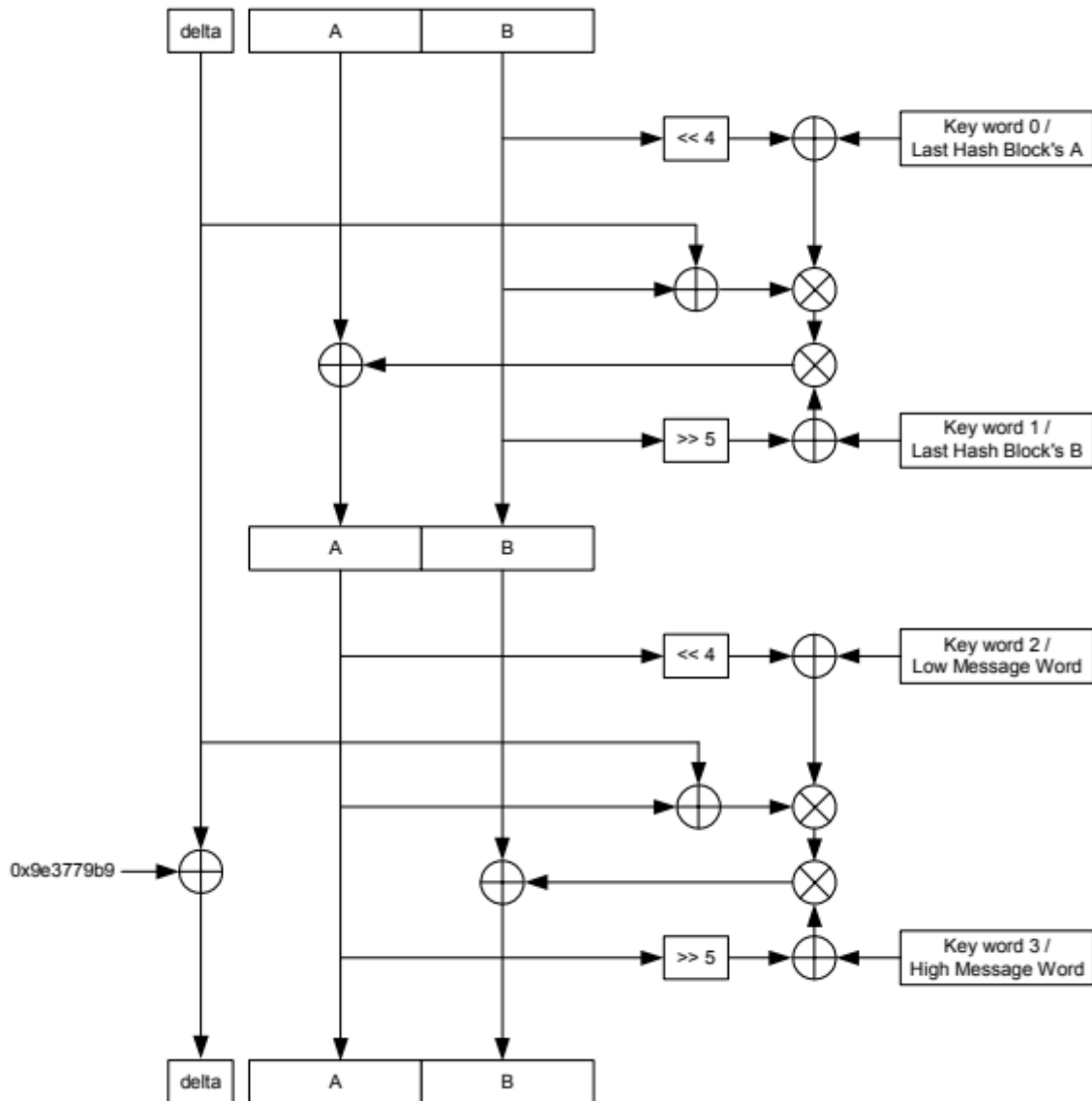


図 2: Figure 7-2 : TEA の内部構造

図 7-2 (下) : TEA の内部構造。TEA の 1 ラウンドを示す図で、完全な暗号では 32 回繰り返される。右側のボックスに、暗号としての使用とハッシュ関数としての使用それぞれの鍵スケジュールを示している。

この小型の TEA アルゴリズムは、データの暗号化・復号に使うと十分に安全だと考えられている。ただし、Xbox では TEA は暗号化には使われていない。実際には改変されたデービーズ・マイヤーモード (Davies-Meyer mode) で動作させることで、ハッシュ関数と

して使われている。ハッシュ対象の領域は 64 ビットのブロックに分割される。これらのソースデータブロックが TEA への鍵入力の半分として使われる。もう半分の鍵入力はその前の TEA 演算の結果から得られ、最初の TEA 演算にはマジックナンバーが使われる。

結果は図 7-1 に示す 64 ビットのハッシュ関数となる。TEA の計算効率が高いことから、このハッシュは誕生日攻撃に対して脆弱で、衝突を見つけるのに平均で 232 個のメッセージペアをテストすれば十分だ。Xbox の使用シナリオでは誕生日攻撃は適用できないものの、Xbox はハッシュを 2 回実行し、それぞれ異なるマジックナンバーシードを使って結果を連結し、128 ビットのハッシュ値を生成している——おそらくブルートフォース攻撃を無効化しようとする試みだろう。

残念ながら、TEA には鍵スケジュールに弱点がある。すべての TEA 鍵には 4 つの関連鍵が存在する。つまり、どの鍵についても、同じ入力データで同じ暗号文を生成する 3 つの別の鍵を生成できてしまう。関連鍵の生成は、鍵ビットのペア（ビット 31 と 63 が 1 つのペア、ビット 95 と 127 がもう 1 つのペア）を反転するだけだ。これにより、TEA はハッシュ関数としての使用には適さない。この弱点は、CRYPTO 1996 で発表された John Kelsey、Bruce Schneier、David Wagner による論文「Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and triple-DES」に詳しく記載されており、後に Andy Green が率いるチームがこの弱点を利用して Xbox セキュリティスキームの第 2 バージョンを破ることになった。

RC-4

RC-4 (Ron's Code 4 または Rivest Cipher 4) は、Ron Rivest による可変長鍵のストリーム暗号だ。RC-4 の核心は鍵ストリームジェネレータで、暗号学的な擬似乱数生成器 (CPRNG : Cryptographic Pseudo-Random Number Generator) と考えることができる。CPRNG の出力は平文ストリームと 1 バイトずつ XOR され、暗号文が生成される。復号も同様に行われる。大まかに言うと、ジェネレータは最大 256 バイト (2048 ビット) の値 (鍵) で「シード」される。鍵が 256 バイトより短い場合は、シードとして使う前に 256 バイトになるまで繰り返される——これにより可変長鍵が使える。Xbox では鍵は 16 バイト (128 ビット) で、この暗号は「RC-4/128」と呼ばれる。

```
typedef struct rc4_key {
    unsigned char state[256];
    unsigned char x;
    unsigned char y;
} rc4_key;

void prepare_key(unsigned char *key_data_ptr, int key_data_len,
    rc4_key *key) {
    unsigned char swapByte, index1, index2;
    unsigned char* state;
    short counter;

    state = &key->state[0];
    for(counter = 0; counter < 256; counter++)
```



```

        state[counter] = counter;
    key->x = 0; key->y = 0;
    index1 = 0; index2 = 0;
    for(counter = 0; counter < 256; counter++) {
        index2 = (key_data_ptr[index1] + state[counter] +
            index2) % 256;
        swap_byte(&state[counter], &state[index2]);
        index1 = (index1 + 1) % key_data_len;
    }
}

void rc4(unsigned char *buffer_ptr, int buffer_len, rc4_key *key) {
    unsigned char x, y, xorIndex;
    unsigned char* state;
    short counter;

    x = key->x; y = key->y;
    state = &key->state[0];
    for(counter = 0; counter < buffer_len; counter++) {
        x = (x + 1) % 256;
        y = (state[x] + y) % 256;
        swap_byte(&state[x], &state[y]);
        xorIndex = state[x] + (state[y]) % 256;
        buffer_ptr[counter] ^= state[xorIndex];
    }
    key->x = x; key->y = y;
}

```

リスト 7-2：Usenet へのオリジナルポストより C による RC-4 コード 4

RC-4 は強力な暗号だと考えられているが、WEP (Wired Equivalent Privacy) などの欠陥あるシステムで悪用される鍵スケジューアルゴリズムの既知の弱点がいくつかある。Scott Fluhrer、Itsik Mantin、Adi Shamir が「Weaknesses in the Key Scheduling Algorithm of RC4」(2001 年 8 月の第 8 回暗号技術に関する国際ワークショップで発表) でこれらの弱点を詳述している。ただし、これらの弱点は Xbox の RC-4 実装に対しては適用できない。

ただし、Xbox セキュリティ第 1 バージョンでの RC-4 の使い方に潜在的な問題がある。Xbox で RC-4 は x86 コードのストリームを暗号化するために使われており、復号されたコードの完全性を確保する有意義なチェックが行われていない。これは、暗号文を変更すると、Xbox が実行するコードも変化することを意味する。肝心なのは、意味のあるコード変更を引き起こすような暗号文の変更を特定することだ。RC-4 は 1 バイトずつ暗号化し、x86 オペコードは最短で 1 バイトとして表現できるため、暗号文を変化させることで既知の位置の命令を「ブルートフォース」するのに必要な試行回数は最大でも $28 = 256$ 回だ。

どの位置をブルートフォースすべきかを特定するのはやや難しいが、キャッシュがオンになっている状態でも、暗号文ビットを変化させて命令フェッチのパターンに何が起こるかを観察することで、多くの情報が引き出せると思う。目標は、ジャンプオペコードの

オペランドの位置を特定し、セキュアなプログラムがセキュアでないメモリ領域にジャンプするようにジャンプ先を変更することだ。古典的なボードゲーム「バトルシップ（海戦ゲーム）」のプロセスに似ている。1 キロバイトのコードを推測する作業は最大 218 回の試行しか必要としないことも念頭に置いてほしい。この推測プロセスは、ロジックアナライザを ROM エミュレータとホストコンピュータ上で動くコントロールスクリプトで連携させることで自動化できる。

RC-4 の背景は実に興味深い。RC-4 は 1987 年に Ron Rivest が発明し、RSA Security 社が企業秘密として保持していたが、1994 年に匿名でサイファーパンクスメーリングリストに投稿されて公開された（リスト 7-2 参照）。シンプルさと堅牢さという美点から、RC-4 は WEP、SSL（Secure Sockets Layer）、SQL、CDPD など多数のアプリケーションで採用されてきた。RC-4 のソースコードは広く配布されてよく知られているが、暗号は RSA Security の知的財産であることに変わりない。商用製品に組み込む前に RSA Security からライセンスを取得することをお勧めする。

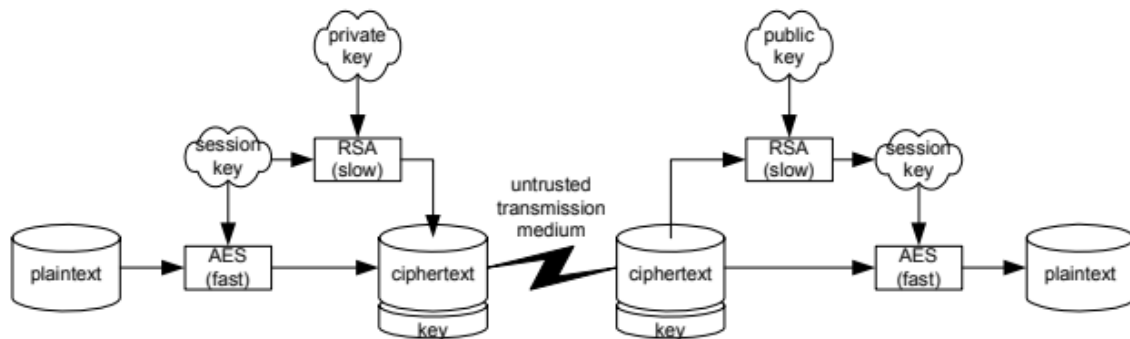


図 3: Figure 7-3 : RSA とセッション鍵の使用

図 7-3 : RSA とセッション鍵の使用。送信側は公開鍵 RSA を使ってセッション鍵を暗号化して送り、受信側は秘密鍵でセッション鍵を復号する。その後、両者は高速な AES を使ってバルクデータを暗号化・復号する。

RSA

RSA は、1977 年に Ron Rivest、Adi Shamir、Leonard Adleman が考案した公開鍵アルゴリズムだ。公開鍵アルゴリズムでは、公開鍵と秘密鍵の 2 つの異なる鍵が使われる。名前のお通り、秘密鍵は秘密にしなければならないが、公開鍵は自由に配布できる。RSA の数学的背景はコラム「RSA アルゴリズム」で簡単に説明する。Xbox のコンテキストで RSA がどう使われるかを把握するために、RSA の数学的詳細を理解する必要はない。

現在、RSA へのブルートフォース攻撃は約 1000 ビット以上の鍵長では実行不可能だと考えられている。また、RSA を暗号システムに組み込む際に不注意であってはならない。RSA を使ったプロトコルへの攻撃がいくつかあり、例えば秘密鍵の保有者に巧みに細工されたメッセージへの署名をさせ、その署名から秘密鍵を導出するといった手法がある。

RSA を使ったメッセージの暗号化は、RSA をメッセージに対して呼び出すだけでシンプルに実現できる。ただし、RSA 暗号化が処理できるメッセージブロックは短すぎ、暗号化プロセスも多くのメッセージに対して実用的な速度を出せない。そのため、RSA は通常、AES などの高速な共通鍵暗号のシングルユースランダム鍵（セッション鍵）を暗号化するために使われ、メッセージのバルク部分はその共通鍵暗号で暗号化される。図 7-3 にこのプロセスを示す。

RSA は暗号化に加えて、デジタル署名も可能にする。デジタル署名は、安全でない通信路でメッセージを交換する当事者間で、メッセージが偽造されておらず改ざんもされていないことを保証するものだ。メッセージ自体を暗号化する必要はない。典型的なデジタル署名プロトコルは次のように機能する。送信者は送信するメッセージのハッシュを計算する。このハッシュを送信者の秘密鍵で暗号化し、メッセージの平文と一緒に送る。受信者は送信者の公開鍵を使って暗号化されたメッセージハッシュを復号し、受信したメッセージのローカルで計算したハッシュと比較する。復号したハッシュとローカルで計算したハッシュが一致すれば、受信者はメッセージが本物で改ざんされていないと結論づけられる。図 7-4 にこのプロセスを示す。

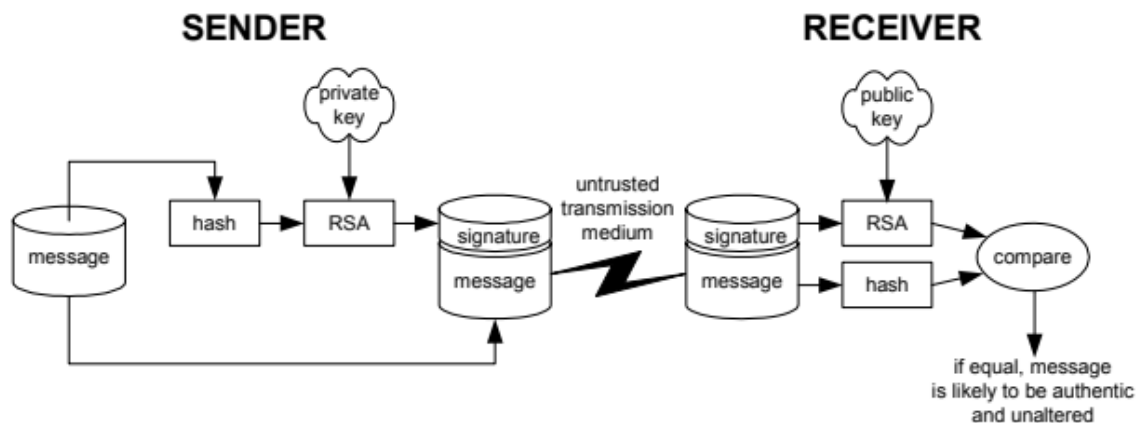


図 4: Figure 7-4 : RSA を使ったデジタル署名

図 7-4 :RSA を使ったデジタル署名の実装。送信者は秘密鍵でメッセージのハッシュを暗号化して署名を生成し、受信者は公開鍵で復号してハッシュを比較することでメッセージの真正性と完全性を検証する。

📦 コラム：RSA アルゴリズム

RSA アルゴリズムは 1983 年にマサチューセッツ工科大学 (MIT) が特許を取得し、RSA Data Security 社に独占ライセンスされた。RSA アルゴリズムの特許は 2000 年 9 月に期限切れとなり、今日ではあらゆるアプリケーションで自由に使用できる。インターネット上には RSA を使った優れたチュートリアルや教育的な例が多数ある。「RSA アルゴリズム」で検索すればすぐ見つかるだろう。

RSA アルゴリズムは次のとおりだ (<http://world.std.com/~franl/crypto/rsa-guts.html> より引用・改変)。

1. 2つの大きな（数千ビット長の）素数「P」と「Q」を見つける。
2. $E > 1$ 、 $E < PQ$ 、かつEが $(P-1)(Q-1)$ と互いに素となるように「E」を選ぶ。Eは素数である必要はないが、奇数でなければならない。EとPQのペアが公開鍵となる。
3. $(DE - 1)$ が $(P-1)(Q-1)$ で割り切れるように「D」を求める。 $D = (X(P-1)(Q-1) + 1)/E$

が整数になるような整数Xを見つけることで実現できる。Dが秘密鍵となる。

4. 平文「T」は次の関数で暗号化する： $C = (TE) \bmod PQ$
5. 暗号文「C」は次の関数で復号する： $T = (CD) \bmod PQ$

ただし、 $T < PQ$ である必要がある。PQより大きなメッセージは小さなメッセージの列に分割する必要があり、辞書攻撃などを防ぐため、非常に短いメッセージには慎重に選ばれた値でパディングする必要がある。

このプロトコルが複雑に聞こえるなら、実際にそのとおりだ。うまくいかなくなる箇所はたくさんある。受信者が送信者の公開鍵の偽のコピーを持っている可能性もある。送信者の秘密鍵が漏洩しているかもしれない。ハッシュに弱点があるかもしれない。敵対的な環境でデジタル署名を使うには、システム設計のあらゆるレベルで細心の注意が必要だ。

Xboxでは、デジタル署名はコンソール向けプログラムの配布と販売を管理するために使われている。Microsoftは実質的に、メッセージの送信者（ゲームメーカー）と受信者（Xboxコンソール）の両方を管理下に置いている。受信者——Xboxコンソール——はMicrosoftがデジタル署名したプログラムだけを実行するようプログラムされている。理想的には、Microsoftがコンソール上で実行できるプログラムを最終的に決定し、ハッカーはウイルス、トロイの木馬、バックドアを挿入するためにゲームを改ざんできないことになる。セーブデータも暗号化で封印されており、原則として実行ファイルにパッチを当てたりキャラクターのステータスを書き換えたりしてゲームでズルをすることは不可能だ。

明らかに、Xboxコンソールのハッキングにおいて重要な問題はこのデジタル署名システムの実装だ。XboxはSHA-1ハッシュと2048ビットRSA鍵を組み合わせており、ブルートフォース攻撃が成功する確率は極めて低い。もちろん、試みなければ確率はゼロだが、分が悪いことに変わりはない（コラム「とてつもなく難しい問題」参照）。宝くじで当てる方がまだましだ。これは偶然ではない——秘密鍵が発見されれば、ゲームコピーが簡単になり、デベロッパーはMicrosoftへのロイヤリティを払わずに済む（法的には義務があるかもしれないが、技術的な障壁はなくなる）。この鍵はMicrosoftにとっておそらく数十億ドルの価値を持つため、単一の人間が鍵の全体を知っているとは考えにくい——「ゴムホース（拷問）」や「グリーンペーパー（贈賄）」による暗号解析は人間には非常に効果的だからだ。（極めて貴重な秘密を守ろうとしているなら、リアルな「ブルートフォース」の可能性も軽視しないこと！）BBNのSignAssure™証明書認証管理システムのような製品は、高価値な鍵の物理的なセキュリティを確保し、マシンを起動するために複数の信頼されたユーザーの承認を必要とする秘密分散方式を実装している。

前述のとおり、RSAに対するいくつかの既知の有効な攻撃があるが、それらがすべてXboxのシナリオに適用できるわけではない——ユーザーグループへの依存や選択暗号

文が必要なものもある。また、弱点のリストは広く知られており、ほとんどのデジタル署名実装にはこれらの攻撃に対する適切な対策が施されている。

セキュリティ全体像

効果的なセキュリティシステムには、強力な暗号とハッシュに加えて、優れた鍵管理、強固なプロトコル、そして Xbox の場合には物理的なセキュリティが必要だ。

鍵管理は、セキュリティアーキテクトが直面する最も難しいシステム実装タスクの一つだ。結局のところ、復号鍵はユーザーの手に渡らなければならない。最低限の訓練しか受けていない一般ユーザーが誤って鍵情報を漏洩させないように、ユーザーインターフェースを設計する必要がある。暗号が強力になるにつれ、攻撃の最も簡単な経路はますますユーザー自身になっていく。監視カメラによる盗み見、ソーシャルエンジニアリング、あるいはパスワード入力時のキーボードの音のパターン分析さえも、暗号解析よりも単位労力あたりのパズル解情報量が多いかもしれない。公開鍵暗号は鍵配布の問題を部分的に解決するが、中間者攻撃の可能性を排除するために公開鍵のフィンガープリントは対面で確認すべきだ。また、公開鍵暗号は、クライアントマシンへの物理的なアクセス権を持つ者が復号済み出力を盗み聞きするのを防げない。

また、**プロトコル攻撃**は、鍵やデータの操作方法、または強力な暗号の使い方における弱点を突く。RC-4 に対する WEP 攻撃や、Mike Bond と Ross Anderson による IBM 4758 暗号プロセッサへの攻撃は、どちらもプロトコル攻撃の例だ。プロトコル攻撃の危険信号は、後方互換性対策を実装しているシステムや、主な職責が暗号セキュリティではないエンジニアによって実装されたシステムだ。

最後に、Xbox のような信頼できるクライアントを確立することを目的とするシステムでは、**バックドアとバッファオーバーラン攻撃**もマシンの信頼状態に対する有効な攻撃となる。広く使われている商用プロセッサは、命令ストリームやデータタグ内に実行権限を埋め込んでいない。プロセッサは、トランジェントなハードウェア障害によるものであれ、悪意を持って配置されたコードによるものであれ、ジャンプするよう命令された任意のコードを盲目的に実行する。マシン状態の定期的なハッシュによってこの欠陥を補えるが、その状態チェックさえ偽装できる。

本章の冒頭で述べたように、クライアントの信頼状態を確立するには、「信頼の種」を担う耐タンパーハードウェアも必要だ。物理的なセキュリティの強度は、一度セキュリティを破るのを経済的に引き合わないものにするとともに、一つのコンソールのセキュリティが破られても残りのコンソールへの簡単な攻撃につながらないよう強固である必要がある。物理的なセキュリティ設計のトレードオフと、Microsoft がその点でどのような判断を下したかは次章で詳しく述べる。

本章の教訓は、セキュリティには優れたシステム設計が必要だということだ。暗号はブルートフォース攻撃を事実上無意味にするほど強力になったが、システム自体は複雑化している。この複雑さはプロトコル攻撃やバックドア攻撃の成功可能性を高める一方で、古典的な盗み聞き・ゴムホース（拷問）・ユーザーエラー攻撃からユーザーを守る助けにはほとんどならない。

注

1. <http://interviews.slashdot.org/article.pl?sid=03/02/04/2233250&mode=nocomment&tid=103&tid=123&tid=172>
 2. Merriam-Webster OnLine Dictionary (<http://www.webster.com>)
 3. コードは <http://vader.brad.ac.uk/tea/source.shtml#ansi> より
 4. コードは <http://www.cc.jyu.fi/~paasivir/crypt/rciv/rc4article.txt> より。1 ページに収めるため空白を若干修正。swap_byte 関数の定義は含まれていないが、名前から推測できる。
-

第 8 章 Xbox セキュリティのリバースエンジニアリング

本章では、第 6 章で最初に遭遇した Xbox セキュリティシステムの初期製品バージョンを、僕がどのように打ち破ったかを説明する。このセキュリティシステムは、フラッシュ ROM を解析してハードウェア初期化とブートイメージの復号シーケンスがフラッシュ ROM の外側に隠されていると気づいたことで発見した。第 7 章では、本章の内容を理解するうえで役立つ基本的な暗号技術の概念を紹介した。

ハードウェアから秘密情報を取り出す

第 6 章で結論づけたとおり、Xbox に隠されたブートコードは次の三つのバスのどれかを盗み聞きすることで取得できる。(1) フロントサイドバス (FSB)、(2) メインメモリバス、(3) ノースブリッジとサウスブリッジ間の接続だ。

Xbox の Pentium プロセッサが使用するフロントサイドバス (FSB) の仕様は、Intel のデベロッパーサイトにある Pentium III プロセッサのデータシートに記載されている。FSB は双方向 64 ビットのデータバスで、約 50 本のアドレス・制御信号を持ち、すべて 133 MHz で動作する。使用する信号規格は AGTL+と呼ばれるものだ。信号数が多く物理的な接続も難しいため、このバスの盗み聞きは費用と労力のかかる作業だ。現実的な手法は二つある。(a) 数千ドルもする特殊なエミュレータ用ブレイクアウトソケットにプロセッサを載せる方法、または (b) Xbox マザーボードの FSB 配線をリバースエンジニアリングし、100 本近い信号それぞれに短いプローブ線をはんだ付けする方法だ。いずれの場合も AGTL+信号規格に対応したロジックアナライザが必要になる。これらの条件が重なったため、盗み聞きの出発点として別のバスを探すことにした。

次の候補はメインメモリバスだ。128 ビットのデータバスにアドレス・制御信号を加えたもので、200 MHz のダブルデータレート (DDR) クロックで動作する。使用する信号規格は SSTL-2 だ。(詳細は Samsung Electronics ウェブサイトにある Samsung K4D263238M のデータシートを参照。) 速度は高いが、Xbox マザーボードにはメモリのスเปアフットプリントが設けられているため、FSB よりも盗み聞きしやすいかもしれない。

標準的な安価な 100 ピン TQFP（薄型四角フラットパック、100 本のガルウィング型ピンを持つ矩形チップパッケージ）アダプタをこのスเปアフットプリントにはんだ付けすれば、ロジックアナライザを接続するプローブポイントとして使える。ただし、この手法ではメインメモリに書き込まれたデータしかキャプチャできない。復号鍵は一般に読み取り専用データであり、隠しブート ROM からプロセッサのキャッシュに直接転送されてメインメモリには格納されない。プロセッサが鍵を含むキャッシュラインの処理を終えると上書きされるため、鍵はプロセッサの物理的な境界を出ることがない。

コラム：高速デジタル信号伝送についてもっと詳しく

コンピュータバス上のデータを盗み聞きしたり改ざんしたりするのは強力な技術で、対策が難しい。盗み聞きを理解するには、コンピュータ内部でデジタル情報がどのように伝送されるかの基礎知識が必要だ。

信号方式は大きく二つに分けられる——シングルエンド信号と差動信号だ。ワイヤ上でデジタル情報を伝送するには、電圧や電流といった物理量への変換が必要だ。古典的な方式では、「グランド」と呼ばれる共通の基準電位を基準に電圧を定義する。これをシングルエンド（非平衡）信号と呼ぶ。ただしこの方式は、信号の変化が伝搬時間に対して遅い場合にしか機能しない。実際には、電位の変化は電流の流れを伴う。電流は保存されるという自然の法則により、ある方向に電流が流れれば、逆方向にも必ず電流が流れなければならない。シングルエンド信号では、この逆向き電流（リターン電流）はグランドを通して戻るが、高速では信号電流と異なる経路を通ることがある。この不均衡が信号の歪みを引き起こす。

差動信号はこの問題を解決する。信号電流用と明示的なリターン電流用の 2 本のワイヤを使うことで、信号路とリターン路を平行に配線でき、電流の流れが均衡する。配線数は 2 倍になるが、より安定した信号伝送が実現できる。

電圧を論理値に対応させる規格を信号規格という。古くからある TTL や 3.3V CMOS の信号規格は、トランジスタの性能が低く大きな信号振幅が必要だった時代に考案された。近年は SSTL（シリーズスタブ終端論理）、GTL（ガニング・トランシーバ・ロジック）、LVDS（低電圧差動信号）、PECL（擬似エミッタ結合論理）など多様な信号規格が広まっている。これらの高速信号規格は二つの事実に対処している。一つは、電気信号の波はデータ伝送レートに比べて遅く伝わること。もう一つは、伝送路の終端でエネルギーを消費する必要があり、消費されなければ反射が生じて入射信号に干渉することだ。

高速アプリケーションでは、ワイヤをしばしば「伝送線路」と呼ぶ。信号遷移時間（信号が「1」と「0」の間を遷移するのに要する時間）に比べて信号の伝搬が遅いという事実を強調するためだ。（速度の比較は信号の動作周波数ではなく、遷移時間に対するものだ。）よくある誤解は、クロック周波数が低ければ伝送線路の効果を無視できると考えることだ。年に 1 回しか遷移しない場合でも、その遷移が 1 ピコ秒（1 兆分の 1 秒）で完了するなら問題が生じることがある。

初心者にとっての朗報は、Xilinx などのベンダーの FPGA には広く使われているほぼすべての信号規格への対応が組み込まれていることだ。また、信号規格のドキュメントが充実してきているのも朗報だ。Xilinx の FPGA データシートには、対応するすべての信号規格について終端抵抗の位置と値が図解されている。データシートとアプリケーションノートの推奨事項

に従えば、FPGA を使って幅広い信号を盗み聞きできる。タップをできるだけ短くすることだけ忘れなければ問題ない。

3 番目の候補はノースブリッジとサウスブリッジ間の接続だ。それぞれ 1 本の制御信号と 1 本のクロック信号を持つ、8 ビット幅の単方向差動バスのペアだ。信号規格は HyperTransport で、DDR クロッキングにより 200 MHz で動作する。このバスの信号規格は、Xbox のチップセットとゆかりの深い NVIDIA の nForce に関する NVIDIA ウェブサイトの公開情報から推測した。その後、HyperTransport コンソーシアムのウェブサイトで公開されている HyperTransport 仕様と照合しながらオシロスコープで数回測定し、HyperTransport の信号規格が実際に使われているという仮定を確認した。

HyperTransport バスは Xbox のマザーボード上ですべての信号が平行かつ等間隔に配線されており、バスの高い動作速度が要因の設計判断だろう。これはバスを盗み聞きの格好のターゲットにするが、データレートが非常に高い点が難点だ。この速度のバスを盗み聞きするには、信号インテグリティを保つためにタップ配線のスタブ長に特別な注意が必要で、高価なロジックアナライザかカスタムのアナライザ回路も必要になる。

最終的に、最初に盗み聞きするバスとしてノースブリッジとサウスブリッジ間の接続を選んだ。配線数がずば抜けて少なく、はんだ付けの作業量が最小で済むからだ。ノースブリッジとサウスブリッジ間の接続には固有の信号が 10 本しかないが、FSB とメインメモリはそれぞれ約 100 本の信号がある。はんだ付け箇所が多いと時間がかかるだけでなく、はんだブリッジや配線パターンの損傷によるハードウェア故障のリスクも大幅に高まる。はんだ接続の数を最小限に抑えることで、マザーボードへの副次的なダメージリスクを減らせる。

高速バスの盗み聞き

2002 年 1 月下旬、HyperTransport の盗み聞きアプローチに取り組む決意をした。この手法には三つの大きな技術的課題があった。

- 信号インテグリティを損なわずに高速差動バスにタップを接続すること
- HyperTransport バスの 400 MB/s というデータレートに追いつけるロギングツールを探すか自作すること
- マザーボード上の差動 HyperTransport バス配線の極性とビット順序を特定すること

予算内でのバスタップ

最初の二つの課題は密接に関連している。高速バスの解析・ロギングツールには独自のインターフェースがあることが多く、Xbox のマザーボードへの接続にはカスタムアダプタが必要になる。最後の課題——ビットの極性と順序の特定——は、データロガーを接続して動作させた後、大量の後処理とデータ解析で解決できる。

HyperTransport は業界で広く受け入れられたオープン標準なので、市販のプロトコルアナライザやロギングツールが利用できる。その一例が FuturePlus の HyperTransport プロトコルアナライザだ。ただし、作業当時このプロトコルアナライザの価格は 25,000 ド

ルを超えていた。さらに、ターゲットボードをアナライザのバスインターフェースポッドに合わせて特別に設計する必要もある。

プロトコルアナライザを購入して Xbox 向けに改造する手間をかける代わりに、自前の簡易版を自作した。HyperTransport プロトコルはかなりシンプルなので、これは実現可能だった。Xbox の HyperTransport 実装は、送信用と受信用の 8 ビット単方向バスを二つ使っている。各バスにはクロックとストロブ線がある。この信号規格では、クロックの各エッジで有効なデータを送出する必要がある。新しいパケットの始まりはデータ線がアイドル状態から抜け出すことで示される。ストロブ線はコマンドパケットとデータパケットを区別する。他のバスで典型的なサイドバンド信号——アドレス、読み書き制御、チップセレクト、割り込み線など——は、HyperTransport ではインバンドコマンドパケットで処理される。つまり、バスを盗み聞きするのに必要なのは 10 本の差動信号 (20 本のワイヤ) だけだ——ハッカーにとって最高の朗報だ。

HyperTransport プロトコル自体はシンプルだが、Xbox バスに物理的に接続でき、かつ 400 MB/s の速度に追いつけるものをどう見つけるかが問題だった。この HyperTransport バスタップの構築に理想的なツールは FPGA だろう。しかし当時、高速データレートに対応できる FPGA は存在せず、さらに重要なことに、ベンダーから HyperTransport 用として認定された FPGA も存在しなかった。

理論上、Xilinx Virtex-II FPGA はこの用途に使えたはずだが、製品が発売されたばかりで非常に高価かつ入手困難だった (現在、ローエンドの Virtex-II FPGA は 100 ドル以下で購入できる)。当時手元にあった最良の FPGA は、修士論文の一環として試作したスーパーコンピュータ用ネットワークルータに組み込んだ Xilinx Virtex-E FPGA だった。そのネットワークルータボードはネットワークインターフェースに CTT (センタータップ終端) の信号規格を使っており、設定・制御・デバッグ用の Intel StrongArm プロセッサも搭載していた。

課題は、HyperTransport 信号を CTT 信号にインターフェースする方法と、本来そのスピードで動作するよう設計されていない FPGA から 400 MB/s の性能を引き出す方法を解決することに集約された。

HyperTransport の信号規格は、TIA/EIA-644 規格で定められた LVDS (低電圧差動信号) の近縁だとわかった。HyperTransport ドライバは通常、コモンモード電圧 600 mV を中心に差動スイング 600 mV の信号を生成する。一方、LVDS レシーバは差動スイングが 100 mV を超え、コモンモード電圧が 50 mV から 2.35 V の範囲にあるデータを正しく受信できる。つまり、LVDS レシーバは HyperTransport ドライバと直接互換性があるのだ。(Virtex-E は LVDS 信号への直接インターフェースをサポートしているが、手元の Virtex-E はすでに CTT 信号用にハードワイヤードされたシステムに組み込まれていたため、その機能を利用できなかった。) 自分でタップボードを設計するなら、ここで述べるハックではなく、FPGA のネイティブ LVDS 機能を使うのが最善策だ。

コラム：HyperTransport への信号注入

本章で説明する盗み聞きアプリケーションは HyperTransport のレシーバのみを必要とする。「中間者(man-in-the-middle)」攻撃のようなアプリケーションでは、HyperTransport の信号をオーバーライドして偽のビットを挿入できるデバイスが必要になる。HyperTransport は LVDS と同様に電流モードドライバを使っているため、これは実現可能だ。ドライバは生じる電圧に関係なく、一定量の電流をワイヤに流すよう設計されている。通常の状態では、ワイヤのインピーダンスがオームの法則に従って電流を電圧に変換するため、これは完璧に機能する。しかし電流は足し合わさったり打ち消し合ったりする。意図された信号を打ち消すオーバードライブ電流を加える拮抗差動ドライバを HyperTransport ラインに接続できる。このオーバードライブは、Xilinx Virtex-E や Virtex II などの FPGA が提供する柔軟でプログラマブルな I/O を使って実現できる。

このバスオーバーライドデバイスの最も単純な応用は、リセットベクタが CPU に送信される際にその送信先を変更し、Xbox の制御を奪うものだ。リセットベクタの送信先は、0xFFFF.FFF0 にある「ジャンプ」オペコードに続く 1 バイトにコード化されている。リセットベクタはリセット解除から確定的なクロック数後に送信される可能性が高いため、この攻撃のタイミング要素は、HyperTransport バスクロックでクロッキングされリセット信号に同期したタイマーだけで実現できる。このような「中間者」攻撃は、暗号学的に安全な公開鍵ブートブロック実装さえも無力化する。

また、信号を劣化させないために、LVDS レシーバは Xbox のマザーボードにできるだけ近い場所に配置する必要がある。ケーブルが長いとワイヤからエネルギーが逃げ、ノイズや反射が発生してシステムが動作しなくなるおそれがある。

HyperTransport 信号を FPGA に届ける問題の解決策は、信号変換チップを使うことだ。LVDS は LCD パネルインターフェースや通信システムのバックプレーンで広く使われているため、安価な LVDS-CMOS コンバータが多数入手できる。目的の信号規格は CTT だが、よく見ると CMOS ドライバを CTT レシーバにインターフェースすることは実は問題ない。CTT は 1.5 V で終端された 50 Ω の伝送線路に +8 mA または -8 mA を流す電流モードの信号規格だ。レシーバは基準終端電圧と伝送線路の電圧を比較する差動アンプだ。Virtex-E の CTT レシーバアンプは、受信電圧が基準電圧から 200 mV 以上スイングする限り動作するよう仕様が定められている。標準 CMOS トランスマッタは CTT 終端ラインを駆動する際、8 mA の電流をソース/シンクできる。また固定電圧で終端されたラインを駆動することも問題ない。したがって、標準的な LVDS-CMOS コンバータチップを使って、Xbox マザーボードの HyperTransport 信号を論文用に自作したボードに入力できる。選んだチップは Texas Instruments の SN65LVDS386 で、データシートは Texas Instruments のウェブサイトで見ることができる。

LVDS-CMOS コンバータチップをボードに接続する作業は、Xbox マザーボードの HyperTransport バスのクリーンなレイアウトのおかげで驚くほど簡単だ。図 8-1 は HyperTransport バス配線の写真だ。すべての線が平行に等間隔で走っているのがわかる。クロック (TX CK/TX CX* と RX CK/RX CX) やストローブ線 (TXD8/TXD8 と RXD8/RXD8*) など一部の線には極性マークまで付いている。このシンプルなレイアウトのおかげで、タッパボードの設計と製作が容易になる。

into a 50 ohm load. Also, CMOS transmitters should have no problem driving a wire terminated into a fixed voltage. Thus, a standard LVDS to CMOS converter chip can be used to take the Xbox motherboard's HyperTransport signals and feed them into the board I had previously built for my thesis. The chip I chose was the Texas Instruments SN65LVDS386, and you can find data sheets for this chip at Texas Instruments' website.

Attaching the LVDS-to-CMOS converter chip to the board is made delightfully simple by the clean layout used for the HyperTransport bus on the Xbox motherboard. Figure 8-1 is a picture of what the HyperTransport bus traces look like. Notice how all the wires run in parallel and how they are evenly spaced. Some of the wires, such as the clock (TX CK/TX CX* and RX CK/RX CX*) and the strobe line (TXD8/TXD8* and RXD8/RXD8*), are even labeled for us with polarity markings! This simple layout enables the use of an easy-to-engineer tap board.

The tap board contains just the LVDS-to-CMOS converter chip, some power conditioning circuitry, and a set of traces laid out right up to the edge of the board that are identically spaced to the HyperTransport bus on the Xbox motherboard. For identical spacing and easy alignment and mounting, I measured the dimensions of these traces using a digital caliper tool. Figure 8-2 illustrates the dimensions of the HyperTransport bus traces.

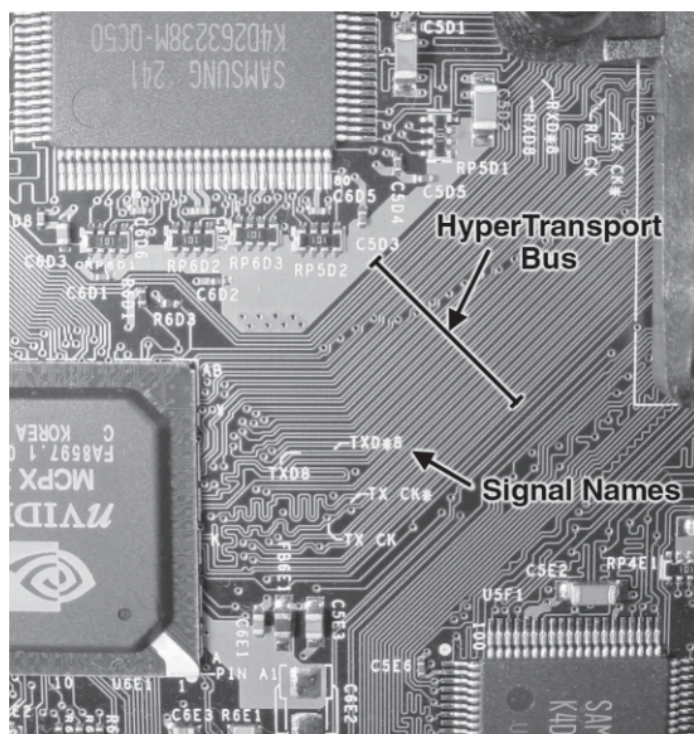


Figure 8-1: HyperTransport bus traces as laid out on an Xbox motherboard.

図 8-1: Xbox マザーボードの HyperTransport バス配線パターン。

タップボードに必要なのは、LVDS-CMOS コンバータチップ、電源調整回路、そして Xbox マザーボードの HyperTransport バスと同じ間隔でボードの端まで引き出した配線パターンだけだ。正確な間隔を実現するために、デジタルキャリパーでこれらの配線の寸法を計測した。図 8-2 に HyperTransport バス配線の寸法を示す。

The measurements were a little tricky to make. My approach was to measure the overall width of the bus and divide the width by the number of traces and spaces to get the average expected spacing and trace width. I then laid out these traces with a PCB CAD program and printed the layout on paper at a 1:1 scale. I compared the printed traces with the board traces and made a few adjustments by hand. (Note that many printers have some small amount of scaling error, so if you are trying this, calibrate yourself by printing out a few long lines of known length and measuring them. Printers can have different scaling errors along the horizontal and vertical axes, so be sure to print lines in both directions.)

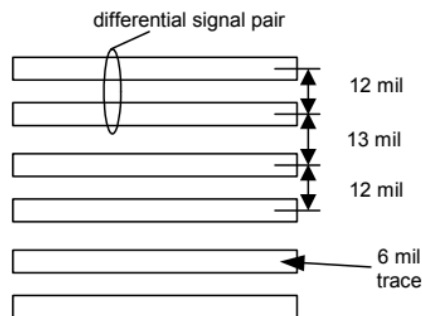


Figure 8-2: Dimensions of the HyperTransport bus traces on the Xbox motherboard. A “mil” is 1/1000th of an inch or 25.4 microns.

Designing your own boards is fairly easy with the right software. You can find out more about how to make your own boards by reading the Appendix C, “Getting Into PCB Layout.”

Once the component selection process was finished, the design and layout of the HyperTransport tap and signal conversion board took just a few more hours. A schematic of the board’s design can be seen in Figure 8-6. The board was then fabricated by an order placed via the Internet. Many board houses offer affordable, quick-turn board fabrication services that take board designs in Gerber file format via an email or ftp upload. In this case, I had two copies of the board built in five days for a price of \$33 per board (see Appendix C, “Getting Into PCB Layout,” for more information on how to build your own boards). This price only includes the price of cutting the board into a square piece. However, I needed the side of the board with the HyperTransport tap to have a special shape that facilitates board mounting without interfering with the existing components on the Xbox motherboard. I also needed the mating edge of the board to be beveled such that the board mounts at a slight angle, to simplify the task of soldering the tap board to the motherboard. I used a belt sander to manually sculpt the edge into the shape described in Figure 8-3. When sculpting, the board had to be oriented such that the belt sander’s abrasive belt made contact with the trace side of the board first to prevent the belt sander from tearing the copper traces off of the board. Be careful when using a belt sander to sculpt small boards like the tap board — a belt sander could just as easily sculpt your fingers by accident.

図 8-2: Xbox マザーボードの HyperTransport バス配線の寸法。1 ミル (mil) は 1 インチの 1/1000 (25.4 マイクロメートル)。

この計測は少しトリッキーだった。バス全体の幅を計測し、配線数とスペース数で割って平均的な間隔と配線幅を求めた。次に、PCB CAD プログラムでこれらの配線をレイアウトし、1:1 スケールで紙に印刷した。印刷した配線とボードの配線を見比べながら手で微調整した。(多くのプリンタには多少のスケーリング誤差があるため、試す場合はまず既知の長さの線を何本か印刷して計測し、誤差を確認すること。プリンタは水平方向と垂直方向で異なるスケーリング誤差を持つことがあるので、両方向で確認すること。)

適切なソフトウェアがあれば、オリジナルのボードを設計するのはそれほど難しくない。詳しくは付録 C 「PCB レイアウト入門」を参照。

部品の選定が終わると、HyperTransport タップと信号変換ボードの設計とレイアウトにはさらに数時間しかかからなかった。ボードの回路図は図 8-6 に示す。その後、インターネット経由で発注してボードを製造した。多くの基板製造業者は Gerber ファイル形式のボード設計データをメールや FTP で受け取り、手頃な価格でクイックターン製造サービスを提供している。今回は 5 日間で 1 枚あたり 33 ドルで 2 枚製造した(詳細は付録 C 「PCB レイアウト入門」参照)。この価格はボードを四角く切り出す費用のみを含む。ただし、HyperTransport タップのある側の形状を特殊にする必要があった。Xbox マザーボード上の既存の部品に干渉せずに取り付けられるよう設計するためだ。また、タップボードをマザーボードにはんだ付けする作業を容易にするために、接合側のエッジをわずかに傾斜(ベベル)させる必要もあった。図 8-3 に示す形状になるようベルトサンダーで手作業で削り出した。削り出す際は、ベルトサンダーの研磨ベルトがボードの配線面から先に当たるよう向きを調整して、銅の配線パターンが剥がれないようにした。タップボードのような小さなボードをベルトサンダーで削るときは注意が必要だ——うっかりすると指まで削れてしまう。

ベベルエッジの削り出しが完成したら、ボードにすべての部品をはんだ付けした。(付録 B 「はんだ付け技術」参照。)

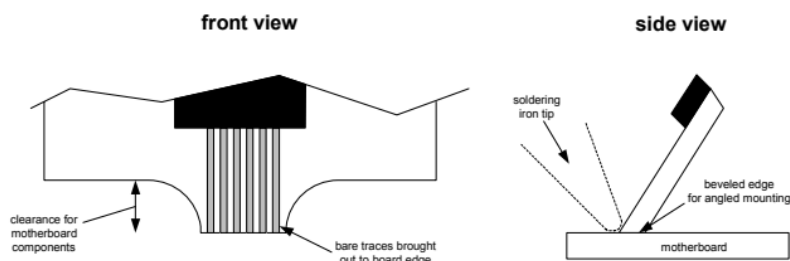


Figure 8-3: Shaping of the HyperTransport tap board edge.

After sculpting the beveled edge, all the parts were soldered onto the board. (See Appendix B, “Soldering Techniques.”)

The finished tap board now had to be attached to the Xbox motherboard. This critical step was perhaps the most difficult one. First, the Xbox motherboard was prepared by using a fine grit sandpaper to strip away the green soldermask, revealing the bright bare copper of the target traces. Then, these traces were fluxed and a thin coat of solder was applied using a hot soldering iron tip.

The procedure I used for attaching the tap board to the motherboard is shown in Figure 8-4. The prepared tap board was tacked onto the motherboard at the approximate location and angle using a thin (30 AWG) wire soldered between a trace on the tap board and the motherboard. The tack wire serves only as a temporary aid for holding

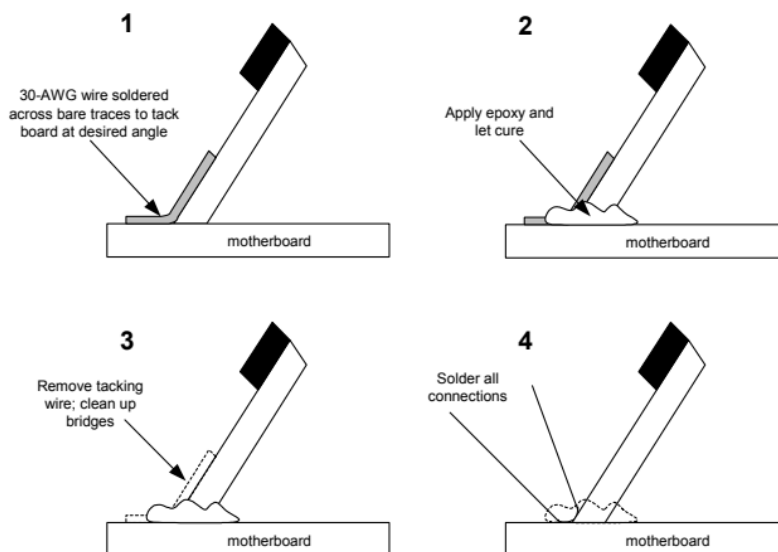


Figure 8-4: Tap board soldering procedure.

図 8-3: HyperTransport タップボードのエッジの形状。

図 8-4: タップボードのはんだ付け手順。

完成したタップボードを Xbox のマザーボードに接続する必要があった。この重要なステップがおそらく最も難しかった。まず、Xbox のマザーボードを細かい目の紙やすりではんだレジストを剥がし、ターゲットとなる配線の素銅を露出させた。その後、これらの配線にフラックスを塗り、熱したはんだごての先で薄くはんだを乗せた。

タップボードをマザーボードに接続するための手順を図 8-4 に示す。準備したタップボードを、タップボード上の配線とマザーボードの間に 30 AWG の細い線を一本はんだ付けして、大まかな位置と角度でマザーボードに仮固定した。この仮固定線は一時的な保持手段として使うだけで最後は取り外すので、複数の配線をまたいでいても問題ない。線をつけたら、銅の配線を浮かせないよう線を熱して接着を解きながら、マザーボード上のタップボードの位置を慎重に微調整した。(最適な位置合わせを確認するために顕微鏡を使った。) ボードの位置に満足したら、接合部に強力なエポキシを塗布してすべてを固定した。エポキシは硬化して堅固な接合部を形成する必要がある。(エポキシを正しく使わないとゲル状に硬化するものがあるが、接合部の機械的な強度は完全にエポキシで確保しなければならず、はんだ接合部に頼ってはいけない。) 使用したのは Miller-Stephenson エポキシフォーミュラ 907 で、タップボードを持って Xbox を持ち上げてもタップ接続が乱れないほどの強度で硬化する。

エポキシが硬化したら、仮固定線を取り除き、素銅の接合面をはんだ吸い取り線とフラックスで少し清掃した。タップボードの配線をマザーボードの素銅配線にはんだ付けする最終ステップは、表面実装部品をボードにはんだ付けするのと変わらない。付録 B で説明する標準的なはんだ付け技術がほぼそのまま使える。図 8-5 に完成したアセンブリの外観を示す。

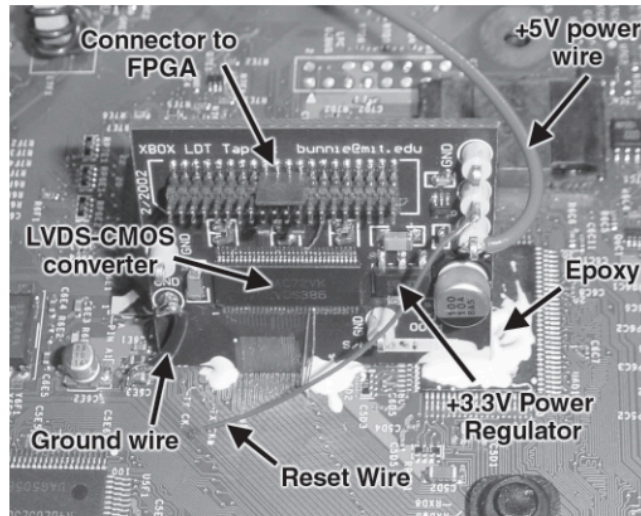


Figure 8-5: HyperTransport tap board mounted on the Xbox motherboard.

the board in place and will be removed, so it does not matter if the wire bridges multiple traces. Once the wire was attached, I carefully adjusted the position of the tap board on the motherboard, heating the wire to release its bond to avoid lifting any of the copper traces. (I used a microscope to aid in determining the optimal alignment.) Once I was satisfied with the position of the board, I applied a strong epoxy to the board joint to hold it all in place. The epoxy should cure and form a rigid, stiff joint. (Note that some epoxies when applied incorrectly cure into a gel; this is not acceptable, as the entire mechanical integrity of the joint must come from the epoxy and not the solder joints.) I used Miller-Stephenson Epoxy formula 907, and it sets with enough strength for me to lift the Xbox by the tap board and not disturb the tap connection.

Once the epoxy had cured, I removed the temporary tack wire that was used to hold the tap board in place, and cleaned the bare mated traces with a bit of solderwick and flux. The last step of soldering the tap board traces to the bare motherboard traces was now no different from soldering any surface mount component onto a board; most of the standard techniques described in Appendix B applied directly to this situation. Figure 8-5 shows what the finished assembly looks like.

Building the Data Logger

The second challenge of eavesdropping the HyperTransport bus is acquiring or building a logging device that can keep up with the 400 MB/s data rate of the bus. Considering my budget, I decided that my only option was to build a logger, as buying any tools with sufficient performance for this job was well outside of my budget.

図 8-5: Xbox マザーボードにマウントされた HyperTransport タップボード。

データロガーの構築

HyperTransport バスの盗み聞きの二つ目の課題は、バスの 400 MB/s データレートに追いつけるロギングデバイスを入手するか自作することだ。十分な性能を持つツールを購入するのは完全に予算外だったため、自作するしかなかった。

ロギングデバイスの構築には、以前自作したボードに組み込んだ Virtex-E FPGA を使うことにした。ただし、Virtex-E FPGA を使う上での問題は、データブック上の性能仕様が HyperTransport バスに追いつくには不十分な点だ。幸い、FPGA は製造マージンが保守的で、設定可能な配線ファブリック内の信号伝搬遅延が性能の主な制約要因になっているため、オーバークロックが良好に機能する。そのため、性能を制限している主要なパスをいくつか手動で特定し、ソフトウェアによるディレイラインとクロックの選択的な反転で補正できる。性能上最も重要なブロックは遅延を最適化するために手動配置し、コンパイラと自動配置配線ツールに非重要な部分を任せた。図 8-7 に HyperTransport バスのデータをキャプチャするために使った全体設計を示す。

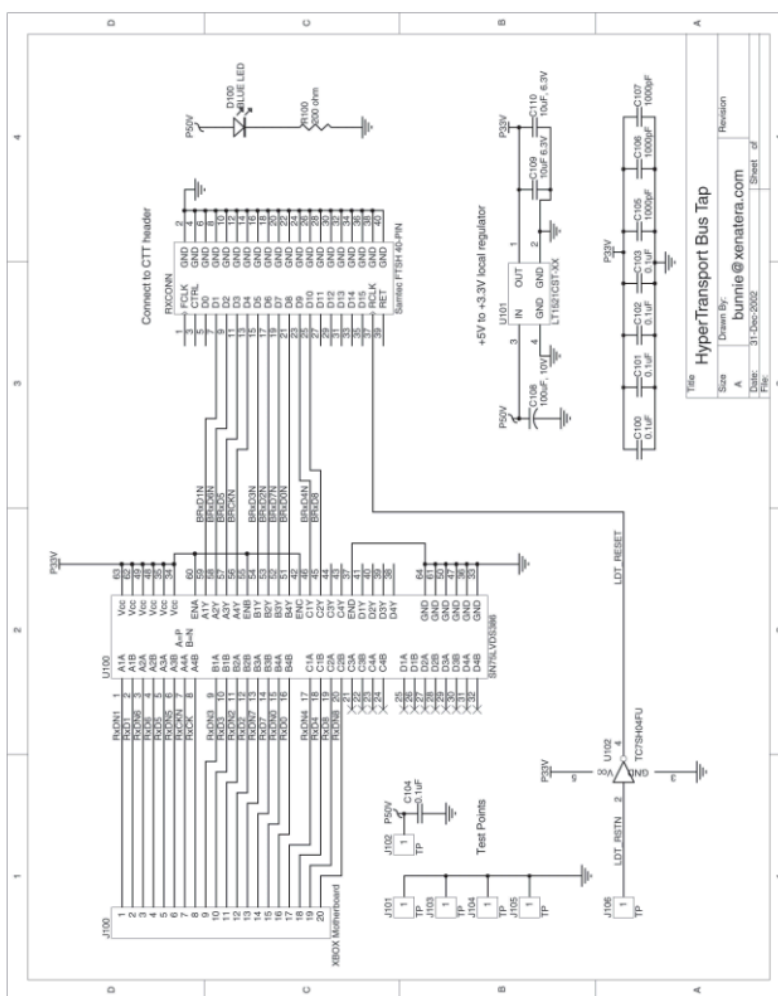


Figure 8-6: Schematic of the HyperTransport tap board.

In building the logging device, I had settled on using a Virtex-E FPGA that was integrated into a board that I had previously built. However, the one problem with using the Virtex-E FPGA is that the performance of the FPGA (as specified in the databook) is insufficient to keep up with the HyperTransport bus. Fortunately, FPGAs overclock well because their manufacturing margin is very conservative, and because FPGA performance is largely limited by signal propagation delays in the configurable wiring fabric. As a result, some key performance-limiting paths can be manually identified and compensated using soft delay lines and selectively inverted clocks. The most performance sensitive blocks can be hand-placed to optimize the delays, while the compiler and

図 8-6: HyperTransport タップボードの回路図。

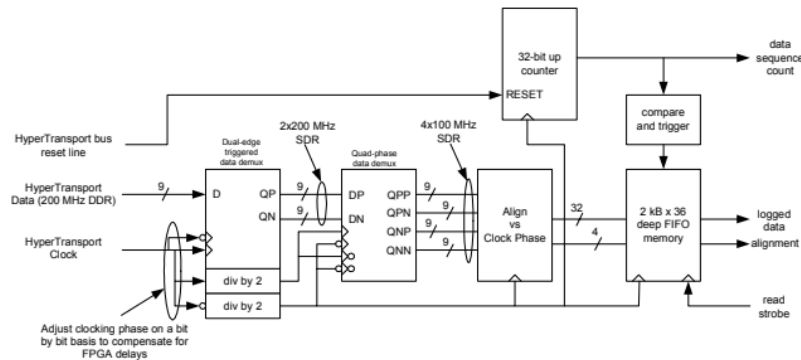


Figure 8-7: Block diagram of the data logger built in the Xilinx Virtex-E FPGA.

automated place-and-route tool handle the non-critical parts of the circuit. Figure 8-7 shows the overall design that was used to capture the data on the HyperTransport bus.

The design is fairly simple in concept: take the high speed data off of the HyperTransport bus and clock it into four phases of a quarter speed clock, creating a data stream that is four times slower but four times wider. This confines all hand-placing and tweaking to just the first few input flip flops. Next, realign the data using a set of delays and rotators, and store the data one piece at a time inside a first in, first out (FIFO) memory. The signal that triggers the start of FIFO capture is generated by a timer-comparator that starts counting up from first reset. Long windows of data can be captured by concatenating the results of multiple runs, each with the capture trigger point delayed from the previous. A later optimization applied to the trigger circuit is a “do not store zeros” (DNSZ) function. In the DNSZ mode, data consisting of all 0’s is not stored in the FIFO. This is helpful in culling out all of the idle data on the HyperTransport bus. The resulting data traces are a time-stamped series of 32-bit words.

The most difficult part of the FPGA data logger design was calibrating the delays on the input paths. Delay calibration was accomplished by using an oscilloscope to probe a small window of data on the HyperTransport bus. Wire delays and byte-wide rotations were tweaked until the probed data matched the log data. This process was aided by the fact that during idle times, a common sequence of commands was repeated on the bus every few hundred microseconds, which served as the calibration reference.

Determining the Bus Order and Polarity

The final challenge after logging the data is figuring out the order of the signals on the HyperTransport bus and their polarities. Note that while the two most important signals of the HyperTransport bus on the Xbox motherboard are labeled for us, the remaining eight data lines have ambiguous polarity and bit ordering.

図 8-7: Xilinx Virtex-E FPGA に実装したデータロガーのブロック図。

設計の概念はシンプルだ。HyperTransport バスから高速データを取り込み、1/4 速度クロックの 4 位相でクロッキングして、4 倍遅いが 4 倍広いデータストリームを生成する。これにより、手動配置と微調整を最初の数段の入力フリップフロップだけに限定できる。次に、遅延器とローテータのセットでデータを再整列し、先入れ先出し (FIFO) メモリに一度に一つずつ格納する。FIFO キャプチャ開始のトリガー信号は、最初のリセットからカウントアップを始めるタイマーコンパレータで生成する。複数のランの結果を連結することで長いウィンドウのデータをキャプチャできる (各ランでキャプチャトリガー点を前のランからずらす)。後でトリガー回路に追加した最適化は「ゼロを格納しない」(DNSZ: Do Not Store Zeros) 機能だ。DNSZ モードでは、すべてゼロのデータは FIFO に格納しない。HyperTransport バスのアイドルデータを除外するのに有効だ。出力されるデータトレースは、タイムスタンプ付きの 32 ビットワードのシリーズだ。

FPGA データロガー設計の最も難しい部分は、入力バスの遅延のキャリブレーションだった。キャリブレーションは、オシロスコープを使って HyperTransport バス上の小さなウィンドウのデータをプローブすることで行った。プローブデータとログデータが一致するまで、配線遅延とバイト単位の回転を調整した。アイドル時に数百マイクロ秒ごとにバス上で共通のコマンドシーケンスが繰り返されており、これがキャリブレーションの基準として役立った。

バスの順序と極性の特定

データのロギングが完了した後の最後の課題は、HyperTransport バスの信号の順序と極性を解明することだ。Xbox マザーボード上の HyperTransport バスの最重要な 2 本の信号はラベルが付いているが、残りの 8 本のデータ線は極性とビット順序が不明だ。

8 本のデータ信号の正しい極性は、アイドル状態のバスデータのビットパターンを観察することで特定した。HyperTransport バスはほとんどの時間アイドル状態なので、これは難しくない。アイドルパターンがすべて 0 であるべき場合、1 として現れるビット位置は極性が反転している。これは FPGA 内の対応する配線に反転項を挿入することでハードウェアで修正した。

しかし、正しいビット順序の特定はずっと難しい。HyperTransport バスを通るデータの多くはフラッシュ ROM から来るという仮定のもと、バイトごとに 1 のカウントを行った。理論は、バスの順序は純粋な並べ替え (permutation) であり、バイト中の 1 のビット数はフラッシュ ROM のデータとロガーでキャプチャしたデータ間で保存されるというものだ。1 のカウントのパターンを照合して、フラッシュ ROM とログデータ間の対応する候補領域を特定した。幸い、HyperTransport バスを最初に通るデータはフラッシュメモリの底部近くにあるチップセット固有の初期化データなので、正しく一致するパターンを見つけるのにそれほど時間はかからなかった。各 ROM とロガーのバイトのセットを表に並べ、短い C プログラムの助けを借りて、すべての行の値が一致する順序が見つかるまでビットの列を入れ替えた。

キャプチャデータの解説

有効なデータトレースが取れたところで、残る問題はその意味を解説することだ。解説の前に、これまで収集したデータについてわかっていることを整理しておこう。

- **時間的相関**：ログデータは大局的に見て、想定される初期化イベントのシーケンス——ジャムテーブルの初期化、続いて復号ステップ、RAM からの実行——と強い時間相関を持つはずだ。各イベントに対応するログトレースの領域は、大きなアクティビティのバーストが発生してその後に無音の領域が続く時点を観察するだけで特定できる。
- **トランザクション長**：Pentium プロセッサにはデータキャッシュと命令キャッシュの両方があるため、HyperTransport バス上でのフラッシュ ROM や隠しブート ROM へのすべてのフェッチは、偶数長のトラフィックバーストとして現れるはずだ。
- **確定的な順序**：収集したデータはタイムスタンプが付いており時系列的に正しいため、リセットベクタで最初にフェッチされる命令をデータログで特定できれば、残りの命令の位置と構造を推測できる。

最初、HyperTransport バスを通るデータの大局的な構成を確認することを怠ったため、いくつかの問題が生じた。図 8-7 のロギングマシンの簡略ブロック図では、HyperTransport バスがリセットされるたびにログ FIFO もリセットされる。これは良いアイデアに思えたが、もともと電源投入時に HyperTransport バスは一度しかリセットされないと誤って思い込んでいた。実際には、ジャムテーブルの初期化ステップの後に HyperTransport バスが二度目のリセットを受ける。そのため、最初にトレースを見始めたとき、暗号化データとわずかなコードの断片しか見えず、ブートベクタと論理的に対応付けられるものが何もなかった。

これがどれほどがっかりすることか！ 一歩引いて、オシロスコープのタイムスケールをミリ秒/分割に設定して HyperTransport バスのイベントを観察した。より早い段階のリセットパルスがあることに気づき、最初のパルスだけをキャッチするようトリガー機構を調整したところ、ブート命令が簡単に特定できた。秘密 ROM の 0xFFFF.FFF0 にある 16 バイトは、フラッシュ ROM の同じ 16 バイトと偶然一致していた。そこから、各命令ブロックをメモリ上の正しい位置に配置できるよう、地道なトレースと逆アセンブルを大量に行いながらプログラムカウンタの現在値を追跡した。キャッシュラインのフェッチは毎回連続する 16 または 32 バイトのメモリで構成されており、独特のデータロガータイムスタンプパターンを生成してリバースエンジニアリングの助けになった。

キャッシュラインを探してトレースを数時間精査した後、ディスアセンブラに入力するのに十分なコードを収集した。（使用したディスアセンブラの詳細は後述のコラムを参照。）

コラム：ソフトウェア解析ツールについて

ハッキング経験を積む中で、必ずどこかでアセンブリ言語コードを逆アセンブルする必要がある。2002 年 1 月、Xbox のセキュリティをリバースエンジニアリングしていた時、仲間のソフトウェアハッカーたちからこの作業に最適なツールを教えてもらった。それ

が Ilfak Guilfanov による IDA Pro で、DataRescue Corporation が販売している (<http://www.datarescue.com/idabase/>)。IDA Pro は x86 コードだけでなく、実に多様な組み込みプロセッサのコードも逆アセンブルできる。IDA Pro の出力品質も非常に高く、コードセグメントは自動的に注釈付けされ、読みやすく整理される。IDA Pro には多数の便利で楽しいツールも搭載されている。僕のお気に入り、コードライブラリのシグネチャを関数呼び出しに自動的にパターンマッチングする機能と、キーを押すだけでジャンプをたどる機能だ。

コード解析で非常に役立ったもう一つのツールは HackMan だ。HackMan は TechnoLogismiki Corporation のフリーウェアだ (<http://www.technologismiki.com/hackman/>)。名目上は「ヘックスエディタ」、つまりバイナリデータを直接操作できるファイルエディタだが、単純な編集をはるかに超えた多くのユニークな機能を持つ。たとえば、HackMan には逆アセンブラが内蔵されている。IDA Pro ほど強力ではないが、ヘックスエディタとインタラクティブに連携する。データログをトレースしながら候補のキャッシュラインを有効なコードかどうか素早くテストし、秘密 ROM の最終的なバイナリイメージを組み立てる際に役立った。

データの後処理とオンラインのハッカー仲間の大いなる助けを借りて、使われている暗号が RC-4/128 だと特定した。RC-4 は対称暗号なので、鍵は Xbox のどこかに保存されているはずだ。しかしデータストリームで鍵を特定しようとしていたが、なかなかうまくいかなかった。鍵は当時は明確な位置に対応付けられないコードの断片と共有されたキャッシュラインフェッチにまたがっているようだった。

夜が更け、16 進数を見つめることに疲れてきた頃、絶対に成功しないはずのことを試してみることにした。データログ内のスライディングウィンドウから導出した鍵を使ってフラッシュ ROM のターゲットイメージを復号する RC-4 復号プログラムを改造した。これはかなりブルートフォースな手法で、データストリーム全体を検索するには数万回の復号（ログの各バイトに対して 1 回ずつ）が必要だ。RC-4 復号の出力をヒストグラムルーチンに入力することで処理を自動化した。鍵が一致しない場合、出力は統計的に「ホワイト（一様）」になるはずだ——一致しない鍵ではすべての値が概ね同じ確率で現れるはずだ。しかし鍵が正しいなら、ヒストグラムに偏りが現れ、他の値よりも突出して多い値が出てくるはずだ。

最終的に、午前 5 時頃にこのブルートフォース検索を行うプログラム「trykeys」を完成させた。目をしょぼつかせ疲れ果てながらも、今夜の作業を終える前にテスト実行することにした。プログラムが候補データストリームを処理し続ける出力を見て、僕がどれほど呆然としたか想像してほしい。

```
$ ./trykeys.exe ms4.bin binout.full
.....
.....found possible key combo: avg 96, min 5,
offset 8745.....
.....
```

フラッシュ ROM イメージは ms4.bin、バイナリデータログトレースは binout.full という名前だ。trykeys プログラムは、オフセット 8745 から始まるデータをテスト鍵として使った ROM イメージの復号で、統計的に異なるヒストグラム（平均値 96、最小バケッ

ト高さ5)を識別した。データストリームから候補鍵を取り出し、候補鍵を使って復号出力を解析した。出力は本物の有効なコードのように見えた。隠しブートセクタの鍵を見つけた——サウスブリッジチップに保存されていた！数日後、睡眠を取って学業に追いつき、データストリームの適切な解析を完了させ、秘密ブートセクタ全体のイメージを組み合わせた。

秘密ブートコードのRC-4鍵を手に入れたことで、当時のすべてのXboxが受け入れるフラッシュROMイメージを生成できるようになった。これはXboxのトラスト（信頼）メカニズム全体が、マザーボード上のROMをオーバーライドするか交換するだけで侵害できることを意味する。これはMicrosoftが製造中のテストと診断目的でフラッシュROMをオーバーライドするために提供しているテスト構造を使って実現できる。Xboxは数秒に1台のペースで生産ラインから出荷しなければならなかったため、MicrosoftはフラッシュROMのオーバーライドを可能にするクイック接続テストポイントのセットを設計していた。

コラム：ハッキングの法的課題

振り返ってみると、Xboxのハッキングで本当に大変だったのは技術的な部分よりも、社会的・法的な部分だった。サウスブリッジチップから秘密鍵を取得した後、MIT人工知能研究所の研究指導教員であるTom Knight教授と面会して結果を議論した。僕の研究はDMCA（デジタルミレニアム著作権法）に違反する可能性があるとして指摘され、発表前にMITの法務部に相談した。MITの法務部は最終的に、DMCAによってリスクが高すぎるため、コンピュータアーキテクチャ研究の一環としてMITで行った研究にもかかわらず、個人として発表しなければならないと回答してきた。弁護士費用を払えず、研究を発表できないかもしれないと絶望したが、その時Hal Abelson教授が電子フロンティア財団（EFF）につないでくれた。その結果、EFFとボストンカレッジのLee TienとJoe Liuが僕の研究発表を支援してくれることになった。

その後、数ヶ月にわたる審議と交渉が続いた。二つの戦線での戦いだった——Microsoftを宥めながら、MITに研究を受け入れさせなければならなかった。4ヶ月後、Microsoftによる好意的な研究レビューと、研究室の同僚と教授たちの圧倒的な支持を受けて、MITが折れた。MITは独立した個人としてではなく、MIT学生として研究を発表できると決定した。5ヶ月の法的な膠着状態の結果、AI研究所のテクニカルメモランダムが作成され、続いて2002年8月の暗号化ハードウェアと組み込みシステムに関する国際会議（CHES：Cryptographic Hardware in Embedded Systems）での学術発表につながった。

この話には一応のハッピーエンドがあるが、指導教員、研究室、そしてEFFの優れた弁護士たちの支援がなければ、まったく違う結末になっていた可能性がある。DMCAは不正なハッカーと正当な研究者の間に曖昧な線を引く。MITの支持がなければ、DMCAの研究免除条項を満たせなかったかもしれず、研究が発表されることなく終わったか、発表されたとしてもMicrosoftに異議を唱えられていたかもしれない。言論の自由はすべての人に適用される——名門学術機関の象牙の塔に座る幸運な人たちだけのものではない。Xboxに取り組み、優れた成果を上げていた人は無数にいたが、彼らの声はDMCAの幕の裏に永遠に封じ込められている。

代替 ROM イメージは、ネイティブの Xbox CPU を使って製造テストプログラムを実行するのに役立つ。Xbox の LPC (Low Pin Count) インターフェース実装の物理的な構造により、ユーザーも Microsoft の製造委託業者も、はんだ付けなしで適切に設計されたフラッシュ ROM オーバーライドデバイスを取り付けられる。

明らかに、Xbox のトラストメカニズムをオーバーライドする能力には厄介な法的含意がある。僕の意図は主に好奇心を満たすことと、次にフェアユースの権利のもとで自分のコードを Xbox 上で実行することだったが、他の人はゲームをコピーしたり、Microsoft の著作権で保護されたカーネルコードを改変・再配布したいと思うかもしれない。暗号はその応用を選ばないので、RC-4 鍵の抽出はすべての用途に等しく使えてしまう。そのため、法的な問題を整理するために電子フロンティア財団 (EFF) に連絡した。法的プロセスは遅くて重たいものだ。鍵を取得したのは 2002 年 2 月だったが、適切な学術フォーラムで研究成果を発表する許可が下りたのは 6 月近くになってからだった。

たった 128 ビットのためにこれほど大騒ぎになるとは思ってもみなかった。1998 年のデジタルミレニアム著作権法 (DMCA: Digital Millennium Copyright Act) は、ハードウェアハッキングの景色を永遠に変えた。リバースエンジニアリングはかつて保護された行為であり、健全で競争的な市場を構成する一部と見なされていた。今では、フェアユースの権利を行使するために自分の家の中で暗号セキュリティシステムをいじって回避するだけで、数千ドルの罰金と訴訟に直面する可能性がある。第 12 章「注意せよハッカーよ」を読んで、自分の法的権利と責任を理解することを強くお勧めする。

コラム：隠蔽によるセキュリティ

Xbox セキュリティの初版で Microsoft が使った技術は、隠蔽によるセキュリティ (security through obscurity) の典型例だ。強力な暗号である RC-4/128 を使って ROM イメージを暗号化し、人々が ROM の内容を解析したり独自の ROM を作成したりするのを防ごうとした。しかし、RC-4/128 は対称暗号なので、Xbox は ROM イメージを復号する鍵——同時に暗号化鍵としても使える——を持たなければならない。この復号・暗号化鍵が、秘密ブート ROM の内部に埋め込まれた重要な保護情報だ。この鍵を隠すことが隠蔽によるセキュリティだ——いったん鍵が見つければ暗号は無意味になり、すべてのセキュリティが失われる。

真のセキュリティとは、ユーザーが Xbox のすべてのパーツにアクセスできても、自分の有効なフラッシュ ROM イメージを暗号化できない状態を指す。つまり、Xbox 外部に秘密情報を保管する必要がある。公開鍵暗号はまさにこのシナリオのために考案された。Microsoft が Xbox のブートコードを公開鍵暗号で暗号化または署名していたなら、安全なブート ROM の内容をすべて知っていても役に立たない。主要な秘密情報である Microsoft の秘密鍵は、ワシントン州レッドモンドのどこかの金庫室に安全に保管されており、手の届かない場所にあるからだ。

ただし、明るい面もある。次章では仲間たちの発見を紹介する。その多くは Xbox 初期化シーケンスのバックドアの発見だ。これらのバックドアを使えば、Microsoft の著作権で保護された著作物へのアクセスを有効にせず、ゲームのコピーも有効にすることなく、Xbox で自分のコードを実行できる。次章ではまた、英国の Andy Green によってわずか数日でクラックされた Xbox セキュリティバージョン 1.1 も紹介する。

第9章 裏口から忍び込む

Xbox に対して実行可能な攻撃の全容は、本書に収めるには余りに多すぎる。Xbox は PC アーキテクチャをベースとしており、そのアーキテクチャは複雑で進化を重ねてきたが、もともとセキュリティを意識せずに設計されたものだ。スマートカードのハッカーが悪用する古典的なハードウェアセキュリティホール——電源変調、サイドバンド攻撃、クロックグリッチなど——の多くは、僕の知る限り Xbox ではまだ手がつけられていない。(これらのセキュリティ上の弱点については、Lecture Notes in Computer Science シリーズ (Springer-Verlag) 所収の Cryptographic Hardware in Embedded Systems (CHES) 会議の論文集でより詳しく知ることができる。)

残念ながら、ゲームコンソールやセキュア PC のメーカーはハードウェアセキュリティの弱点をさほど問題視していない。ハードウェア攻撃は「一般消費者には難しすぎて実行できない」ため、脅威が少ないと見なされているからだ。攻撃手法を研究するには正しいツールを持つ熟練のハッカーが必要だというのは事実だが、いったん攻撃手法を実装すること自体は非常に安価で簡単にできる。ある整備士の話思い出す。重要な機械の修理に呼ばれた整備士が1時間かけて状況を観察し、まさに正しい場所を叩くだけで機械を直した。1,000 ドルの請求書を受け取った機械の所有者たちが、なぜ叩くだけでそれほどの費用がかかるのかと尋ねると、整備士はこう答えた——「叩くこと自体は10セントです。どこを叩くかを知っているのが999.90ドルなんです」。この話の教訓は、具体的な手順が示されれば誰でもその「叩く」動作を実行できる、ということだ。

セキュリティ攻撃も多くは同じだ——解明するのは難しくても、共有して実装するのは簡単だ。セキュアなハードウェアのメーカーは、ハッカーの侵入に対して事後対応型のポリシーを採用することにも警戒すべきだ。多くのハッカーは秘密裏に活動し、ベンダーが適切な対策を講じられないよう手法と成果を非公開にしている。また既知の攻撃とバックドアのライブラリを保持し、一度に一つずつしか公開しない。そのため事後対応型のセキュリティポリシーを持つベンダーは、常に後手に回ることになる。

コラム：用語について

ハッキングコミュニティはしばしば重要な概念に独自の用語を作り出す。その用語はコミュニティごと、また業界標準用語とも異なることがある。以下は Xbox-Linux コミュニティで受け入れられている用語の一覧だ。本書の用語との相違点はそのつど注記する。

- **X-code (エックスコード)** : ジャムテーブルのオペコードのこと。サウスブリッジ (MCPX) 内のシークレットブート ROM が Xbox のハードウェアを初期化するために使うオペコード群だ。
- **2BL (ツービーエル)** : セカンドブートローダ (Second Boot Loader)。シークレットブート ROM によって復号されるコードのことだ。カーネルイメージの復号と解凍が主な役割であることから「セカンドブートローダ」と呼ばれる。
- **FBL (フラッシュブートローダ)** : バージョン 1.1 セキュリティでは、シークレットブート ROM と 2BL の間に位置する中間ブートローダ。FBL は、シークレットブート ROM 内に

ハードコードされた値に対して軽量ハッシュで検証される。そのため FBL は MCPX シリコンを変更しない限り書き換えられない。フラッシュ ROM の重要な領域すべてのデジタル署名を検証する役割を担う。

- **カーネル:** Xbox の OS カーネルコード。フラッシュ ROM に圧縮・暗号化された状態で格納されている。
- **バージョン 1.0 セキュリティ:** 2BL に RC-4 暗号を使った Xbox の初代セキュリティシステム。
- **バージョン 1.1 セキュリティ:** TEA ハッシュを使ってフラッシュ ROM の特定領域を検証する Xbox の第 2 世代セキュリティシステム。バージョン 1.1 セキュリティを搭載した Xbox で確認されている最も早い製造日は 2002 年 8 月頃だ。

前章では、最終的にサウスブリッジチップ内の秘密情報として隠されていた RC-4 鍵を取り出した、Xbox セキュリティメカニズムへの盗聴攻撃を説明した。本章では、仲間たちが考案した Xbox へのその他の攻撃、および改訂された Xbox セキュリティスキーム（ここではセキュリティバージョン 1.1 と呼ぶ）への攻撃を紹介する。

バックドアとセキュリティホール

Xbox へのバックドア攻撃の一形態として、シークレットブートコードによるハードウェア初期化の根本的な弱点を突く手法がある。

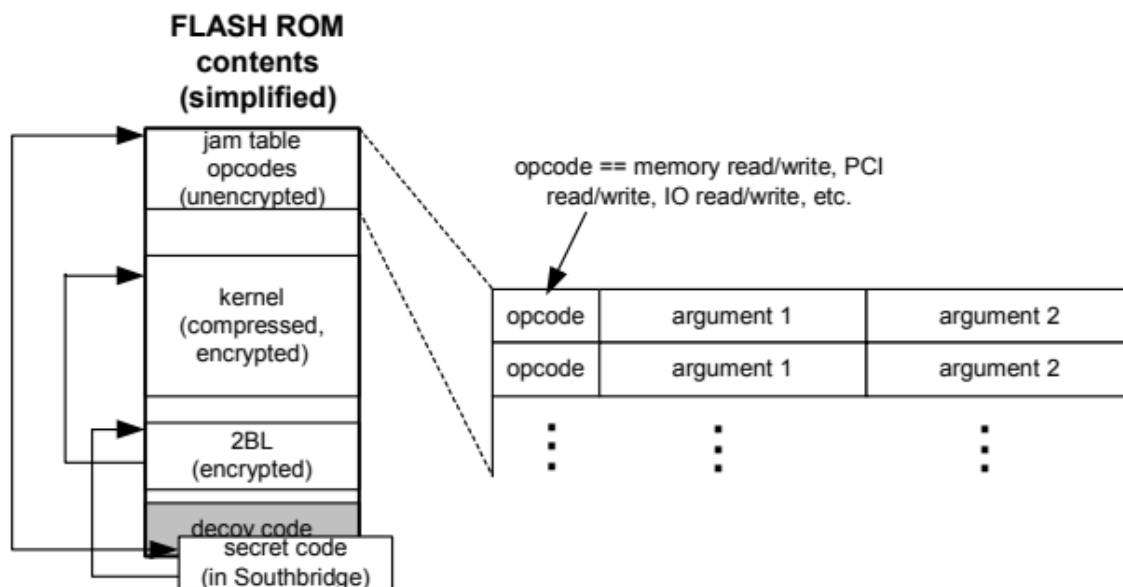


図 9-1: フラッシュ ROM の構成と、ジャムテーブルのオペコードの位置関係。

この弱点の根本的な原因は、ハードウェア初期化を担う強力なジャムテーブル用オペコードインタプリタが、その命令群を検証されていない平文で保持していることだ。

ヴァイザーのジャムテーブル攻撃

Xbox への攻撃の一形態として、ハードウェア初期化シーケンスを改ざんする手法がある。Xbox のハードウェア初期化は、フラッシュ ROM の「ジャムテーブル」と呼ばれる暗号化されていない領域に格納された命令群をオペコードインタープリタが読み込む仕組みで行われる（図 9-1 参照）。ジャムテーブルのエントリは <オペコード, 引数 1, 引数 2> の組として、フラッシュ ROM の最も低いアドレス付近に格納されている。利用可能なオペコードには、x86 アーキテクチャのすべてのアドレス空間に対するメモリの読み書き機能が含まれる。ジャムテーブルは暗号化されておらず、改ざん検証も行われていないため、フラッシュ ROM 内の 2BL を RC-4 で復号する前に、不正な命令や変更されたハードウェア状態を Xbox のメモリに「仕込む」オペコードをジャムテーブルに挿入できる。

ジャムテーブルの改ざんを応用すると、RC-4 鍵を知らずともカーネルの平文を取り出すことができる。「Visor」という名前しか知られていないハッカーが、この手法を最初に僕に教えてくれた。Visor のアプローチを要約すると次のようになる。

1. Xbox を通常どおり起動する。通常の起動プロセスの中で、復号済みのカーネルイメージがメインメモリに展開される。
2. Xbox への電源を維持しながら、フラッシュ ROM のジャムテーブルを、メインメモリの特定領域をフラッシュ ROM のバスなど監視しやすい場所にコピーする別テーブルに切り替える。
3. Xbox の CPU をソフトリセットする。これにより、メインメモリを消去せずにハードウェアを再初期化できる。
4. 改ざんされたジャムテーブルプログラムが実行される間、メインメモリの内容を記録する。

手順 2 で必要となるフラッシュ ROM 内容の動的切り替えは、ROM エミュレータを使うか、余分なアドレスビットを複数のスイッチに配線した大容量 ROM を使うことで実現できる。

Visor はさらに、Xbox の CPU の命令ポインタ (IP) を掌握する精巧なハックの一部としてジャムテーブルを利用する方法も説明した。このハックをより深く理解するために、無効なフラッシュ ROM イメージに対してシークレットブートコードがどう振る舞うかを詳しく見てみよう。

フラッシュ ROM イメージ内の 2BL を復号した後、シークレットブートコードは 2BL の末尾付近にある特定のマジックナンバーを確認する。無効なフラッシュ ROM イメージの場合、このマジックナンバーが一致しないため、CPU は 0xFFFF.FFFA にある短い命令列へジャンプする。この命令列は物理メモリの最後のアドレス 0xFFFF.FFFF まで続く。CPU がこの無効な ROM イメージの最後の命令を実行すると、IP が 0xFFFF.FFFF から 0x0000.0000 にロールオーバーする際にコードセグメント境界エラーが発生してクラッシュするはずだ。しかし実際にはそうならず、CPU は 0x0000.0000 にある命令が有効であれ無効であれ、そのまま実行しようとする。通常この命令は無効であり、命令フォルトによって CPU は停止する。だが、ジャムテーブルのメモリ書き込みオペコードを使えば、

Xbox のジャムテーブル初期化シーケンス中に有効な命令を 0x0000.0000 に置くことができる。

つまり、暗号化されたフラッシュ ROM イメージを破損または消去し、ジャムテーブルを改ざんしてフラッシュ ROM 内の独自の非暗号化コードへのジャンプ命令を挿入することで、著作権で保護された著作物へのアクセスを有効に制御する暗号技術やそれに類する技術的手段に一切触れることなく、Xbox の CPU の命令ポインタを制御できる。したがって、このハックは DMCA (デジタルミレニアム著作権法) のもとで合法かもしれない。「かもしれない」と言うのは、DMCA には曖昧な規定が多く、解釈を明確にする判例がほとんどないからだ。このアプローチの合法性の根拠は、Microsoft の著作権で保護されたコードが実質的に復号・実行されないという事実にある。唯一の例外は、サウスブリッジのシリコンにハードワイヤードされているために実行されるシークレットブート ROM の一部だ。今日のハッキングコミュニティが直面する法的問題については、第 12 章「注意せよハッカーよ」でより詳しく論じる。

MIST プリマチュア・アンマップ攻撃 1

ハッカーが Xbox を制御した場合に備え、サウスブリッジチップ内のシークレットブートコードは、終了する直前に自身をアンマップ (メモリマップから除外) する。つまり、実行が終わった時点でシステムから永久に隠れてしまう。そのため、メモリ上位 512 バイトにアクセスしようとするユーザプログラムには、シークレットブートコードの代わりにフラッシュメモリのデコイブロックが見える。Xbox-Linux プロジェクトのリードである Michael Steil は、この機能を逆手に取る方法を発見した。

アンマップ処理は PCI 設定空間のハードウェアレジスタ 0x8000.8008 への書き込みによって行われる。基本的な戦略は、初期化シーケンスが完了する前に 0x8000.8008 に書き込んでシークレットブートコードをアンマップするジャムテーブルオペコードを追加することだ。この時点ではキャッシュがオフになっているため、プロセッサはデコイブロックから命令をフェッチして実行し始める。デコイブロックはフラッシュ ROM の一部なので自由に変更できる。問題は、ジャムテーブルインタープリタが 0x8000.8008 への書き込みをブロックしている点だ。しかし、サウスブリッジチップセットの PCI 設定空間デコーダにバグがあり、アンマップ命令が複数のエイリアスアドレスに反応してしまう。具体的には、「function」ビットフィールドが無視されるため、0x8000.8X08 (X が 0 以外) への書き込みでも同様の効果が得られ、このアドレスへの書き込みはジャムテーブルインタープリタにブロックされない。つまり、MIST ハックで CPU の命令ポインタを制御するには、フラッシュ ROM 内のデコイブロックを自分のコードで書き換え、ハードウェア初期化中にシークレットブート ROM をアンマップする適切なジャムテーブルオペコードを追加すればよい。

Microsoft の反撃

セキュリティホールが発見されると、Microsoft がすぐにセキュリティスキームを刷新するという憶測が広まった。2002 年 8 月、オーストラリアで静かに新しいマザーボードを搭載した Xbox が出回り始めた。新セキュリティシステムに関する公式情報は意外な情報

源からもたらされた——Xbox のチップセットを製造する nVidia だ。2002 年第 2 四半期の振るわない決算を受けて、nVidia の広報担当者はその理由の一つとして次を挙げた。

「Xbox について言えば、ボリュームディスカウントのマイルストーンに達し、利益率がさらに低下した。また、MIT のハッカーによって MSFT が新しいセキュリティコードに移行した際に使えなくなった Xbox MCP の在庫評価損と、Athlon ベースのパソコン需要を見込んで製造した nForce チップセットの余剰在庫の Q2 における評価損を計上した。」 — Derek Perez、nVidia 広報ディレクター 2

Xbox セキュリティバージョン 1.1 のリバースエンジニアリング 3

セキュリティコードの変更内容の詳細は、2002 年 10 月まで明らかにされなかった。その月、Andy Green というハッカーが英国で入手できた最初のバージョン 1.1 Xbox の調査を始めた。マザーボード上での Xbox バージョン 1.1 と 1.0 の外観上の違いは微妙なものだった。GPU のファンが大型ヒートシンクに置き換わり、USB サブカードがマザーボードに統合され、PLL クロックシンセサイザチップが省略されていた。あちこちのフィルタキャパシタも削除されていたが、大きな変化はないように見えた。しかしさらに調査すると、MIST 攻撃に使われた脆弱性はふさがれているものの、ジャムテーブルのオペコードは変更されていないことがわかった。Xbox へのアクセスのキーとなる LPC バスも変更されていなかった。

Andy は Asterisk という仲間のハッカーが考案した手順を使い、1 日で MCPX ROM を取り出すことができた。この手順は、以前に特定されていたセキュリティホールとバックドアの非公開の組み合わせを活用したものだ。ROM の内容を初期解析したところ、セキュリティが根本的に異なる方式で実装されていることが明らかになった。バージョン 1.1 Xbox を大まかに調べると、旧来のセキュリティ・バイ・オブスキュリティの仕組みが廃棄され、公開鍵暗号の強度に基づくスキームに置き換えられていた。

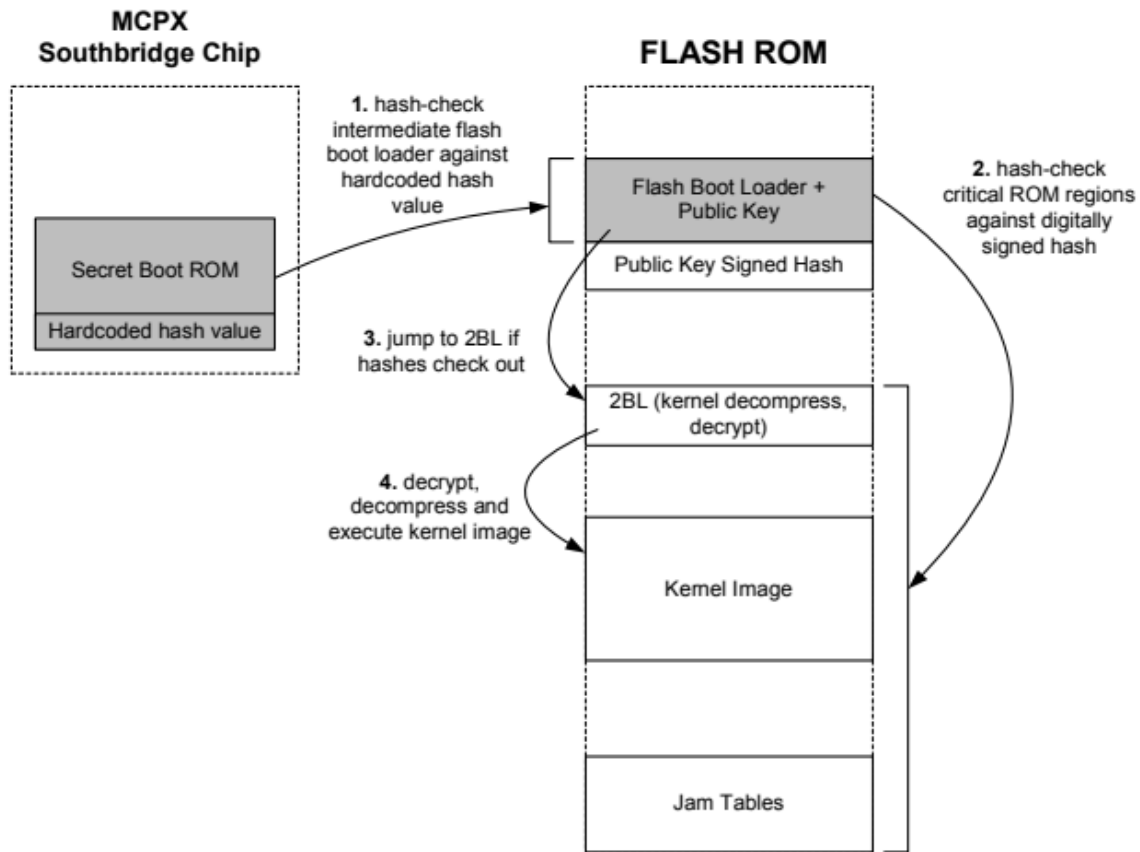


図 9-2: Xbox セキュリティバージョン 1.1。MCPX シリコンを交換しなければ変更できない領域はグレーで示されている。

しかし、Microsoft による新セキュリティスキームの実装はやや直感に反するものだった。MCPX に内蔵されたシークレットブート ROM のサイズが引き続き 512 バイトのままだったため、フル公開鍵デジタル署名アルゴリズムをシークレットブート ROM 内に収めることができなかった。そこで Microsoft は、シークレットブート ROM 内では軽量ハッシュを使い、FBL（フラッシュブートローダ、Flash Boot Loader）と呼ばれるフラッシュ ROM の領域を検証することにした。FBL には、フラッシュ ROM のデジタル署名検証に必要なコード——RSA 暗号、SHA-1 ハッシュ、Microsoft の Public Key、ドライバプログラム——が含まれている。FBL はそのハッシュ値がシークレットブート ROM 内の定数と一致した場合にのみ実行される。つまり FBL は、変更可能なフラッシュ ROM に格納されているとはいえ、理論上はシークレットブート ROM と同等の不変性を持つ。

このスキームはかなり堅牢に思えたが、ハッキングコミュニティは諦めなかった。シークレットブート ROM のハッシュを詳しく調べた結果、ケンブリッジ大学コンピュータ研究所の David Wheeler と Roger Needham による TEA（Tiny Encryption Algorithm、超軽量暗号アルゴリズム）に基づいていることを突き止めた。協力者の Franz Lehner は、TEA ハッシュの弱点についてニュースグループ sci.crypt に問い合わせを送った。

金曜午後、10月11日に回答が届いた。John Kelsey、Bruce Schneier、David Wagner が執筆し CRYPTO 1996 で発表された論文は、TEA 暗号の鍵スケジューラに弱点があることを指摘していた——すべての鍵には、特定のビットペアを反転することで生成できる3つの関連鍵が存在するという弱点だ（この弱点と TEA 暗号については第7章でより詳しく説明している）。翌土曜日、Andy Green は XboxHacker.net に次の投稿を行った。

ああ、僕は所詮ただの人間で、確かに弱点もある。

OK、この領域の最初の5バイトを公開する程度なら問題ないと思う。

ffffd400: E9 83 01 00 00

これは relative longbranch で 0xffffd588 へ飛ぶ。

DWORD として b31 をフリップすると（その DWORD アドレス+1 のビットも同様にフリップ）、代わりに 0x7fd588 へ分岐する……それは何だ、8M 上、どこへ……どこへ……

RAM がある

Xcodes.. Visor の ram プッシュメソッド……（MCPX で RAM Write X-Code を探す）

X-Code オペコード 3 … 無制限

帽子をしっかりと被っておけ！テストの時間だ！

（mviz、なんと素晴らしいタイミングのひらめきだ。不思議な見えない力が助けてくれているような気がして、本当に感謝している！）4

つまり、TEA 暗号の関連鍵の弱点により、FBL 内で隣り合うダブルワードのペアは、最上位ビット1ビットを変えるだけで結果のハッシュ値を変えずに改変できる。この弱点により、Andy とそのチームは単一のジャンプ命令のターゲットをメインメモリ上の特定アドレスに書き換えるだけの余地を得た。

プロフィール：Andy Green

自己紹介と、ハッキングとの出会いを教えてください。

37 歳、英国のイーストミッドランズ地方ケタリング近郊に住んでいます。妻、4 人の子供、2 匹の猫と暮らしています。

12 歳頃から、兄が Commodore Pet を買ったことをきっかけにコンピュータに興味を持ちました。この 1MHz 6502 マシンで何ヶ月も熱中しました。最初は雑誌の BASIC コードを打ち込んでいましたが、やがてゲームを作るようになり、最終的には機械語で素晴らしいキャラクターセル版スペースインベーダーを作りました。機械語とは CPU を直接 16 進数で命令するものです。今でも 6502 の一般的なオペコードを 16 進数で覚えています。これほど大変な作業だったので、次のプロジェクトは機械語で書いたアセンブラにしようと決めました。1978 年当時はインターネットがなく、商用アセンブラを買う余裕もなかったし、それをコピーしてくれる知り合いも周りにいませんでした。

できあがったアセンブラはかなり貧弱なものでしたが、十分機能しました。これで正しいツールを持つことの価値を学びました。アセンブラで書けばはるかに速く開発できるし、相対ブランチを手作業で計算するようなミスも完全になくなります。次は BBC Model B を手に入れ、またツールとゲーム作りに熱中しました。名門校の奨学金のオファーがありましたが断り、代わりに 16 歳で高校を中退しました。自分が興味を持ったことは何でも独学できると確信していたからです。

このマシンと、もう一つの 6502 プラットフォーム Oric でゲームをいくつか売り、そのお金でアセンブラやその他の開発ツールを作る会社を立ち上げました。途中で C と C++ を習得するたびに、バグや時間の無駄が大量に消えていきました。

「人類の進化」の絵みたいです。ネアンデルタール人的な相対ブランチ計算から、仮想関数を扱うホモ・エレクトゥスの段階まで。

同時に、デジタルハードウェア設計も独学で探求し始めました。ハードウェアとソフトウェアは同じコインの両面であることに気づきました。教育の間では完全に別物として扱われますが、ある論理関数をソフトウェアで実装するかハードウェアで実装するか、あるいは両者の組み合わせにするかは実装上の詳細にすぎません。両方に足をかけているとデザインの本質がより深く見えてきます——たとえば C++ は、インターフェースの重要性という面でエレクトロニクスから多くの概念を借用していると言えます。

Xbox に興味を持つ少し前は、オックスフォードにオフィスを持つ米国系企業で多くの仕事を担当していましたが、最後の仕事はスマートカードのシリコン設計でした。設計は面白く、現場には素晴らしい人たちもいましたが、経営陣の政治問題にますます落胆していきました。英国に拠点を置いているというだけで、サンノゼのスタッフの 3 分の 2 の給料しかもらえないのも問題でした。無報酬で特許を取られた件については話したくありません。2001 年 12 月、誠実さはお金より大事だと気づいて退職し、再び自分で仕事をすることにしました。

この会社を辞めたことでいくつかの辛い経験をして、それを消化している間に、知的財産に関わる普通の企業の醜い、貪欲で支配的な本能と、GPL プロジェクトの性格や特許・著作権法の厳格さを緩和しようとする人々の姿勢との大きな落差に気づきました。次第に Microsoft と以前の会社を同じ目で見えるようになっていきました。

そんな時期に、Slashdot で bunnie のハックについて読みました。複雑な気持ちで bunnie の手法を読みました。主な感想は、bunnie が使った FPGA を 1989 年から使っていたので自分にもできたはずという思い、攻撃の簡潔さへの感嘆、そして自分が同じくらいクールで面白い——自分の哲学的な信念とも合致する——ことをやっていなかったことへの自己嫌悪でした。代わりにコーヒーを飲みながら Slashdot を読んでいるだけで、何も貢献していなかったのです。(余談ですが、これは多くの Slashdot 読者に共通する経験だと思います。誰かのクールなハックを読んで少し嫉妬し、刺激を受ける、あの感覚です。なぜそんなことをしたいのかと常にやじを飛ばす背景ノイズの正体はこれだと思います。)

その後数週間で、Xbox の内部についてできるだけ多くの情報を集めました。xboxhacker.net が非常に役立ちました。そこで Xbox Linux プロジェクトが始まったばかりの頃に Michael Steil と出会いました。間もなく、たとえば Milksop プロジェクトのように自分が貢献できる面白いプロジェクトを見つけることができました。Surferdude 氏の助けも借り、リセットなしに Xbox を起動して動かし続けることができた最初のクリーンな ROM を完成させることができました。これが後に crom 1MB Linux と Cromwell (Xbox Linux 向けクリーン ROM)

の基礎になりました。最初のハックと設計が済んだあとは、ほぼ Xbox Linux のゴールに向けた作業に集中しました。

なぜ Xbox をハックするのですか？

理由は人それぞれですが、僕の場合は Microsoft の目に余る独占禁止行為への怒りが大きかった——何でも否定する、何でも上訴する、何でも引き延ばす、その間に市場にコストを下回る価格で Microsoft 専用 PC である Xbox を何百万台と投入する——というものです。欧州や米国の代表者がこれをさほど問題視しない以上（おそらく、EU でも最近あったように、彼らは Microsoft のもとへ行ってお褒美をもらうつもりなのかもしれない）、GPL と Linux という武器を使ってこの肥大した独占企業の邪悪な計画に風穴を開けることの一端を担えるなら光栄だと思います。目をつむって株のことだけ考えている人のことはわかっています。でも、そこで働く誠実な人々——そして Microsoft の大部分の社員は誠実なはずですが——が、そんな怪物のために働くのはつらいことでしょう。

2002 年を通じていくつかの契約を得ることができ、年後半は最初の Linux カーネルを crom で動かし、Cromwell で Xbox の主要周辺機器をコントロールして HDD または CD から Linux を起動できるようにすることに専念できました。その後、プロジェクト A の賞金（寄付者 Michael Robertson 氏に感謝します）の自分の取り分のおかげで、少なくともあと数ヶ月はフルタイムで活動を続けられます。

最後に何かアドバイスはありますか？

最後に伝えたいのは、特に若い人たちへのメッセージです。興味が湧いたことについては、自分の脳の声に耳を傾けてほしい。気になることを掘り下げて理解しようとするのを恐れないでください。何かを理解したいという気持ち、あの渴望感は、「この知識はいつか役に立つかもしれない」と脳が教えてくれているサインです。その声に十分耳を傾けていれば、いつか何か小さな違いを生み出すための正しい知識を正しいタイミングで持てる可能性が高まります。

そのアドレスには、先に述べたジャムテーブルコードを使って、任意のユーザコードへのジャンプ命令があらかじめ書き込んでおける。Xbox ハッキングコミュニティはひとつになって英雄的な努力を重ね、3 日間で Xbox セキュリティバージョン 1.1 を解析きった。同様に勇敢な別のチーム Xecuter も、同じ時間枠でセキュリティを解析していた。

この話の最初の教訓は、セキュリティはその最も脆弱な部分の強さで決まる、ということだ。RSA 暗号と SHA-1 ハッシュのデジタル署名としての堅牢さに疑いの余地はほとんどないが、それらはセキュリティシステムの唯一の要素ではなかった。フラッシュ ROM へのブート ROM の信頼を引き継ぐために使われた TEA 暗号には、ハッカーが強力なデジタル署名アルゴリズムを迂回できる弱点があった。

第二の教訓はこうだ——複雑さは弱点を生む。複雑なシステムは設計・テスト・分析が難しい。Xbox バージョン 1.1 のセキュリティはおそらく短期間で実装されたため、弱点を分析するための十分な時間がなかったのだろう。あるいは、Microsoft は TEA の弱点を知っていながら、FBL をシリコンに固定するリスクを軽減するためにこのバックドアを意図的に組み込んだのかもしれない。Microsoft が意図的にこのバックドアを入れたというのはかなり考えにくい話だ。MCPX シリコンを変更するのは非常に高コストだからだ（そ

のコストは最終的に nVidia のバランスシートに計上されたけれど)。一方で、複雑さを完全に避けることも難しい。MIT での僕の指導教員である Tom Knight は、かつてこう言っていた——「この世のデザインには 2 種類ある。役に立つものと、正しいことを形式的に証明できるもの、だ」。ある意味では、現実のシステムのセキュリティを確保する唯一の方法は、その詳細を公開し（セキュリティ・バイ・オブスキュリティはなし！）、あらゆる角度から分析にさらすことだ。ある意味、Xbox セキュリティの徹底的な分析は、ハッキングコミュニティのおかげで Microsoft に費用を一切かけることなく実施されている。

もし Microsoft がシークレットブート ROM により強力なハッシュ関数を使っていたとしても、Xbox にはまだ試されていない有効な攻撃手法がいくつかある。HyperTransport バスに対して、精密にタイミングを合わせたパルスで信号を過剰駆動する中間者攻撃を仕掛けることができる（第 8 章参照）。各 HyperTransport バスの配線にはマザーボードの部品面から見えるテストポイントがあるため、この攻撃はかなり簡単に実装できる。完全なハードウェアソリューションとしては、ポゴピンを使ったベッド・オブ・ネイルズ方式のテストコネクタを備えた基板に FPGA を搭載する形が考えられる。この基板ははんだ付け不要でテストポイントに押し当てただけで使える。もう一つの攻撃は、CHES 会議で Adi Shamir が僕に提案したものだ——CPU のクロックや電源に時限グリッチを加えてジャンプターゲットアドレスの計算を乱す手法だ。この種の攻撃は暗号スマートカードのプロセッサで実際に成功している。こちらの攻撃もユーザが取り付けられるモジュールとして安価に実装できる。（セキュリティを一回だけ突破して秘密鍵や重要コードのブロックを取り出すことが目的の場合、ハッカーが使える攻撃の幅はずっと広くなることを念頭に置いてほしい。）

バックドアの脅威

この章で示したとおり、バックドアを探すことは暗号によって保護されたハードウェアへの攻撃として実用的な手法だ。Xbox でバックドアが比較的高い確率で見つかった理由の一つは、Xbox がベンダーによってコンピュータを PC 上で暗号技術によって保護しようとした最初の重要な試みだったからだ。

プロフィール：Franz Lehner

Franz Lehner (29 歳) は、オーストリアにガールフレンドと住んでいます。電気工学を 5 年間学び、現在は ISP を運営しながら「自動化ソリューション」のプログラミングをしています。余暇には、楽しく教育的なプロジェクトを探しています。

bunnie の Xbox ハッキング文書を見つけた後、sourceforge.net で Xbox-Linux チームと出会い、チームプログラミング、Linux カーネルのハッキングとデバッグ、暗号システムについて学ぶために Xbox-Linux プロジェクトに参加しました。Palladium などの関連システムへの理解を深める目的でもプロジェクトに加わりました。

Xbox 経験から得た教訓にもかかわらず、将来のセキュア PC 実装は依然としてハードウェアセキュリティの弱点を抱えるリスクがある。PC のレガシーがオープンで安全でないハードウェアアーキテクチャだからだ。

PC ハードウェアは複雑でありながら脆弱であり、それを基盤に信頼の連鎖を構築することは難しい。根本的な問題として、PC 内の各コンポーネントはその物理環境を「信頼する」設計になっている。市販の集積回路コンポーネントの仕様書には、動作が保証される温度・電圧・周波数などの条件範囲が明記されている。この最大定格を超えると、デバイスの動作は「未定義」となり、何が起きるかわからない。ほとんどのチップエンジニアは、通常の動作条件でチップを動作させることだけでも十分難しいため、範囲外の条件から回路がうまく回復するような設計をしようとすら考えない。しかも、コンシューマ向け製品はコスト感度が高く、堅牢なフォールトトレランス対策を組み込むと価格競争力を失ってしまう。

そのため、チップは通常、内部エラーチェックなしで実装される。算術論理演算装置、つまり ALU (Arithmetic Logic Unit) —— CPU の演算の「頭脳」—— が 2 つの数を誤って計算しても、その問題は症状としてしか現れない。エラーを引き起こした出来事からずっと後になって、その影響が観測されるだけだ。範囲外の条件によって誘発された故障を利用する攻撃は、ソフトウェア世界のバッファオーバーランの類似物だと考えることができる。

PC アーキテクチャのもう一つの問題は、プロセッサがコード環境を信頼しすぎることで、Pentium プロセッサアーキテクチャには、安全でないコードと安全なコードを区別するためのハードウェア機構がない。バグや誘発された障害によって命令ポインタが安全でないコードセグメントに紛れ込んでも、プロセッサは気にせずそのコードを実行してしまう。

[!NOTE] ハードウェアセキュリティレベルに基づくコードの区画化は、サンドボックス化とは異なる技術だ。サンドボックス化は、ユーザプログラムが秘密情報や保護されたコード・データから指示を受けたり相互作用したりする必要がある状況には、十分な解決策を提供しない。最近では、データタグによってセキュリティ監査ログのような仕組みを埋め込むことでこの問題を解決できる新しいプロセッサアーキテクチャが提案されている。4

バックドアのもう一つの発生源は、複雑なチップには必ずと言っていいほど存在する設計上のバグだ。既知のバグ (エラータとも呼ばれる) を抱えたままチップを出荷することはよくある慣行だ。たとえば、Intel i860 XP プロセッサ (1991 年に最初にリリースされたもので、Pentium 4 プロセッサ向けに最近リリースされた i860 チップセットとは別物) は、プロセッサのデータシートに匹敵するほどのエラータ集を抱えて出荷された。身近な例では、MIST プリマチュア・アンマップ攻撃を可能にした nVidia MCPX のアドレス空間デコーダのバグがある。ほとんどのエラータには簡単なワークアラウンドがあるか、通常の動作条件では機能に軽微な影響しか与えない。しかし、キャッシュコヒーレンシ、アドレスデコーディング、メモリ管理に関連するエラータの中には、重大なソフトウェアセキュリティホールにつながるものもある。

Xbox の場合、ハードウェアのバックドアがビジネスに与える影響はおそらく小さい。Microsoft がゲーム売上の一部を失うかもしれないが、海賊行為による損失は Xbox のハードウェア販売で Microsoft が被る損失に比べれば微々たるものだ。また、Xbox は単なるゲームコンソールにすぎない——Xbox のセキュリティ上の弱点によっておぼ

あちゃんの銀行口座が空になったり、クレジットカード番号が盗まれたりはしない。しかし、信頼される PC が普及すれば、ゲームの売上以上のものが危険にさらされる。トラステッド PC アーキテクチャが従来の PC からの根本的な変革でなければ、人々は知らないうちに金融の秘密情報や個人データのセキュリティを信頼できないハードウェアに委ねることになる。

人生のほとんどのことと同様に、最初の一步は学ぶことだ。ゲームコンソールをいじる行為を含めても、ハードウェアセキュリティについて学べば学ぶほど、明日のセキュリティシステムは良くなる。さあ、次のレッスンへ……

注

1. Andy Green の第 19 回カオスコミュニケーションズコンgresにおける Xbox セキュリティハッキング発表より。
2. The Inquirer の記事より (<http://www.theinquirer.net/?article=4735>)。
3. Andy Green の第 19 回カオスコミュニケーションズコンgresにおける Xbox セキュリティハッキング発表より。
4. <http://www.xboxhacker.net> “Xbox Hacker BBS -> Xbox Hacking (TECHNICAL) -> BIOS/Flash ROM/Firmware -> Xbox Linux チームのニュースより MS が’ハッシュをめっちゃくちゃにした’、内部が明らかに” スレッドの投稿より。
5. <http://www.ai.mit.edu/projects/aries/Documents/Memos/ARIES-15.pdf>。 「 A Minimal Trusted Computing Base for Dynamically Ensuring Secure Information Flow」 by Tom Knight and Jeremy Brown.

第 10 章 さらになるハードウェアプロジェクト

Xbox は PC アーキテクチャをベースとしているため、ハードウェアプロジェクトを進める際に PC 界の技術と知識をそのまま流用できる。その結果、PC のハードウェア、モニター、ケーブル、周辺機器が次々と Xbox 向けに転用されてきた。本章では、世界中のハッカーたちが発見・文書化・実装してきたハードウェアプロジェクトを紹介する。

LPC インターフェース

LPC (Low Pin Count) インターフェースのバージョン 1.0 は、1997 年に Intel が策定した規格だ。LPC インターフェースはロイヤリティフリーのバスで、ISA や X-bus (メモリや汎用 I/O デバイスのための ISA 互換拡張バス) を明示的に搭載しないシステムを実現するために設計されている。

LPC インターフェースが必要とされた背景には、標準 PC に搭載されている低ビットレート・多ピンのデバイスが互いに異なるインターフェースを持つという問題がある——フロッピーディスク、キーボード、マウス、シリアル、IrDA、パラレル、ISA、ブート ROM などがその例だ。これらのデバイスが消費する合計帯域幅は小さいが、接続するために必要

な信号線の本数は、PCIやAGPなど高帯域バスに必要な信号線数をはるかに超えてしまう。さらに悪いことに、すべてのPC構成でこれらのレガシーI/Oデバイスが必要なわけではなく、使われないピンや機能はそのままコストを押し上げる。チップパッケージのピン1本のコストは、同等の機能を持つシリコン面積のコストよりも割高だ。(目安として、パッケージのピン1本は1セント。一方、0.13 μm プロセスのシリコンでは約1万ゲート——小規模なプロセッサを実装できる規模——が同じく1セントのシリコン面積で実現できる。ただしこれはボンドパッド制約がない設計を前提とする。1)

LPC インターフェースは、この問題に対してシンプルな解決策を提示する。ISA バスに必要な36ピンに対し、わずか7本の必須ピンで済む低ピン数の単一バスによって高速動作を実現するのだ。レガシーI/Oおよび拡張機能はすべてこの高帯域バスにマッピングされ、システム設計者はいわゆる「スーパーI/O」チップを作れるようになる。スーパーI/Oチップを使えば、サウスブリッジチップのピン数を大幅に削減できる。さらに、機能をスーパーI/OチップとサウスブリッジチップAに分離することで、設計者は用途に最適な組み合わせを自由に選択できる。

LPCの物理インターフェース自体はシンプルだ。4ビットの双方向バスが33MHzのクロックで動作する。加えて、LPCバスサイクルの開始と終了を示すフレーミング信号と、すべてのLPC周辺デバイスを既知の状態にリセットする信号という2本の「サイドバンド」信号を持つ。さらにオプション信号として、DMA・割り込み機能や高度なI/OデバイスのPower Management機能を提供するものもある。(LPCバスとプロトコルの詳細については、IntelのLPC Interface Specification バージョン1.1を参照。仕様書はIntelの企業サイト <http://www.intel.com/design/chipsets/industry/lpc.htm> で入手できる。)

Xbox の LPC インターフェース

XboxのマザーボードにはLPCインターフェースが搭載されている。ここではデバッグ・テスト用バスとして使われており、キーボードやマウスの接続、および診断目的のオルタネートブートROMの接続が可能だ。LPCインターフェースは、XboxのフラッシュROMが使用できない場合にオルタネートブートコードをロードするために起動する。フラッシュROMがない状態は、フラッシュROMデータバスの最下位ビット(D0)を強制的にゼロボルトにすることでシミュレートできる。

LPCインターフェースはXboxの生産ラインに不可欠だという見方が多い。完成品のXboxに対して、LPCインターフェース経由で包括的な自己テストプログラムを書き込めるからだ。CPUを高速なテストコントローラとして活用することで、高価なテスト装置を使わずに工場のラインで欠陥品を迅速かつ効率的に特定できる。

ハッカーにとって、LPCインターフェースが提供するオルタネートブートROM機能は、Xboxにコードを入れ込む理想的な手段だ。Xboxの暗号的に保護されたブート手順が完全に解明された今、LPCから読み込めるブートROMイメージは誰でも作成できる。実際、Xbox向けのオルタネートブートROMデバイスを販売しているいくつかのベンダーは、XboxマザーボードのLPCコネクタのピン配置が規則的であることを利用して、ほとんどを一切使わずに取り付けられるROMデバイスを開発した。

LPC インターフェースの活用

これらのデバイスは、Xbox のテスト工程でも使われているのと同様のスプリング内蔵接触プローブピン——ポゴピン——を使用している。押し当てるだけで LPC インターフェースに接続できるため、はんだ付けが一切不要だ。(Xbox の LPC バスのピン配置については付録 F「Xbox ハードウェアリファレンス」を参照。)

LPC インターフェースが業界標準であることは、Xbox のハードウェアハッカーにとって非常に好都合だ。第一に、スーパー I/O チップから内蔵 LPC インターフェースを持つファームウェア ROM まで、LPC 互換インターフェースデバイスが豊富に存在する。第二に、LPC インターフェースが汎用 PC の診断・利便性バスとして広く普及していることで、LPC インターフェースデバイスの使用・販売に伴う法的リスクが軽減される。LPC インターフェース用のファームウェア ROM は、Xbox 固有のコンテンツなしに販売できる——エンドユーザーが安価な PC 用アダプターを使って LPC デバイスを簡単に再プログラムできるからだ。また、Xbox の LPC コネクタのピン配置が Intel の汎用 PC 向け推奨仕様とほぼ同一であることも、法的な観点で LPC ファームウェアデバイスに追い風となっている。結果として、Xbox 向けに販売される LPC ファームウェアデバイスは、標準 PC 向けのものとほとんど変わらない製品になる。

最初期の LPC ブート ROM デバイスのひとつを開発したのは Andy Green だ。このプロジェクトは「Cheapmod」と呼ばれ、LPC インターフェースを内蔵した 256 KB のフラッシュ ROM チップ「SST 49LF020」を LPC 互換ヘッダに配線したソケットに搭載した構成だ。Andy の Cheapmod ウェブページ (<http://warmcat.com/milksop/cheapmod.html>) には「2.50 ドルの SST 49LF020 を入手できれば、4 ドルで代替 BIOS を作れる」と書かれている。このデバイスは彼が作った「CheapLPC」プログラマ (<http://warmcat.com/milksop/cheapLPC.html>) を使って書き込む。PC のパラレルポートを使ったシンプルな装置で、LPC デバイスとゆっくりながら確実に通信して再プログラムできる。市販されているオルタネートファームウェアデバイスの多くは Andy の設計を基に、または参考にして作られており、Xodus/Matrix もその一つだ。Xodus/Matrix は Andy のオリジナル設計の特に興味深い派生品で、完全なはんだ不要インストール手順を実現した最初の Xbox オルタネートファームウェアデバイスだ。これにより、はんだ付けを敬遠するソフトウェア指向のハッカーにも Xbox ハッキングの扉が開かれた。(Xodus/Matrix の写真は図 10-1 を参照。) Xodus/Matrix デバイスにはコードが書き込まれていない状態で出荷され、オルタネートファームウェアイメージはユーザーが自分で用意する必要がある。

Xbox 向けに LPC インターフェース付きフラッシュ ROM チップを選定する際には、いくつかの重要な機能上の考慮点がある。最も重要なのは、Xbox のネイティブアーキテクチャがブート ROM に 16 MB のアドレス空間を割り当てているという点だ。物理的なブート ROM のサイズが 16 MB より小さい場合、その内容が 16 MB 全体に複製 (エイリアス) されて埋められる。この仕組みにより、Xbox 設計者は ROM チップのサイズを柔軟に選択でき、ボトム相対アドレスとトップ相対アドレスの両方を使うルーチンが問題なく動作する。

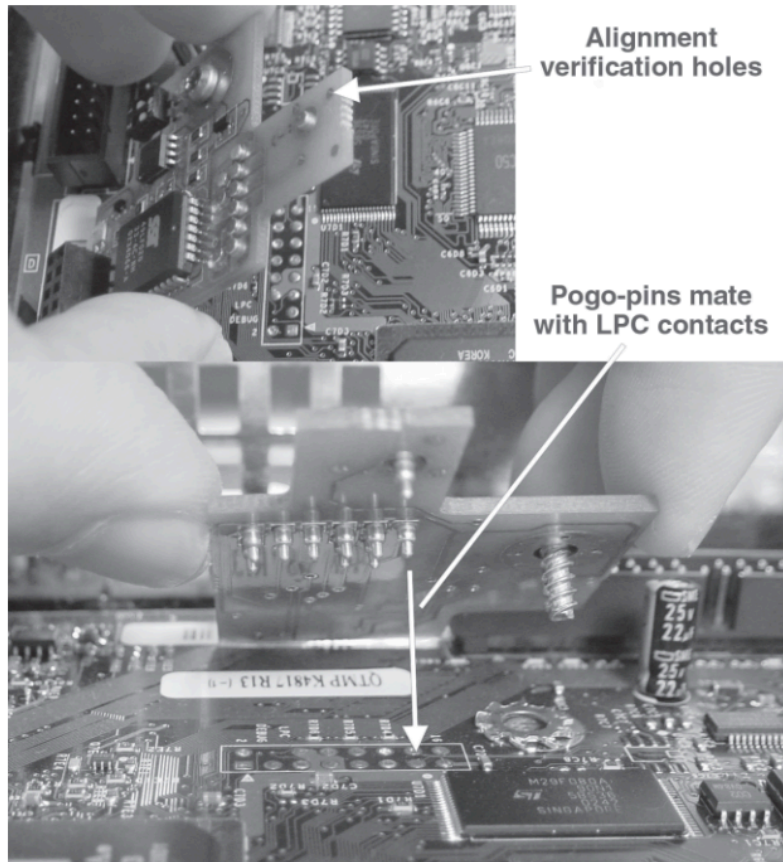


Figure 10-1: The solderless Xodus/Matrix alternate firmware device, showing the spring-loaded “pogo-pin” contacts that enable a solderless connection to the LPC connector on the Xbox motherboard.

Let’s make the concept of bottom- and top-relative addressing more concrete with an example. The addresses for the 16 MB boot ROM area in the Xbox spans from $0xFF00.0000$ to $0xFFFF.FFFF$. Programs on the Xbox that use bottom-relative addressing will compute addresses using $0xFF00.0000 + \text{offset}$ (bottom address plus offset), while programs that use top-relative addressing will use $0xFFFF.FFFF - \text{offset}$ (top address minus offset). Suppose a 1 MB boot ROM is installed in the Xbox. This means that the processor will see 16 identical copies of this 1 MB ROM spread evenly over the 16 MB ROM address space. In other words, the contents of the boot memory appear identical for every address $A + 0xFF00.0000 + n * 0x0010.0000$, $n = 0$ through 15, $A = 0$ through $0x000F.FFFF$. As a result, programmers can pack data into the smaller 1 MB boot ROM using both top- and bottom-relative addressing without having to change any of their code: A valid copy of the ROM image

図 10-1: はんだ不要の Xodus/Matrix オルタネートファームウェアデバイス。Xbox マザーボードの LPC コネクタにはんだなしで接続できるスプリング内蔵のポゴピン接点が見える。

ボトム相対アドレスとトップ相対アドレスの概念を具体的な例で説明しよう。Xbox のブート ROM 領域 (16 MB) は 0xFF00.0000 から 0xFFFF.FFFF までのアドレス空間を占める。ボトム相対アドレスを使うプログラムは $0xFF00.0000 + \text{オフセット}$ (ベースアドレスにオフセットを加算) でアドレスを計算し、トップ相対アドレスを使うプログラムは $0xFFFF.FFFF - \text{オフセット}$ (最上位アドレスからオフセットを減算) で計算する。たとえば 1 MB のブート ROM を搭載した場合、プロセッサは 16 MB 分の ROM アドレス空間に同じ 1 MB の ROM が 16 コピー並んでいるように見える。具体的には、アドレス $A + 0xFF00.0000 + n \times 0x0010.0000$ ($n = 0 \sim 15$, $A = 0 \sim 0x000F.FFFF$) のどのアドレスを参照しても、ブートメモリの内容は同一だ。その結果、プログラマーはコードを変更せずに、小さい 1 MB のブート ROM にトップ相対・ボトム相対の両方でアクセスするデータを格納できる——ROM イメージの有効なコピーがトップ側・ボトム側どちらのベースアドレスからも見えるからだ。

コラム：オルタネートファームウェアデバイスとモッドチップ

オルタネートファームウェアデバイスとは、ユーザーが指定したファームウェアを Xbox ハードウェア上で動作させるためのハードウェアモジュールだ。「モッドチップ」とは区別される——オルタネートファームウェアデバイスは空白の状態を提供され、著作権保護機構を回避する固有の機能を持たない。たとえば空白の LPC インターフェース ROM デバイスはオルタネートファームウェアデバイスだ——米国権利章典を書き込むことだってできる。空白の状態で出荷されるユーザー設置型フラッシュ ROM もすべてオルタネートファームウェアデバイスと言える。

一方モッドチップとは、俗にゲームのバックアップを動作させたり、DRM (デジタル著作権管理) ポリシーを変更・除去したりするために作られたデバイスを指す。したがってモッドチップという用語には、DRM ポリシーの変更を可能にするコードが書き込まれたブート ROM デバイスだけでなく、ROM を持たずに Xbox のファームウェアが実行のためにロードされる際にいくつかの重要な箇所を動的にパッチするデバイス (「パッチャー」と呼ばれる) も含まれる。

次に、Microsoft がコストを下げようとして 1 MB のブート ROM を 256 KB に縮小したとしよう。プロセッサは今度は 16 MB の ROM アドレス空間に同じ 256 KB の ROM が 64 コピー並んでいるように見える。それでもボトム相対・トップ相対アドレスを使う古いコードはすべて正常に動作する。重要な点として、Xbox の CPU はメモリ最上位から 16 バイト下のアドレス (「リセットベクタ」) から電源投入時にコードの実行を開始するようにハードワイヤードされている。一方、Xbox チップセットにワイヤードされたハードウェア初期化ルーチンは、16 MB のフラッシュ ROM 空間の下部付近の ROM アドレスを使う。その結果、Xbox ハードウェアは LPC ROM の実装として 16 MB のサイズか、またはより小さい ROM の内容をフラッシュ ROM アドレス空間全体にエイリアスするものが必要になる。(SST 49LF020 は ROM の内容をアドレス空間全体にエイリアスする数少ない LPC

フラッシュ ROM のひとつだ。ただしこのエイリアス機能は実はバグとも言える——上位アドレスビットを無視して ROM 内容をアドレス空間全体にエイリアスすることで、本来他の機能に割り当てられるべきスペースを占有してしまうからだ。そのため SST は「A ステップ」と呼ばれる改訂版「49LF020A」をリリースしており、こちらは ROM 内容をメモリ全体にエイリアスしない。したがって A ステップのシリコンは Xbox のオルタネートファームウェアデバイスとして機能しない。)

もう 64MB の SDRAM

目の鋭い観察者なら、Xbox マザーボードのトップサイドにチップがない空きスペースが 2 か所あり、現在メモリチップが搭載されている場所とよく似た形をしていることに気づくだろう。基板を裏返すと、さらに 2 か所の空きチップフットプリントがある。これらはメモリチップ用のフットプリントだ。これらの空きスペースの場所を図 10-2 に示す。

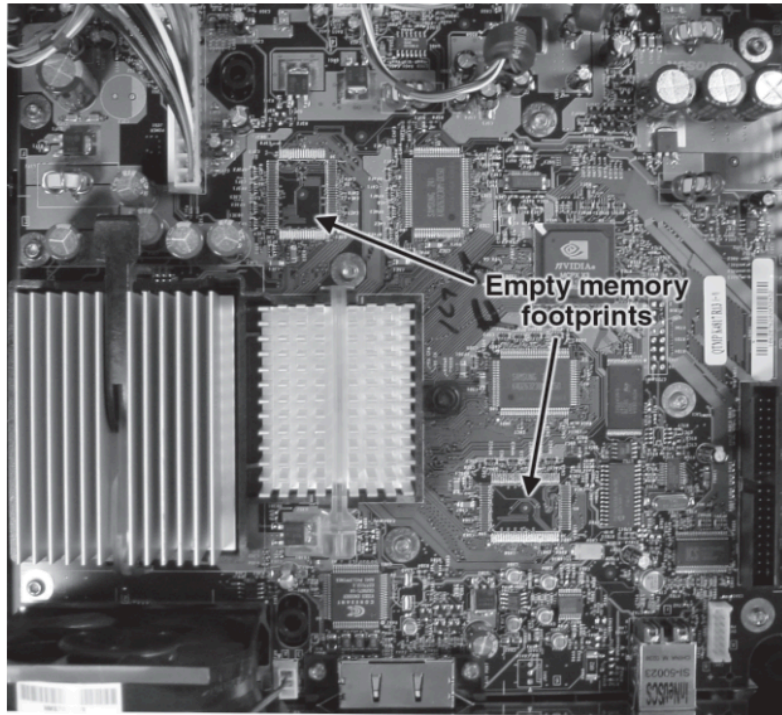


Figure 10-2: The unpopulated memory footprints on the Xbox motherboard.

Fiduciaries

Look at an unpopulated memory spot on the Xbox motherboard. The silver dot surrounded by a dark annulus inside these unpopulated chip footprints is called a *fiducial*. Fiducial patterns are used by circuit board assembly machines as reference points for aligning large chips with many pins. They are designed to be easily recognized by the machine vision systems employed in board assembly machines. Specially shaped fiducials can also be used to enable automatic identification of the orientation and type of a circuit board.

図 10-2: Xbox マザーボードのメモリチップ未搭載フットプリント。

コラム：フィデューシャルマーク（基準マーク）

Xbox マザーボードのメモリチップ未搭載スペースをよく見ると、暗い輪に囲まれた銀色のドットがある——これをフィデューシャルマーク (fiducial mark、基準マーク) と呼ぶ。フィデューシャルマークのパターンは、基板実装機械が多ピンの大型チップを正確に位置合わせするための基準点として使われる。基板実装機械が搭載するマシンビジョンシステムに認識されやすいよう設計されている。特殊な形状のフィデューシャルマークを使えば、基板の向きや種類を自動的に識別させることも可能だ。

次の疑問は当然こうなる——「Xbox マザーボードの空きスロットに適切なメモリチップをはんだ付けすれば、メモリを 128MB に倍増できるのか？」答えは Yes だ。ただし、チップセットが増設メモリを認識・使用できるようにするには、Xbox の初期化コードを変更する必要がある。さらに、増設メモリはグラフィックスやゲーム性能の向上には寄与しない——Xbox ゲームは増設メモリを活用するよう設計されていないため、増設分は通常そのまま使われない状態になる。これらの空きメモリスロットは主にゲーム開発者向けの特別なコンソール製造のために用意されている。ゲーム開発者は増設メモリを使って、比較的タイトな Xbox のメモリフットプリントへのゲーム移植を楽にしたり、ゲームイメージとは別にデバッグ、パフォーマンスモニタリング、テストユーティリティをメモリに常駐させたりできる。増設メモリはホームブリュースoftwareにも活用できるが、メモリチップの入手と取り付けの難しさから、Xbox のメモリ増設は実用的な改造というよりも「はんだ付けの練習として面白い」程度の作業に留まる。

Xbox VGA

Xbox VGA アダプターが何をするものかについて、少し混乱がある。多くの Xbox VGA アダプターは実際には TV-to-VGA コンバーターで、Xbox からのロードレゾリューション TV 出力をラインダブラーに通して低品質の VGA 表示を出力するだけだ。真の Xbox VGA アダプターは、Xbox がより高解像度のビデオ出力を行うよう設定し、VGA モニター上で TV 画質を超える表示を実現する。

VGA アダプターは AVIP (Audio Video I/O Port) コネクタの指定ピンを使って Xbox のグラフィックスモードを設定する。この方式の主な問題は、ゲームがこの高解像度モードをサポートするよう専用に作られている必要があることだ。そのため真の Xbox VGA アダプターに対応していないゲームも存在するが、幸い通常の TV アダプターケーブルを差し込むだけで TV 解像度に戻せる。

オリジナルの Xbox-VGA アダプターを開発したのは Ken Gasper だ。彼のウェブサイト <http://xboxvga.xemulation.com> でこのアダプターを販売しており、現在は「むき基板」形式と完全組み立て済み形式の両方を提供している。便利で回路組み立てスキルも磨ける Xbox ハードウェアハッキングプロジェクトを探しているなら、彼のむき基板を購入してアダプターを自分で組み立ててみる価値があるだろう。

Xbox AVIP のピン配線図は付録 F にある。

大容量ストレージの交換

XboxにはDVD-ROMドライブとハードドライブが搭載されており、どちらもPC標準のIDEインターフェースでXboxマザーボードと通信している。DVD-ROMドライブには独自の電源・DVDトレイ状態コネクタも付属する。Xboxで人気のある、また時に必要不可欠なハッキング作業のひとつが、これらのドライブの交換だ。

DVD-ROMを交換・調整しようとするユーザーが後を絶たないのは、XboxのネイティブDVD-ROMドライブがCD-RやほとんどのCD-RWメディアを読めないからだ。初めてXbox-Linuxをインストールしようとしているユーザーや、CD-Rコレクションの音楽をXboxのハードドライブにリッピングしようとしているユーザーにとって特に厄介だ。

Xbox DVD-ROMドライブを交換・調整する方法は多数ある。一部のXbox DVD-ROMドライブのモデルはレーザー強度を調整してCD-RやCD-RWメディアの読み取り能力を向上させられる。これはリスクを伴う操作で、レーザーの出力を誤って調整するとDVD-ROMドライブを永久に破壊する可能性があるが、正しく実行すればメディア互換性が向上したというハッカーの報告が多い。Xboxに搭載されるDVD-ROMドライブのスタイルやモデルは頻繁に変わるため、最新の情報と手順についてはウェブ検索を勧める。またXboxのDVD-ROMドライブを標準PC用DVD-ROMに丸ごと交換することも可能だ。ただしこの方法には二つの問題がある。第一に、通常のPC DVD-ROMドライブはXboxゲームディスクに施された物理的なセキュリティ対策のため、オリジナルのXboxゲームディスクを読み取れない。第二に、PC DVD-ROMドライブをXboxマザーボードの独自のDVD電源・トレイ状態コネクタに合わせる必要がある。

最もお手軽だが見た目の悪い方法は、標準PC DVD-ROMドライブを取り付けながら、もとのXbox DVD-ROMドライブを独自ケーブルに接続したまま残す方法だ。この方法では、グレーのIDEケーブルを標準PC DVD-ROMドライブ（ドライブ上のジャンパ設定でスレーブモードに設定）に接続し、ハードドライブの電源コネクタから標準の電源スプリッターケーブルを使って電力を分岐する。Xbox DVD-ROMドライブはそのまま置き、IDEコネクタは外し、独自の黄色い電源・トレイ状態ケーブルのみ接続した状態にする。このXbox DVD-ROMドライブの役割は、DVDドライブのトレイ状態をXboxに手動で伝えるダミードライブとして機能することだ。つまりメディア交換時に、ユーザーはXbox DVD-ROMのトレイを使って標準PC DVD-ROMのトレイ状態を手動で複製する必要がある。

この構成でXboxを操作する正確な手順は使用するPC DVD-ROMドライブのモデルやXboxのハードウェア構成の細かい違いによって異なるため、最新情報についてはウェブ検索を勧める。特定のPC DVD-ROMドライブをXboxの独自トレイ状態・電源コネクタに対応させる方法を解説したウェブサイトも存在する。このようなプロジェクトははんだ付けとドライバーに慣れたハッカーにとって適切な中級レベルの作業だ。標準DVD-ROMドライブに施す改造によって標準ドライブのDVDトレイ状態をXboxに正確に伝えられるようになる。ただし、DVD-ROMドライブのセキュリティチェックを回避する追加ハードウェアでXboxが改造されていない限り、オリジナルゲームはプレイできない。それでも、Xbox-Linuxのインストール問題の切り分けや、CDコレクションのリッピン

グ、DVD の視聴には有用な手法だ。(なお、IDE コネクタを Xbox DVD-ROM ドライブに戻し接続すれば、元のゲーム機能が復元される。)

Xbox のハードドライブも時に交換が必要になる。Xbox のソフトウェアを本格的に開発しているなら、より大容量のハードドライブを Xbox に搭載すると便利だし、ハードドライブが壊れたユーザーも交換を望む。残念ながら、OEM 版の Xbox ハードドライブには Microsoft の著作権付きプログラムが含まれている。また Xbox のハードドライブはファームウェアロックアウトによって保護されており、オリジナルのゲーム機能を維持したまま新しいハードドライブを取り付けることは、特に法的な問題もあって相当な難易度だ。ファームウェアロックアウトはハードドライブごとに固有なため、使用済みの Xbox 用ハードドライブとの交換もできない。ただし、Xbox-Linux やその他のホームブリューソフトウェアを動かすことだけが目的でゲームにこだわらないなら、Xbox への新しいハードドライブの取り付けは通常の PC へのハードドライブ取り付けと同様に簡単で、法的問題もない。

注

1. チップ上の回路は通常、「ボンドパッド」と呼ばれる金属の正方形で囲まれ、そこからチップパッケージのピンに配線される。ボンドパッドの輪に必要な面積がチップ内部の回路に必要な面積を超える場合、そのチップは「ボンドパッド制約」を持つと言われる。ボンドパッド制約がある場合、余剰ピンのコストはさらに高くなる。

第 11 章 Xbox 向けソフトウェア開発

本書の焦点はハードウェアハッキングとセキュリティについて読者を教育することだが、Xbox ハッキングの最終的なゴールのひとつはホームブリューソフトウェアを動かすことだ。本章では、執筆時点で Xbox 向けに進行中のホームブリューソフトウェアプロジェクトを紹介する。

Xbox-Linux

Xbox-Linux プロジェクトのゴールは、GNU/Linux および GNU/Linux アプリケーションのユーザーフレンドリーかつ合法的な Xbox ハードウェアへの移植を実現することだ。世界中のハッカーたちの献身と貢献のおかげで、Xbox-Linux プロジェクトはそのゴール達成に向けて大きな成果を上げてきた。Xbox-Linux プロジェクトのコアチームの写真は図 11-1 に示す。また本章と第 9 章のコラムには Xbox-Linux プロジェクトのチームメンバーへのインタビューが収録されている。(Xbox-Linux プロジェクトのホームページは <http://xbox-linux.sourceforge.net> だ。)

重要な点として、Xbox-Linux プロジェクトとそのコアハッカーたちはアンチ Microsoft ではない。「いじる自由」を支持する人たちであり、幼稚な Microsoft 嫌いではない。テクノロジーを今日の姿にまで育てた思想と言論の自由を守ることにつながるアジェンダを持っている。

Xbox-Linux は Xbox にとっての究極のソフトウェアプロジェクトではない——それどころか、Xbox ソフトウェアハッキングのほんの始まりにすぎない。なじみ深い GNU/Linux 開発環境を Xbox に移植することで、より多くのソフトウェアハッカーが Xbox ハッキングプロジェクトに参加できるようになる。GNU/Linux によって、Xbox はフリーのオープンソースビデオゲームからワープロアプリケーション、Beowulf 型コンピュータクラスター構築用のクラスタリングソフトウェアまで、多様なアプリケーションを実行できるようになる。

Xbox-Linux のインストール

現時点で Xbox-Linux を動かすには、オルタネートファームウェアデバイスを使って GNU/Linux ブート用の ROM をインストールする必要がある。そのためには Xbox を開ける必要がある。LPC インターフェース経由でオルタネートファームウェアデバイスを構築・インストールする方法については第 10 章で説明している。現在、インストールが簡単な LPC インターフェース対応オルタネートファームウェアデバイスを販売しているベンダーがいくつかある。特に注目すべきは、Xodus/Matrix デバイスだ——市場初の完全はんだ不要インストール手順を実現したオルタネートファームウェアデバイスで、インストールに必要なツールはすべて第 1 章「保証を無効にする」に記載されており、Xodus/Matrix デバイス自体にも使いやすい説明書が付属している。

[!NOTE] Microsoft はマザーボードのレイアウトやセキュリティシステムを随時改訂する可能性がある。購入前に、デバイスベンダーに自分のシステムハードウェアとの互換性を確認すること。また、Xbox で標準のキーボードとマウスを使いたい場合は、Xbox ゲームポートを USB に変換するケーブルが必要だ。これは Lik-Sang (<http://www.lik-sang.com>) などのアフターマーケット小売業者で購入できるほか、第 4 章のステップバイステップガイドに従って自作することもできる。

オルタネートファームウェアデバイスをインストールする前に、GNU/Linux カーネルを起動できる ROM イメージを書き込む必要がある。「Cromwell」は Xbox 向けのオープンソース・クリーンルーム（つまり Microsoft のコードを一切含まない）ブート ROM で、GNU/Linux をブートできる。重要な点として、Cromwell のソースコードとバイナリイメージに含まれる情報は、Xbox のネイティブな著作権保護機構を回避するためには使用できない。言い換えれば、Cromwell がいかなる著作権保護回避ツールであるかを主張することは難しい。（Cromwell は、Sourceforge.net サーバー上の Xbox-Linux ウェブサイト <http://xbox-linux.sourceforge.net> からダウンロードできる。）

Cromwell の ROM をオルタネートファームウェアデバイスに書き込み Xbox にインストールしたら、次は GNU/Linux インストールイメージを CD/RW メディアに焼く必要がある。このイメージは Xbox-Linux ウェブサイト (<http://xbox-linux.sourceforge.net>) からダウンロードできる。bzip2 で圧縮された 100MB 超の ISO イメージで、Xbox 上でユーザーフレンドリーな GNU/Linux ディストリビューションを立ち上げるために必要なソフトウェア、インターフェース、ツールがすべて含まれている。この ISO イメージを焼く際は、CD バーナーソフトのイメージ書き込みオプションを使うこと。ISO イメージを CD 上に単独の大きなファイルとしてコピーしてはいけない。（ISO イメージは CD のビッ

トパターンそのものなので、完全なファイルシステムの記述を含んでいる。ISO イメージをイメージとしてではなく通常のファイルとして焼くと、ISO イメージが新しいファイルシステムの中に封じ込められてしまい、ファイルシステムとしてではなく「ビットの袋」として見えるだけになる。)

Xbox-Linux をブートするためのブートプログラムを焼いた 2 枚目のディスクも必要になる場合がある。このブートプログラムは、GNU/Linux のメインインストールイメージをダウンロードしたのと同じ場所から入手できる小さな ISO イメージとして提供されている。このブートイメージを使えば、ゲームを始めるのと同じように Xbox に入れるだけで Linux インストールをブートできる。(別のブートディスクを使いたくない場合は、サードパーティのダッシュボードを使ってこのディスクの内容をハードドライブにコピーし、ハードドライブから直接 Xbox-Linux をブートすることもできる。)

CD/RW イメージをうまく焼くことは、Xbox-Linux インストールで最も難しい部分のひとつかもしれない。Xbox 内蔵の DVD-ROM ドライブのレーザーは書き込み可能な CD メディアの読み取りに向いておらず、CD/RW イメージを作成するためのメディアの種類、バーナーの種類、バーナーの設定について非常に気難しい。さらに、レーザーの劣化具合は Xbox ごと、搭載されているドライブのモデルによって異なる。CD-R メディアを安定して読める Xbox は少ないとユーザーたちは報告しており、CD/RW メディアを使う必要がある。また、ファイルシステムをリセットするだけで実際にはデータを消去しない「クイックイレース」ではなく、未使用の新品 CD/RW か完全消去済みの CD/RW を使って、バーナーの設定を最低速に設定して焼くといい。

特定の CD/RW メディアを選ぶ前に、Xbox の通常のダッシュボードの WMA リッピングツールを使って、音楽を焼いた CD/RW の内容をハードドライブにコピーできるか試してみよう。それがエラーなく安定して動作するなら、その CD/RW メディアは Linux のインストールにも使えそうだ。(Xbox-Linux のインストール問題の多くは、CD/RW ドライブからのデータ読み取り問題に起因していることが判明している。) 執筆時点では、プレスされた CD-ROM メディアで提供されているディストリビューションはない。Xbox-Linux コミュニティではカスタムの CD-ROM イメージを注文する話も出ており、そうすればユーザーを悩ませてくる CD/RW 関連の問題が解決されるだろう。(なお、前章で説明したとおり、書き込み可能な CD フォーマットとの互換性が高いアフターマーケットの DVD-ROM ドライブを Xbox に取り付けることも可能だ。)

[!NOTE] Xbox-Linux は活発に開発が進んでいるプロジェクトで、常に進化し続けている。Xbox に GNU/Linux をインストールする最新の手順は Sourceforge の Xbox-Linux ウェブサイトで見つかる。この手順は執筆時点で少なくとも 6 か国語に翻訳されている。Xbox-Linux プロジェクトにスキルで貢献することに興味があるなら、Sourceforge の Xbox-Linux ウェブサイトに ToDo リストがあり、開発者のメーリングリストへの参加方法も記載されている。

コラム：プロフィール — Michael Steil

Michael さん、自己紹介をお願いします。

1979年にドイツのErding生まれ。現在、ミュンヘン工科大学でコンピュータサイエンスを学ぶ学生です。1学期生にアセンブリを教えており、来年は修士号を取得する予定です。10歳からコンピュータに関わってきて、最初のコンピュータはCommodore 64、その後すぐ386 PCに移りました。常にハードウェアとオペレーティングシステムに強い興味を持ち、ハードウェアアーキテクチャの多様性（Commodore、PC、Amiga、Macintoshなど）や、ゲームコンソールのような組み込みシステムに特に魅了されてきました。（「SEGA CD」にはCPUが3つ——Z80が1つとM68000が2つ——あることをご存じですか？）だから任天堂SNES、SEGA Genesis、任天堂ゲームボーイなど、多くのビデオゲームシステムを実験目的で購入しました。SEGA DrivecastのLinuxも見ましたが、Sony PlayStation 2向けのLinuxは見たことがない——セット全体が僕には高すぎて、実験にも実用にも手が出ませんでした。

Xbox ハッキング、特に Xbox-Linux プロジェクトに関わるようになったきっかけは？

2002年4月30日にXboxを購入しました。ハッキングに最適で、Linuxも動くはずだと確信して。システムソフトウェアを1〜2時間見た後（ゲームは一本も買いませんでした）、Xboxを分解しました。ハッキングの情報を探してみたら最初は失望——ハードドライブをPCに接続する方法と、ほとんど情報のないXbox Linuxのサイトしか見つかりませんでした。それで自分でXbox ハッキングサイトを立ち上げて、ハードドライブをPCに接続して調べた情報を公開することにしました。

Xboxhacker.net とオリジナルのXbox Linux メーリングリストには大いに助けられました。どちらも優秀なハッカーを集め、貴重な情報を発信していました。Xbox Linux Project のもとのインフラに不満を感じて、5月23日にSourceforgeに移行することにしました。これで全コントリビューターがメンテナーを通さずに自由にウェブサイトに追加できるようになりました。とはいえ当時はまだすべてが理論上の話——モッドチップなしではブートローダーを完成させる以上のことはほとんどできませんでした。Andy Green の Filtror がすべてを加速させました。このMODのおかげでブートローダーを完成させ、Milosch Meriac の協力ですべてに短期間でLinux カーネルを適合させられました。

6月に「匿名の寄付者」から連絡があったことは、プロジェクトの知名度が上がりさらに多くのコントリビューターが集まっただけでなく、個人的な友人関係にもつながりました。BioXX（OpenXbox）モッドチップの作者Walter Meyerは、なんと自宅から20キロの距離に住んでいました。彼には特にボックスのMODで大いに助けてもらいました——僕自身はあまりはんだごてを使う人間じゃないので。

2002年12月、LinuxがXbox上で動くようになってから、Xbox Linux コアチーム（Andy Green、Milosch Meriac、Franz Lehner、そして僕。Edgar Hucekは残念ながら来られなかった）がベルリンのChaos Computer Club Congressで初めて対面で集まりました。

僕のそもそもの動機は純粋に「楽しいから」「やることで多くを学べるから」でした。Microsoftに損害を与えたからではありません——ただ、Microsoftが購入したハードウェアで使いたいソフトウェアを使えなくすることで顧客を傷つけているとは思っており、だからこそXbox Linux Projectは特別な意義があると思っています。

僕たちは「アンチMS」でも「MSヘイター」でもありません。彼らの市場戦略が嫌いだけで、だから彼らに対抗する合理的な理由がある、ということです。

Xboxのための200,000ドル賞金についてもっと教えてもらえますか？

この賞金が期待したものではなかったと思います。締め切りから1か月が経っても賞金はまだ配られておらず、いつ賞金がもらえるか聞いてきた人は一人もいませんでした。賞金がプレスを引き付け、知名度が上がり、より多くのハッカーが集まりました。でも誰もお金のためにやったわけじゃない。だから、Michael Robertson に報酬をもらって仕事をしていると見られたくありません。締め切り後も僕たちが全員活動を続けているのがその証拠です。

「MIST X-Code ハック」について教えてください。

bunnie のオリジナルハックからしばらくして、Andy が MCPX ROM を完全に抽出しました。Steve、Paul、そして僕でコードを解析し始め、僕がその中に含まれている X-Code インタープリターをリバースエンジニアリングしました。X-Code の解釈ループを脱出するために使えるバグを探していたとき、コードの一部がすでに僕たちの攻撃を意識して書かれていることに気づきました。以下が僕の元のディスアSEMBルです：

```
cmp ebx, 80000880 ; ISA Bridge, MCPX disable?
jnz short not_mcp_x_disable
; BUG: too specific: bits 24 to 30
; undefined and ignored by PCI hardware!
and ecx, not 2 ; clear bit 1 (MCPX ROM will be
; turned off by setting bit 1)
not_mcp_x_disable:
mov eax, ebx
mov dx, 0CF8h
out dx, eax ; PCI configuration address
add dl, 4
mov eax, ecx
out dx, eax ; PCI configuration data
jmp short next_instruction
```

以前「PCI configuration」を扱っていたので、この攻撃チェックが過度に具体的すぎるのがわかりました。似たようなコードでも同じ効果を持つけれど、このチェックをすり抜けられるんです。MS 開発者は良いアイデアを持っていたけれど、実装が間違っていて、それによって彼らのアイデアを逆に教えてくれました。

Andy、Steve、Paul にアイデアを伝え、短時間後に 0x88000880 が 0x80000880 と同じように MCPX ROM をオフにし、インタープリターコードをメモリからマッピングアウトすることでインタープリターを終了できることを確認してくれました。

「プロジェクト B」

Xbox-Linux の開発者たちが「プロジェクト B」と呼ぶ、ハードウェア改造なしに Xbox-Linux をインストール・起動する方法を見つけるための取り組みが進行中だ。プロジェクト B という名称は、Lindows 社の CEO、Michael Robertson が提供する 200,000 ドルの賞金の授与基準から来ている。「プロジェクト A」の賞金 10 万ドルは、ハードウェア改造で Xbox 上で Linux を動かした最初のグループに授与された。残りの 10 万ドルはプロジェクト B を達成した個人またはグループに授与される。賞金の非対称な配分がプロジェクト B 達成の難しさを示唆している。(プロジェクト B の詳細は Sourceforge の Xbox-Linux ウェブサイト <http://xbox-linux.sourceforge.net/articles.php?aid=2002354043211> にある。)

プロジェクト B に向けていくつかのアプローチが様々なグループによって追求されている。最も概念的にシンプルなアプローチは、Xbox ゲームディスクの署名に使われている 2048 ビットの RSA 鍵を素因数分解することだ。このアプローチは OperationProjectX (<http://sourceforge.net/projects/opx>) が分散コンピューティング方式で追求している。簡単に言うと、2048 ビットの RSA 鍵を素因数分解して Microsoft の秘密鍵を明らかにできれば、Microsoft が通常の CD や CD/RW メディアからプログラムをロードする機能を Xbox カーネルから削除しない限り、誰でも Microsoft のデジタル署名を偽造して Xbox 向けの起動可能なゲームディスクを作れる。重要な点として、Microsoft は特殊なセキュリティ構造を持つ 2 層 DVD-9 フォーマットのディスクでゲームを出荷している。Xbox のファームウェアは、デジタル署名チェックに関わらず、この特定の構造を持つディスクからのみ起動するように設定できる。現在の一般的な DVD バーナーでは 2 層 DVD を焼くことができないため、セキュリティ付き DVD-9 メディアのみを実行可能ファイルのソースとして要求すれば、インターネットからの無償ダウンロードによる Xbox-Linux 配布に障壁が生じる。またこのアプローチの別の問題は、ブルートフォースサーチで Xbox の秘密鍵を素因数分解できる可能性が非常に低いことだ。(第 7 章「セキュリティ入門」の囲み記事「非常に難しい問題」がこのタスクの計算難易度を伝えようとしている。) もしこのアプローチで秘密鍵が合理的な時間内に復元されれば、RSA アルゴリズムへの信頼は大きく損なわれるだろう。(とはいえ、宝くじは買わないと当たらない。CPU のアイドルサイクルを使って無料の分散素因数分解クライアントを動かすことは、Powerball のチケットよりずっと安上がりだ。)

RSA-2048 ビット鍵の解読に関連した別のアプローチは、既存の署名済み Xbox 実行可能ファイルを、その暗号化ハッシュ値を変えずに有用な形で変更することだ。このような「構成的ハッシュ衝突」ができれば、改変された実行可能ファイルはデジタル署名チェックの観点からオリジナルと同一に見える。Xbox のデジタル署名アルゴリズムで使われているハッシュ関数は SHA-1 だ。SHA-1 は公知のアルゴリズム的弱点のない 160 ビットハッシュで、ハッシュのソースが固定されているため、衝突を見つけるにはランダムなバリエーションを約 2160 回試す必要がある。補足として、バースデー攻撃で難易度を 280 回のランダムなバリエーションに下げることができない——同じ任意の値にハッシュされる 2 つのメッセージを見つけようとしているのではなく、特定のターゲットハッシュ(またはすべての公開 Xbox ゲームタイトルから収集した非常に限られたターゲットハッシュのセット)を生成しようとしているからだ。したがってこのアプローチも「非常に難しい問題」の範疇に入る。

contains a sidebar on “Very Difficult Problems” that attempts to communicate the computational difficulty of this task.) If the private key is successfully recovered within a reasonable amount of time by this approach, it will significantly reduce people’s confidence in the RSA algorithm. (On the other hand, you can never win the lottery if you don’t buy a ticket, and running a free distributed factoring client using the spare cycles on your CPU is much cheaper than a Powerball ticket.)

Another approach, related to cracking the RSA-2048 bit key, is to modify an existing, signed Xbox executable in a useful manner without changing its cryptographic hash value. Such a constructive hash collision would make the modified executable look identical to the original as far as the digital signature check is concerned. The hash used in the Xbox’s digital signature algorithm is SHA-1. SHA-1 is a 160-bit hash with no publicly known algorithmic weaknesses; since the source of the hash is fixed, about 2^{160} random variations would have to be tried to discover a collision. As a side note, you can’t use a birthday attack to reduce the difficulty of the attack to 2^{80} random variations because we are not trying to find two messages that hash to the same arbitrary value. The goal is to generate a specific target hash, or perhaps one of a very limited set of target hashes harvested from the set of all published Xbox game titles. Hence, this approach also falls into the category of “Very Difficult Problems.”



Figure 11-1: The Xbox-Linux core team at the 19th annual Chaos Computer Conference, held in Berlin, Germany. In the back, Michael Steil; in the front, from left to right: Andy Green, Milosch Meriac, and Franz Lehner. (Photograph courtesy of Gerhard Farfeleder.)

図 11-1: ドイツ・ベルリンで開催された第 19 回 Chaos Computer Conference に集まった Xbox-Linux コアチーム。後列が Michael Steil、前列左から Andy Green、Milosch Meriac、Franz Lehner。(写真提供: Gerhard Farfeleder。)

プロジェクト B への別のアプローチは、Xbox のソフトウェアにセキュリティホールを見つけ、そのホールを使って CPU の命令ポインタを掌握することだ。これがどう役立つのか、次の例で考えてみよう。ネットワーク経由のバッファオーバーランエクスプロイトがあるゲームで発見され、任意のコード実行につながる可能性があるとする。Xbox とネットワークでつながった PC 上で動作するプログラムがこのエクスプロイトを使って Xbox にパケットを送り、Xbox-Linux 向けのシンプルなブートローダーをインストールできる。このブートローダーは Xbox のハードドライブや DVD ドライブの指定された場所のコードを実行するプログラム程度でいい。Xbox がデータを Accept できるすべてのポート——USB やネットワークポート、ハードドライブ、DVD-ROM ドライブを含む——がこの種の攻撃のベクターになりうる。ハードドライブや DVD-ROM ドライブにイメージングされた壊れたセーブデータやファイル構造が Xbox にユーザー開発のコードを実行させることもある。Microsoft の功績として、すべてのネットワーク通信とセーブゲームのプロトコルはかなり強力によく検証されたセキュリティ技術を使っている。また MIT での Microsoft による Xbox に関する発表で、すべてのゲームコードがバッファオーバーランチェッカーで検査されており、ゲームコードに意図的なバックドアを仕込んだことが判明したゲーム開発者には Microsoft が契約上の是正措置を取ると聞いた。これは Xbox のコードベースが典型的な Microsoft の製品よりも安全であることを示しており、それがハッカーにとって一層興味深い問題として映る理由だ。

📦 コラム：プロフィール — Milosch Meriac

自己紹介をお願いします。

経歴はかなりシンプルです。1976 年にチェコスロバキアで生まれました。教師の母と土木技師の父は、共産主義政権による弾圧を逃れて冷戦中に西ドイツに逃れました。ドイツに来たとき僕は 3 歳でした。ドイツの幼稚園でドイツ語をすぐに覚えました。そこからは順風満帆——10 歳のときに数か月ねだり続けて最初のコンピュータを手に入れました。そこからすべてが始まりました。

高校を卒業してドイツ連邦軍の兵役を経て、サイバネティクスとコンピュータサイエンスを学び始めましたが、3 年後に大学を辞めて自分の会社を長期目標として集中することにしました。在学中にビジネスコネクションを築いておいたので、ドイツの様々な企業でフリーランサーとして働くのは容易でした。リバースエンジニアリングプロジェクト、小さなフットプリントのリアルタイム組み込み Linux システムの開発、Windows システムのリアルタイム Extension などのローレベルプログラミング、ドイツの有名企業向けのソフトウェアベースのハードディスクセーフガードの開発など。今はベルリンで彼女と楽しく暮らしています。

なぜハッキングするのですか？

プログラミングの経験を積むうちに、コンピュータの世界の美しく輝かしいと思っていたものが、実は脆いパッチワークであることに気づき始めました。

最初、ハッキングはゲームのようなものでした。コンピュータシステムの中を歩き回り、毎日新しいコードと可能性の世界を発見できました。たまにアプリケーション作者との決闘も楽しんで、コピープロテクションを解析・回避しようとしていました。チェスのような時もあるれば、デスマッチのような時もありました。

自分の知識が成長していくのを感じ、同時に14歳の子どもにとっては高給取りの天才ハードコアプログラマーたちのセキュリティシステムを回避できることは当然大きな自信になりました。高校の頃にこの見方を改めました——学校の休暇中にいくつかの地元企業でプログラミングツールやアプリケーションを開発しながら、本物のプログラマーたちに出会ったのです——そして失望しました。彼らは神でも神のような存在でもありませんでした。

しばらくして、クールなデモを作ったり、アプリケーションXをハックしたり、Yにニッティなハックをしたりしてもどこかのサック・オブ・ライスを倒すよりも世界を変えられないことに気づきました。そこでもっと賢く対象を選び始めました——電話、コンピュータ、ネットワーク、衛星といった日常生活のテクノロジーを。平均的なユーザーにテクノロジーを説明したり、企業が製品を安全にするのを助けたりすることで、物事を変える力があることを発見しました。

今、ホワイトハットハッカーとしての自分の力を自覚しています。今日の誰もがITの影響を受けている——監視技術、データマイニング、情報戦争、デジタルミレニアム著作権法(DMCA)、TCPA、デジタル著作権管理、著作権と特許法の新解釈がモンスーン後のキノコのように増殖しています。かつてのように僕はこれらの美しく輝かしいものの裏を覗きたい——そして彼らが僕たちを見つける前にバグと罠を見つけたいと思っています。

Xbox-Linux プロジェクトでの経験を教えてください。

Xbox Linux プロジェクトに参加し、カーネルの動作を実現するために貢献しました。Xbox アーキテクチャが通常のPCと比べていくつかの落とし穴や違いがあるため、これは容易ではありませんでした。Microsoft Xbox の Linux ディストリビューションの初期版を作りました。フラッシュメモリは1MBしかなく、ハードドライブはまだアンロックされていなかったもので、完全なディストリビューションとカーネルをそこに収める必要があって、これは重要な取り組みでした。またAndy GreenのFiltrorデバイス用のコンソールドライバーも提供しました。これで彼のデバイスをリモートインターフェースとして使ってカーネルの起動メッセージを見てLinuxコンソールを得られるようになりました。このディストリビューションにはネットワークドライバ、サウンドカードドライバ、MP3サポート、TelnetサーバーやWebサーバー、NFSサポート、そして広範な標準Linuxツールがすでに含まれていました。これによってカスタムハードウェアを手放し、何百人もの人がコードコントリビューターかテスターとしてプロジェクトに参加できるようになりました。まだスクリーン出力がなかったので、Xbox Linux カーネルにフレームバッファインターフェースを追加し、他にも多くの貢献をしました。

貢献する開発者の数が急速に増え始めました。世界中から素晴らしい協力をいただいてXbox Linuxを実現できています。DMCAなどの米国の法的不確実性を恐れて姿を隠している人もいれば、自由に貢献できる人もいます。

その他にコメントはありますか？

なぜ僕のような大人がこの Xbox というおもちゃに関わっているのか不思議に思う人もいるでしょう。もちろん人によって理由は違うし、僕の理由はスキルを向上させてより新しいテクノロジーについて学ぶことです。Microsoft Xbox は、たとえば技術的・社会的含意を持つ TCPA/Palladium 保護コンピュータの前身です。ユーザーへの制約なしにより安全なコンピュータシステムの研究をするための素晴らしい遊び場です。

最大の理由のひとつはコミュニティです。この輝かしいオタクたちと一緒に作業するのは本当に楽しく、大きな喜びです——オンラインでも、特に上質なビールのパブでオフラインで集まるときも。コミュニティが日々強くなっていることに毎日驚かされます。これを可能にしてくれたすべての人に感謝！

最近、Electronic Arts の「007: Agent Under Fire」ゲームのセーブゲーム処理方法にバッファオーバーランエクスプロイトが発見された。このエクスプロイトは 2003 年 3 月 29 日に「habibi_xbox」というハッカーによって XboxHacker.net BBS への投稿で最初に明かされた。重要な点として、このエクスプロイトは未公開の数のゲームで確認されたが、「007: Agent Under Fire」が明示的に名指しされた唯一のゲームだった。このエクスプロイトはチェックされていない文字列を利用して、一連のカーネルパッチを挿入する短いセグメント（数百バイト）のコードを実行する。このハックには、意図された Xbox-Linux ターゲット以外のことに使いにくくする様々な対策が含まれていた。たとえばこのハックは、デジタル署名の検証に使われるオリジナルの Xbox RSA 公開鍵を新しい公開鍵に差し替えながら、デジタル署名チェックアルゴリズム自体はパッチを当てないままにする。ハックの一部として提供される Xbox-Linux ブートローダーだけが、対応する新しい秘密鍵で適切に署名されている。他のハッカーがこのハックを使って他の実行可能ファイルを動かすには、新しい公開鍵を素因数分解する必要がある。また「007: Agent Under Fire」ゲーム自体がすべてのセーブゲームに独自のデジタル署名チェックを行うため、ハックされたセーブゲームファイル内のエクスプロイトコードの変更も簡単ではない。このようなセキュリティ対策をハックに含めるのは称賛に値する決断で、Microsoft の利益を守ることで Xbox-Linux プロジェクトを守るためのものだ。

将来を見据えると、プロジェクト B の成功は Xbox ハッキングの新時代を開くか、あるいは Xbox ハッキングの終焉を意味するかのどちらかかもしれない。プロジェクト B のハッカーたちは Microsoft の利益を守ろうと社会的な良識と誠意を示してきたが、悪意あるハッカーがそのハックをリバースエンジニアリングして最終的にそのテクニックを Microsoft に友好的でない形で再現する方法を見つけることを防ぐことは不可能だ。最終結果は、Microsoft によるすべてのハッキング活動への厳しい取り締まり、あるいは Dreamcast の著作権侵害の惨事で Sega が撤退したように収益が断たれてビデオゲームビジネスから Microsoft が撤退するか、のどちらかになりうる。あるいは、Microsoft はビジネスにさらにお金を注ぎ込み、既知のセキュリティホールへのパッチと対策を組み込んだ再設計されたコンソールをリリースすることも選べる。結果は今後数か月の展開に大きく依存する。ただし、Xbox の深い値下げが見えており、大幅に再設計された「シュリンク」バージョンのコンソールの噂が出回っている今、Microsoft の近期戦略は、フェアユースや著作権侵害を食い止めることよりも市場を席巻することにエネルギーを集中しているように見える。どのみち、PlayStation 2 や GameCube が 1 台売れることのほ

うが、GNU/Linux を動かすために Xbox が 1 台改造されること——さらには著作権侵害ゲームのために Xbox が改造されること——よりも Microsoft のビジネスへの悪影響が大きいだろう。

OpenXDK

XboxMediaPlayer や MAME-X (Xbox 向け Multiple Arcade Machine Emulator) など、Xbox ネイティブのゲームプラットフォーム向けに多くの興味深く有用なプロジェクトが開発されてきた。残念ながら、これらのプログラムは Microsoft Xbox SDK の非公認版を使って開発された。Microsoft の Xbox SDK は承認を受けたライセンス取得済みの開発者だけに提供されることになっているが、コンソールが発売される前に SDK が流出し、それ以来多くの人が流出した Xbox SDK を使って独自の Xbox プログラムを作ってきた。プロプライエタリな Xbox SDK は便利で使いやすいが、使用は技術的に違法でもある。ネイティブ Xbox プラットフォーム向けの合法的な SDK がないことで、オープンソース開発者の大規模なベースを引き付けることが難しくなっている。

OpenXDK プロジェクトは Xbox SDK に対する合法的な代替手段の必要性に応えるために作られた。OpenXDK の表明されたゴールは、Xbox 実行可能ファイル (XBE) を作成するための合法的な開発キットを作ることだ。OpenXDK によって、適切なデジタル署名を付与することでバニラな Xbox で動作できるネイティブ XBE ファイルを作成できるようになる。その適切なデジタル署名が現時点ではまだ不明なため、この作業は OpenXDK を使って開発されたプログラムとの相互運用性を可能にする合法的な技術を見越して行われている。

その有用性にもかかわらず、OpenXDK プロジェクトはまだ初期段階で開発者を求めている。OpenXDK プロジェクトの詳細は <http://openxdk.sourceforge.net> にある。OpenXDK のプロジェクトマネージャーは Dan Johnson (SiliconIce と呼ばれる、XboxHacker BBS の作者) と Aaron Robinson (caustic と呼ばれる。caustic は CXBX 実行可能ファイルリンカーと CXBE Xbox エミュレータプロジェクトも率いている) だ。

第 12 章 注意せよ、ハッカーよ

リバースエンジニアリングと知的財産法の間には、いくつかの厄介な法的絡み合いがある。一方では、イノベーションにはそれにふさわしい報酬が必要だ。発明者や作者が自らの労働の成果を独占的に生産・販売する権利は保護されなければならない。他方、自由で競争的な市場もイノベーションを守り公正な市場を確保するために必要だ。既存の製品に具現化された設計原理の研究と、改善された派生品を生産する能力は、競争市場の重要な一部だ。

本章では知的財産法の概観と、ハッカーとして知っておくべき重要な点をいくつか紹介する。無知は有効な弁護にならない。リバースエンジニアリングと知的財産権を規定する法律を無視した者には厳しいペナルティが規定されており、知的財産権侵害の一部の行為は重罪として多額の罰金とともに罰せられる。

本章の大部分は Electronic Frontier Foundation のシニアスタッフアトニーである Lee Tien が執筆した。Lee（と Joseph Liu）は僕が Xbox セキュリティシステムに関する調査結果を発表しようとしていた時期に僕の顧問弁護士だった。第8章には「ハッキングの法的課題」というタイトルのコラムがあり、論文発表をめぐる MIT との戦いを説明している。

本章の内容はハッカーのための情報リソースを提供することを目的として提示されている。もし法的に問題のある状況にあると思われる場合は、弁護士に連絡して自分の具体的な状況について適切な法的アドバイスを得ることに代わるものはない。

コラム：プロフィール — Lee Tien

Lee Tien はフリースピーチ法を専門とする Electronic Frontier Foundation のシニアスタッフアトニーである Lee Tien は、知的財産法やプライバシー法との交差点も専門とする。EFF に入る前は Freedom of Information Act (FOIA) 訴訟を専門とする個人弁護士として活動していた。Tien 氏は児童の性的表現と情報技術、匿名性、監視、コンピュータソフトウェア発表の合衆国憲法修正第一条上の地位に関する論文を発表している。スタンフォード大学で心理学の学士号を取得し、そこでは Stanford Daily でジャーナリズムに積極的に関わった。Tacoma News Tribune で1年ニュースレポーターとして働いた後、カリフォルニア大学バークレー校の Boalt Hall で法律を学んだ。また UC バークレーの Program in Jurisprudence and Social Policy で大学院の課程も修了した。1

コラム：Electronic Frontier Foundation

Electronic Frontier Foundation (EFF) は Xbox セキュリティシステムに関する論文を発表しようとしていた時期に僕に法的顧問を提供してくれた。以下の段落では EFF が何をしている組織なのかを紹介する。

電子ネットワーク——インターネット——によって今ここに実現された世界がある。友人、見知らぬ人、そして未来の世代と知識、アイデア、思想、ユーモア、音楽、言葉、芸術を共有できる世界だ。将来のテクノロジーの発展によって、情報へのアクセスや他者とのコミュニケーションはさらに強力な形で実現されるだろう。

しかし、中世に権力の座にある者たちが人々に読ませたくない本の製造と流通を管理したり燃やしたりしたように、世界中の政府や企業の利益が、新しいテクノロジーを通じた自由なコミュニケーションを妨げようとしている。だが、書籍、電話、コンピュータなど媒体を問わず、自由に語る権利を守って戦ってこそ、人間の状況を守り高めることができる。

Electronic Frontier Foundation (EFF) は、インターネットや World Wide Web などの新技術を使って考え、語り、アイデア、思想、ニーズを共有する私たちの権利を守るために設立された。EFF はオンラインでの基本的な権利への脅威をいち早く特定し、デジタル時代のフリースピーチのために擁護する最初の組織だ。

サンフランシスコを拠点とする EFF は、テクノロジーに関わらず私たちの基本的権利を守り、テクノロジーに関連する市民的自由問題についてプレス、政策立案者、一般市民を教育し、そうした自由の擁護者として行動する、ドナーサポートによる会員組織だ。EFF の各種活動には、誤った立法への反対、個人の権利を守る訴訟の開始と弁護、グローバルな広報キャンペーンの実施、最先端の提案と論文の発表、頻繁な教育イベントの開催、プレスとの継続的な関わり

り、世界で最もリンクされるウェブサイトのひとつ <http://www.eff.org> でのデジタル市民的自由情報の包括的なアーカイブの公開などが含まれる。2

知的財産法の概観：古典的枠組み

Lee Tien による「注意せよ、ハッカーよ：知的財産入門」

リバースエンジニアリングとは、成果物から知識やノウハウを抽出するプロセスだ。市場では「競合他社の製品が何で動いているかを突き止める昔ながらの手法」と呼ばれてきた。1 しかし今日、大量販売製品を研究する者は誰でも、リバースエンジニアリングを取り巻く法的地雷原を意識しておくべきだ。デジタルミレニアム著作権法 (DMCA) の技術的保護手段回避規定、2 リバースエンジニアリングを禁じる契約条項、経済スパイ法 3 は、技術者が知っておくべき危険な法的分野のいくつかだ。本章ではこれらの分野を概説して、ハッカーたちに問題の大まかな輪郭を伝える。

ここには二つの一般的な問題がある。第一に、リバースエンジニアリングは合法か？ 第二に、リバースエンジニアリングが許可されている場合でも、その過程で学んだことを発表できるか？

知的財産法は伝統的に著作権と特許を意味していた。どちらも、憲法の知的財産条項「議会は……限られた期間、著者と発明者にそれぞれの著作物と発見に対する独占的権利を確保することにより、科学と有用な技術の進歩を促進する権限を有する」4 に基づく連邦法律によって創設・制限されている。コンピュータプログラムは通常「文芸的著作物」として著作権によって保護されるが、特許を取得することもできる。5

近年、営業秘密も知的財産の一種とみなされるようになった。営業秘密はもともと判例法に基づいて裁判所によって保護されてきたが、現在は州法と連邦法の両方の対象となっている。著作権や特許と異なり、営業秘密法は歴史的に不正競争の原則に根ざしている。

米国では、作者や発明者に「自然権」はない。6 代わりに、その権利は公共の福祉という概念に基づいている。他者が自由に使えるなら作者や発明者は創作する十分なインセンティブを持てないため、社会は保護を与えることで恩恵を受ける。しかしその保護は、公衆が最終的に恩恵を受けられるよう制限されている。7 たとえば著作権と特許の権利は「限られた期間」のみで、最終的に保護された著作物はパブリックドメインに入らなければならない。8 要するに知的財産法は、公衆と作者・発明者の間の「取引」の条件を設定するものだ。

著作権

著作権法は有形媒体に「固定」された独創的な表現の著作物を保護し、著作者（または権利の譲受人）に著作物の複製、配布、翻案、公開展示、公開実演に対する独占的権利を与える。独立した創作からは保護しない。

著作物はコピーやフォノレコード（録音物のコピー）とは異なる。本を買っても、コピーを所有するだけで、著作権者は著作物自体への権利を保持する。なお、「初回販売」の原

則により、合法的なコピーの所有者はそのコピーを売却または譲渡できる。9 ただしいくつかの例外がある。10

著作物の種類は多く、それぞれに異なるルールがある。4 著作権法は非常に複雑で、テクノロジーはその複雑さをさらに増した。たとえば著作権の付いた曲を考えよう。楽曲 (MC) は著作権によって保護され、通常は作詞作曲家が権利を持つ。録音するには MC 著作権者の許可が必要だ。11 いったん録音されると、原曲に対するシンガーの解釈、プロデューサーやサウンドエンジニアの努力を含む実際の録音された音を保護する独立した著作権が録音物 (SR) に発生する。レコード会社は通常 SR 著作権を持つ。そのため、著作権付きの録音物を TV コマーシャルで使いたい場合は MC 著作権者と SR 著作権者の双方の許可が必要となる。

著作権者の権利のほとんどは明らかだが、特にコンピュータが関わる場合はそうでないものもある。たとえば、コンピュータはプログラムを RAM にロードして著作権上のコピーを作成する。著作権法にはコンピュータプログラムのコピーの所有者がプログラムをコンピュータメモリにコピーすることを認める特定の免除規定がある。12 これは著作権法の一般的な厳格さを示している——著作物を意図された目的で使用するためにコピーを作る必要があるとしても、そのコピーが著作権侵害でないとは限らない。このインターネットに対する厳格さの意味は深刻だ。インターネットでの普及は一般にコピーの作成を伴うからだ。

翻案 (派生的著作物) に対する権利も混乱を生じさせることがある。翻案 (「派生的著作物」) は著作権付き著作物に基づく著作物だ——外国語翻訳、書籍を原作とした映画など。ある批判の多い事例では、合法的に所有するコピーから絵を切り抜いて陶器のタイルに貼り付けることが侵害的な派生物を作ることに当たるとした裁判所の判断があった。13 ほとんどの裁判所はこの結論に同意していない。14

著作権保護は著作物が作成された時点で自動的に始まり、一般的に著者の生存期間プラス70年続く。15 著作権期間が終了すると著作物はパブリックドメインに入り、誰でも自由に使えるようになる。

著作権には多くの例外がある。著作権が表現を保護するという原則により、著作物で明かされているアイデアや事実の使用は禁止されない。「アイデア」にはストーリーのプロットが含まれる。さらに一般的に、著作権は著作物の機能的側面を保護しないため、他のプログラムと同じことをするコンピュータプログラムを、その表現をコピーしない限り書くことができる。

事実は発見されるものであって作られるものではないため著作権の「外」にあるとみなされる。これには新しい素数の発見なども含まれる。しかし、事実やその他著作権で保護できないものの選択、順序、配列には著作権を持てる。典型的な例はパブリックドメインの詩のアンソロジーだ。個々の作品が保護されていなくても、選択、順序、配列が十分な独創性を持つなら編集物に著作権を持てる。典型的な電話帳の「ホワイトページ」ディレクトリの事実のアルファベット順配列は憲法上の独創性要件を満たさない。電話番号の収集にお金、時間、労力を投資しただけでは保護は得られない。

著作権は多くの「普通の」使用をカバーしない。本を読む行為自体は著作権の対象ではない——著作権者のいかなる権利も侵害しないからだ。シャワーで歌うのは実演だが、著作権者が権利を持つのは公開の実演に対してのみだ。ただしインターネットはここでも状況を変えた。ウェブブラウザで文書を読むとき、あなたのコンピュータはおそらくその文書のコピーを作成している。こうして、かつては普通だった多くの使用が今やコピーの作成を伴い、著作権上の問題を提起する。

今日、「フェアユース」については多くの論争がある。フェアユースは著作権侵害に対する抗弁で、人々がいくらかの無許可使用を行えるようにすることを意図していた。フェアユースは書評者が書籍から引用することを認める。これは非常に複雑な法分野で、使用が「フェア」かどうかは目的、性質、量、使用の経済的影響などの要素に依存する。16

特許

特許法は発明を保護し、発明者（またはその権利の譲受人）に、特許出願日から 20 年間、他者が発明を製造、販売、使用することを排除する権利を与える。著作権と異なり、特許法は他者による独立した発明に対しても保護する。

ここでの取引はこうだ——特許と引き換えに、発明者は当該技術分野の「当業者」が大した実験なしに発明を実現できる十分な情報を特許出願書に記載しなければならない。特許が付与されると出願書は公開される。情報を公開することで、特許権者は社会の知識の蓄積に貢献する。

しかし特許は積極的権利を与えない——他者の発明の改良に特許を取得しても、基礎となる特許を侵害せずにその改良を実施することはできない。新薬を発明して特許を取得しても、薬を販売するには規制当局の承認が必要かもしれない。

特許を取得するためには、発明が有用で、新規であり、当業者にとって「自明でない」ことが必要だ。新規性と非自明性の要件は、発明が排除権を得るためには技術の十分な発展である必要があることを意味する。これらの高い水準を満たさない発展は保護を否定される。

営業秘密

第三の法律分野——営業秘密——も知的財産法の一部とみなされているが、実際には財産ではない。営業秘密とは、ユーザーには知られているが競合他社には知られていない商業的に価値のあるビジネスその他の情報だ。秘密性（絶対的な秘密ではないが）が営業秘密の本質であり、営業秘密を漏洩から守るために合理的な予防措置を取らなければならない。

特許と営業秘密の間には明らかな関係がある。どちらも有用な情報を保護するからだ。有用な情報がそもそも特許を取得できない場合、選択肢はない。しかし特許を取得できる発明でも特許にしたいくない理由があるかもしれない。特許出願に情報を開示したくない場合もある。また、テクノロジーが長期間価値を持つとは思えない場合、20 年続く特許を取得する価値がないかもしれない。

営業秘密の主な欠点は、独立した発明やリバースエンジニアリングから保護できないことだ。したがって製品から秘密を解明できる場合、営業秘密は賢明ではない。一方、秘密が製品の製造に使用されるプロセスである場合は、リバースエンジニアリングが難しいかもしれない。コカ・コーラは何年もマーケットに出回っているが、その配合を複製する方法を誰も解明していないようだ。

憲法上の著作権の取引

知的財産権は目的のための手段——知識とテクノロジーの進歩を促進するためのものだ。最高裁がかつて述べたように「議会が付与できる独占的特権は無制限でも、主として特別な私的利益を提供するために設計されたものでもない」。17

上記の一節は、知的財産法が著作権・特許法によってもたらされる潜在的な独占的権力の制限に長らく関心を持ってきたことを示している。たとえば初回販売の原則は、特許付き製品や著作権付き著作物のコピーが販売された後、特許権者や著作権者が市場を支配することを防いでいる。

また著作権法は、言論の自由との均衡を保つよう裁判所と議会によって長く解釈・立法されてきた。アイデア・表現二分法、フェアユース原則、著作権の限られた期間などの法理は、一般に著作権と表現の自由の間の潜在的な衝突を軽減するものとして評価されている。18

興味深いことに、独占への懸念は歴史的にフリースピーチへの懸念と結びついている。英国の著作権法は長らく国家が後援するカルテルの一種として機能してきた——著作物に対する私的独占と引き換えに、出版業者は政府の検閲——特に聖書やその他の宗教著作物——のために警察の役割を担うことに合意した。19 同様に、著作権法のアイデア・表現二分法は、著作権で保護されない事実やアイデア、特許で保護されない機能的原理が将来の創造者が基盤とするパブリックドメインに残ることを確保する。

リバースエンジニアリングの伝統的な見解

歴史的にリバースエンジニアリングは、大量販売製品に具現化された情報を得るための合法的な方法として常に認められてきた。多くの技術企業にとって、競合他社の製品をリバースエンジニアリングしてイノベーションを研究することは標準的な慣行だ。実際、米国の裁判所もリバースエンジニアリングを知的財産法のバランスを保つための重要な要素として扱ってきており、最高裁はリバースエンジニアリングを「イノベーションの本質的な部分」と呼んでいる。

法律は正当なリバースエンジニアリングの三つの主な目的を認識している。競争的リバースエンジニアリングは直接的な代替物を作ることを目的とする。互換性または相互運用性のリバースエンジニアリングは、リバースエンジニアリングされた製品と連携する製品の作り方を解明することを目指す。そしてもちろん研究者は商業的目的なしに知識を得るためにしばしば製品をリバースエンジニアリングする。

営業秘密と「不正な手段」

一般に、営業秘密は、人物や企業が秘密を合意または秘密保持関係に反して悪用・開示した場合、他の不正な行為（贈賄、強制、不法侵入など）を行って秘密を入手した場合、または秘密を盗んだ者から情報が不正流用された営業秘密であることを知って、あるいは知るべき理由があって取得した場合にのみ、不正流用となる。

カリフォルニアをはじめとする多くの州は、リバースエンジニアリングが営業秘密を取得する合法的な方法であることを明示的に規定している。20 営業秘密法の健全な原則としてリバースエンジニアリングを支持するいくつかの理由がある。公開市場で製品を購入することは一般に購入者に製品に対する人的財産権を与え、それには製品を分解し、測定し、検査などに供する権利が含まれる。法律はまた、公開市場での製品の販売を、製品が具現化するイノベーションの公表とみなし、発明者が特許保護を取得していない限りパブリックドメインへの奉獻とみなす。

営業秘密のリバースエンジニアリングに対する脆弱性は憲法の全体的なスキームの一部だ。Bonito Boats v. Thunder Craft Boats で最高裁は、ボートのメーカーが既存のボート部品を競合製品を生み出す直接成形プロセスの「プラグ」として使うことを禁じたフロリダ州法を、その法律が「パブリックドメインにある製品のリバースエンジニアリングの一形態に公衆全体が従事することを禁止した」として無効とした。21 裁判所はリバースエンジニアリングは「イノベーションの本質的な部分」であり、「テクノロジーの大きな進歩につながりうる」製品のバリエーションをもたらす可能性があると説明した。実際、「リバースエンジニアリングの競争的現実、追加の特許可能なアイデアを開発するための発明者への刺激として機能しうる」。

Bonito Boats のような事例では、州法が連邦法によって「先占」されているかどうかの問題になる。連邦法と州法が直接的または連邦政策の目標として衝突する場合、合衆国憲法の最高法規条項の下で連邦法が一般に州法よりも優先される「衝突」先占の原則により州法は無効となる。22 著作権法にも特定の先占条項があり、後述する。

著作権法と中間コピーの問題

最近まで著作権法はリバースエンジニアリングを心配する必要がなかった。書籍、美術、音楽をリバースエンジニアリングする理由がほとんどなかったからだ。コンピュータプログラムが「文芸的著作物」になった今、状況は大きく変わっている。多くのコンピュータプログラムはオブジェクトコードのみで配布されるため、リバースエンジニアリングプロセスは通常、ソースコードへの最初の逆コンパイルを必要とする——それはコピーの作成を伴う。

米国の裁判所は、リバースエンジニアリングへの付随的なコピーは「フェアユース」になりうるとして、著作権法は必ずしもリバースエンジニアリングを禁止しないと判断している。「著作権法は、著作物のコピーを合法的に所有する個人が著作物のアイデア、プロセス、動作方法を理解するために必要な努力をすることを認めている。」23 これはリバースエンジニアリングの最終的な目的が商業的なものであっても当てはまりうる。裁判所は一般に著作権保護の憲法上の目的、すなわち「科学の進歩の促進」24 に依拠してい

る。フェアユース原則は「著作物によって伝えられたアイデアと情報を自由に基礎として構築することを他者に奨励する」²⁵ ことでこの憲法上の目的を前進させる。

ここでの重要な事例はSega Enterprises Ltd. v. Accolade, Inc.だ。²⁶ AccoladeはSegaゲームプログラムを逆アセンブルして、Sega Genesis ゲームコンソールと互換性のあるゲームを作るために必要な情報を得た。Accoladeはその後、Sega とそのライセンスを受けた開発者が作るゲームと競合する自社のゲームを販売した。Accoladeは逆アセンブルのコピーが侵害にあたるというSegaの主張に対してフェアユースの抗弁を提起した。裁判所は上記の理由でAccoladeの抗弁を認めた。また裁判所は、AccoladeがSegaのコードを逆アセンブルできない場合、Segaはプログラムの「保護されていないアイデアや機能的概念に対する事実上の独占権」を得ることになり、これは特許法の下でのみ利用できるものだと言及した。²⁷

しかし裁判所の判断は、互換性のあるプログラムを作るために必要な機能仕様へのアクセスを得るためなど「正当な理由」のためのリバースエンジニアリングに限定され、かつ「著作権によって保護されていないコードの要素へのアクセスの唯一の手段を提供する場合」に限定された。²⁸

特許法

特許法には一般的なフェアユース抗弁やリバースエンジニアリング免除はない。理論上は、特許された製品をリバースエンジニアリングする必要はないはずだ——特許明細書が発明を作る最良の方法に関連する技術コミュニティに知らせるべきだからだ。

いくつかのリバースエンジニアリング活動は特許を侵害しない。たとえば特許された発明を具現化した機械の購入者は、一般に特許法の初回販売原則の下でその機械を分解してどのように動くか研究することは自由だ。製品を購入することはそれを使用する権利を意味し、単に研究することは特許権者の発明を製造または販売する独占的権利を侵害しない。とはいえ裁判所はリバースエンジニアリングに対する契約上の制限を時に強制する。²⁹

また、科学的好奇心を満たすために特許された発明を作ろうとする者には「実験的使用」の抗弁があるかもしれない。米国法では、この抗弁は狭く、特許された発明の事業への適用を目的とした研究や商業製品の開発につながりうる研究用途を含まない可能性が高い。³⁰

これら三つの分野の衝突は、Segaの状況を再び見ると明らかだ。SegaがすべてのSegaゲームプログラムで使用されているアルゴリズムに特許を持っているとしよう。Segaプログラムを逆アセンブルすることで、Accoladeは偶然であっても特許されたアルゴリズムを「製造」または「使用」したと言えるかもしれない。要するに、特許の文脈でも中間コピーの問題が再浮上する。

リバースエンジニアリングの新たな課題

リバースエンジニアリングの重要性は、大量販売製品における商業暗号の台頭とともにのみ増してきた——なぜなら、システムをより安全にしようとするとなしに、システム

のセキュリティを破ろうとすることは不可能だからだ。皮肉にも、暗号化の使用の拡大がリバースエンジニアリングに反する法律に貢献してきた。たとえばエンターテインメント産業は現在、CD の音楽や DVD の映画などのデジタル情報を不正コピーから守るために暗号化やその他の技術に頼っている。

当然のことながら、暗号化やその他の形式のセキュリティを「回避」することを防ぐための新しい法律が制定されてきた。

リバースエンジニアリングへの法的侵食は暗号化に限定されていない。1970 年代と 80 年代には、一部の州がクローンボートの製造のためにボートの船体をリバースエンジニアリングする直接成形プロセスの使用を禁じた。³¹ 1970 年代後半から 80 年代初頭にかけて、半導体産業はクローンチップを作るためのチップレイアウトのリバースエンジニアリングから保護する立法を求め、取得した。³² 知的財産権に関する主要な国際協定はリバースエンジニアリングについて何も言っていない。³³

デジタルミレニアム著作権法（DMCA）と不正アクセスの問題

DMCA はリバースエンジニアリングを現在規制する最も重要な法律のひとつだ。DMCA の一部である「技術的保護手段回避」規定は、著作権のある著作物へのアクセスを効果的に制御したり、複製を防止したりする技術的手段に法的保護を与える。残念ながら DMCA は非常に複雑で、たとえば「有効な技術的保護手段」という言葉の意味を明確に定めず、そのような手段を迂回することを違法としている。

不正アクセス

DMCA は本質的に著作権者に「アクセス」に対する新しい権利を作り出した。著作権産業の代弁者は技術的保護システムを回避する行為を家への「不法侵入」にたとえる。

簡単な例として、子供が不適切な画像を見るのを防ぐために学校や図書館が使う検閲ソフトウェアプログラムがある。これらのプログラムはしばしば検閲されたウェブサイトの暗号化された「ブラックリスト」を含んでおり、ベンダーは通常これを営業秘密として扱う。特定のプログラムが子供にとって完全に適切なサイトをブロックしていることを発見した研究者が、不当にブロックされている適切なウェブサイトがいくつあるかを調べるためにブラックリストを読みたいとする。³⁴ ベンダーがその内容へのアクセスを防ぐために暗号化し、リストが著作権のある事実のコンパイルである可能性があるため、暗号化は著作権のある著作物に適用された技術的保護手段であり、無許可の復号は違法な回避行為となる——ただし DMCA には現在、検閲ソフトウェアのブラックリスト復号に対する一時的な免除規定がある。

別の例として、映画産業は DVD の映画を保護するために CSS（Content Scrambling System）と呼ばれる暗号化方式を使っている。2600 事件では、³⁵ CSS は映画へのアクセスを「有効に」制御する技術的手段と判断された。著作権者の許可なしに CSS を迂回することは DMCA の下で違法な「回避行為」だ。裁判所は（著作権法に対するのとは対照的に）フェアユース原則が DMCA に適用されるとは判断していないことに注意。した

がって、CSSがDVD映画のコマーシャルを早送りすることを妨げている場合でも——その制限を「回避」することは依然として違法だ。

ここで「有効」は暗号学的な意味での有効性とは関係ないことに注意。DMCAの下では弱い暗号でも「有効」だ——普通の人々がそれを破れないからだ。

回避技術

DMCAは技術的手段を二番目の方法でも保護している。その「反デバイス」規定は回避を可能にする技術の製造と配布を違法とする。36「不法侵入」のたとえを続けると、著作権産業の代弁者は回避技術を多くの州で違法である「泥棒の道具」にたとえる。

DMCAのセクション1201は「いかなる者も、以下の三つの特徴のいずれかを持つ技術、製品、サービス、デバイス、コンポーネント、またはその一部を製造、輸入、公衆への提供、または流通……してはならない」と規定している。(1)「保護の回避を主な目的として設計または生産された」場合、(2)「保護を回避する以外に限定された商業的に重要な目的または用途しかない」場合、または(3)「保護を回避するために使用するとその人物またはその知識を持って代わりに行動する別の人物が宣伝する」場合。

これらの規定は「アクセス制御」という新しい権利だけでなく、著作権者の権利一般にも適用されることに注意。したがって、CDのコピープロテクション手段を回避する技術もこれらの規定の下で違法になりうる。

先の二つの例を思い出してほしい。2600事件では問題となったのはDeCSS——CSSで保護されたDVD映画を復号できるプログラムだ。DeCSSは禁止された回避技術と判断された。検閲ソフトウェアの例では、DMCAの免除規定は復号行為を認めるが、暗号化されたブラックリストを復号するために使ったコンピュータプログラムを利用可能にできるかどうか、あるいは復号方法の詳細すら同様に利用可能にできるかどうかについては何も言っていない。

DMCAの免除規定を利用する

オブジェクトコードを逆コンパイルや逆アセンブルせずにリバースエンジニアリングできないのと同様に、技術的保護手段を回避せずにリバースエンジニアリングはできない。さらに実際にリバースエンジニアリングを実行するには技術的なデバイスやツールが必要が多く、回避技術の禁止はリバースエンジニアリングも制限する。

これらのDMCA規定が組み合わさることで、実際の大量販売製品で使われているセキュリティ手段を分析したい暗号学者やセキュリティ研究者にとって大きな障壁が生まれる。別の企業の技術的手段の問題を発見して改善方法を提案する商業的リバースエンジニアは、DMCAの刑事罪で起訴されるリスクがある。

学術的リバースエンジニアでさえ、企業のセキュリティ手段の弱点に関する論文を発表することで訴訟を起こされるリスクがある——そのような論文は「回避ツール」と見なされうるからだ。37一例として、プリンストン大学のEdward Felten教授がいる。Felten教授は保護されたデジタル音楽のための技術を検討していたSecure Digital Music Initiative (SDMI) が開催した「SDMI チャレンジ」コンテストに科学者チームを率いて

参加した。彼のチームは実世界のセキュリティケーススタディとしてこの参加を意図しており、最終的に査読付き学術論文を執筆した。論文が発表される前に、RIAA（録音産業協会）が Felten 教授と主催者に、論文の発表が DMCA を含む知的財産法に違反するという警告書を送った。

DMCA にはリバースエンジニアリングに関連するいくつかの免除規定もある——コンピュータプログラム間の相互運用性の実現に必要な場合の技術的保護システムの回避、合法的な暗号化研究の過程での回避、コンピュータセキュリティテストを目的とした回避などだ。残念ながら、これらの免除規定はいずれも複雑で狭い。リバースエンジニアリングの行為が認められている場合でも、DMCA は結果として得られた情報に対してできることを厳しく規制している。

1201(f)：相互運用性のためのリバースエンジニアリング

この免除規定は相互運用性のためのリバースエンジニアリングのための技術的保護手段の回避を認める。またリバースエンジニアリングから得た情報の非常に限定的な範囲での普及も認める。なお、1201(f)は Felten の SDMI ウォーターマークへの攻撃を免除しない——相互運用性との関係がないからだ。

先述の 2600 事件は、2600 誌のウェブサイトへの DeCSS の公開に関する事件だ。DeCSS は DVD 映画へのアクセスを制御する技術的保護手段である CSS を迂回するために使える。EFF は 2600 誌を代理して、DeCSS は 1201(f)の相互運用性特権の要件を満たすと主張した。DeCSS は正当に購入した DVD 映画を Linux コンピューターなど自分が選んだプラットフォームで再生できるソフトウェアを作れるように設計されたもの、と我々は主張した。裁判所はこの主張を退け、1201(f)はプログラム対プログラムの相互運用性実現を目的とした回避のみを認めているが、DeCSS が可能にするのは 1201(f)がカバーしないプログラム対データの相互運用性だと述べた。この判決は疑わしい——DVD には映画のデータだけでなくコンピュータプログラムも含まれているからだ。

1201(f)は Sega 判決に倣って相互運用性のリバースエンジニアリングを認めているように見えるが、いくつかの点でより制限的だ。相互運用性だけがリバースエンジニアリングが行われる唯一の正当な目的であること、プログラム対プログラムの相互運用性のみが適格であり、ハードウェア対プログラムやプログラム対データの相互運用性実現に回避が必要な場合は含まれないこと、そしてリバースエンジニアリングから得た情報は自由に発表できないことだ。

1201(g)：暗号化研究

DMCA には暗号化研究のための明示的な免除規定も含まれている。しかし残念ながらこれも非常に狭い。まず、この免除は暗号学者が回避を実行する前に著作権者に許可を求めた（たとえ得ていなくても）場合にのみ適用される。第二に、法律は暗号学者がこの免除の対象となるために専門家であることを必要とし、暗号学の分野で最も優れた頭脳の一部が正式な訓練を受けていないにもかかわらず。第三に、法律は暗号解析者がアクセス制御を迂回するツールを作ること認めるが、使用またはコピー制御を迂回するツールが認められるかどうかについては沈黙している（つまり一方の反デバイスルールには例外

があるが両方にはない)。第四に、復号の結果の普及について暗号学者の能力を規制している。

Felten 教授の SDMI 研究を再び考えると、デジタルウォーターマークは暗号化ではないため 1201(g)の免除は受けられない。

1201(j)：セキュリティ研究

DMCA のセキュリティ研究免除は同様の構造を持っている。テスターが事前に許可を求めた場合にのみ適用され、アクセス制御を迂回するツールの作成のみを認めており、コピーや使用制御ツールは認めない。1201(g)と同様に、テストの結果の普及についても規制している。

この狭い形でも、Felten の研究が対象になるかどうかは明確ではない。1201(j)はアクセス制御を迂回するツールの作成のみを認める。デジタルウォーターマークはアクセス制御かコピー制御か？ この問いへの答えはウォーターマークの使い方に大きく依存する。EFF は、RIAA が想定したように、SDMI ウォーターマーク技術はアクセス制御と複製制御の両方の技術であると主張した。

エンドユーザーライセンス契約とリバースエンジニアリングの契約上の禁止

知的財産だけがリバースエンジニアリングの障害ではない。ソフトウェアライセンスがリバースエンジニアリングを禁止することは一般的だ。典型的なライセンス条項はこう言う可能性がある——「あなたはソフトウェアからソースコードを逆アセンブル、逆コンパイル、またはその他の方法で導出すること、ソフトウェアをリバースエンジニアリングすること、ソフトウェアを改変または派生物を準備すること、本契約で明示的に許可されている場合を除いてソフトウェアをコピーすること、ソフトウェアをレンタルまたはリースすること、あるいはライセンサーまたはその他の当事者の知的財産やその他の権利を侵害する方法でソフトウェアを使用することをしてはならず、また他者がそうすることを許可してはならない。」

企業はそのような規定が購入者をリバースエンジニアリングしないことに法的に縛ると主張する。それでもリバースエンジニアリングをした場合、契約に違反したとして損害賠償を求めて訴えられることがある。もちろん問題は、反リバースエンジニアリング規定がたとえば Sega 判決の下でのものより広い権利を著作権者に与えることだ。

この種の契約上の禁止が強制可能かどうかは激しく争われている問題だ。裁判所はこの活動がソフトウェアの認可された使用範囲を超えるとして営業秘密事件でリバースエンジニアリングの抗弁を時に退けた。38 裁判所はリバースエンジニアリングに反するソフトウェアシュリンクラップライセンスの制限を連邦知的財産政策との衝突を理由に時に強制拒否した。Vault Corp. v. Quaid Software Ltd.において、39 コピープロテクションソフトウェアのメーカーがルイジアナ州法に基づくコピープロテクションスキームをリバースエンジニアリングした企業に対して反リバースエンジニアリング条項を強制しよ

うとした。裁判所は、Bonito Boats のフロリダのボート外装法を無効とした際に使われたのと同じ連邦政策の論拠として、連邦法が契約条項を先占すると判示した。

また、著作権法の第 301 条は著作権の「一般的範囲内の独占的権利に等価な……」州で作られた、または州によって強制される権利を先占する。予想通り、「等価な」が何を意味するかについては議論がある。裁判所は州法下で強制可能な契約規定が侵害の条件が同じ場合に連邦著作権と「等価な」と述べてきた。しかし州で作られた権利の侵害が「追加の要素」を必要とする場合は「等価」ではない。

そのような契約条項が最近強制可能と判断された。Bowers v. Baystate Technologies, Inc.において、40 発明者は反リバースエンジニアリングライセンス条項を付けて特許取得済みのコンピュータ支援設計 (CAD) ソフトウェア「ツールキット」を販売した。競合他社の Baystate は Bowers のソフトウェアをリバースエンジニアリングして競合する CAD ツールキットを販売した。複雑な訴訟の末、裁判所は最終的に Baystate が Bowers との契約に違反したと判示した。

裁判所はライセンスが先占されていないと判示した。契約には「追加の要素」があるからだ——当事者が合意しなければならない。41 これによると連邦著作権法は契約上の禁止を先占することは決してできないことになる。Bowers 判決の問題は著作権法の特定の先占条項にのみ焦点を当て、憲法上の「衝突」先占を完全に無視していることだ。42

Uniform Computer Information Transactions Act (UCITA) はこれらの問題に対処する州立法上の試みだが、これも論争に巻き込まれている。

営業秘密と経済スパイ法

経済スパイ法 (EEA) 43 は営業秘密の不正流用に対する最初の連邦訴因を作り出した。しかしリバースエンジニアリングの抗弁はない。これは問題だ。EEA の下で付与された権利は、以前は合法だと考えられていたある種のリバースエンジニアリング活動に対して疑問を呈するからだ。特に、コンピュータプログラムの逆コンパイルと逆アセンブルが EEA の営業秘密の複製を禁じるルールに違反するかどうかは明確でない。

責任あるハッカー：無知は弁護にならない

一般に知的財産法を侵害する方法は二つある。直接侵害は実際に侵害したことを意味する。間接侵害は他者による実際の侵害を促進したことを意味する。たとえば Betamax 事件では、問題は Sony が VCR を販売することで顧客の著作権侵害に対して責任を負うかどうかだった。

民事および刑事上の罪と罰則

私たちが論じてきた法的理論は幅広い潜在的な罰則を伴う。主な懸念は民事責任、経済的損害賠償またはその活動に対する差止命令、あるいはその両方だ。損害賠償は通常侵害によって引き起こされた損害の量に連動する。

特許法では、損害賠償の通常の根拠は「合理的なロイヤリティ」だ。裁判所は、ライセンスを契約した場合に特許権者に支払うべきだったロイヤリティを計算する。損害賠償は侵害者の利益または特許権者の失った利益に基づくこともある。

「故意の」侵害はより厳しく扱われる。特許法は、侵害者が特許を知っていながら有能な特許弁護士に相談しなかった場合、裁判所の裁量で基本損害賠償の3倍まで増額（および特許権者の弁護士費用の支払い）を認めている。

知的財産法における現在のトレンドは刑事罰への注目の高まりだ。1790年の最初の連邦著作権法では著作権侵害は純粋に民事問題だった。1897年まで議会が著作権法に刑事罰を加えず、著作権侵害は軽罪として分類された。44 さらに刑事著作権侵害が使われることはほとんどなかった。

今日、刑事訴追のリスクはかなり高くなっており、刑事罰も大きくなっている。1982年と1992年の著作権法の改正は、特定の種類の侵害を重罪に分類した。それでも刑事侵害は故意に、商業的利益または個人的な金銭的利益のために行われる必要があった。

1997年のNo Electronic Theft法（NET法）は、コピーがどのように作成または配布されたかに関わらず、任意の180日間に合計小売価格が1,000ドルを超える著作権著作物の1部以上のコピーの複製または配布を犯罪化した。故意要件は維持したが、被告の侵害が利益や商業的利益によって動機づけられていることの要件は削除した。

DMCAにも刑事規定が含まれており、Dmitry SklyarovとElcomSoftの訴追に援用された。ElcomSoftはAdobeのeBookフォーマットからAdobeのPDFフォーマットにデジタル書籍を変換するために使えるソフトウェアを製造・配布した。フォーマット変換の過程で、eBookフォーマットが課している使用制限が取り除かれる。ElcomSoftのソフトウェアがeBookの著作権を侵害しない使用（例：フェアユースによる抜粋、視覚障害者のためのブライユ点字への自動翻訳）を促進するために使えることは争いがなかった。Sklyarov自身は著作権の侵害、あるいは第三者の侵害行為への関与で告訴されたことは一度もなかった。それでもソフトウェア開発への関与のためにFBIが彼を逮捕し3週間拘留した。45 彼とElcomSoftは大陪審によって起訴され、起訴状に基づく、Sklyarovは最長25年の懲役と200万ドルを超える罰金に直面した。46 ElcomSoftとSklyarovは最終的にDMCA違反で無罪となった。

リバースエンジニアリングとしての「いじる自由」とその他の法的問題

プリンストン大学のコンピュータサイエンス教授Edward Feltenはリバースエンジニアリングを「いじる自由」の一部と見ている。それは「機器を分解し、議論し、どのように動くかを探求し、改造し、より良くする」自由を含むべきだ。Feltenは「電子デバイスを通じて私たちの世界のますます多くの部分が経験され、コミュニケーションと文化がこれらのデバイスによってますます媒介されるにつれて、それらをいじることができること、私たちの世界のこの部分を理解できることがますます重要になる」と主張する。

いじる自由にはいじることについて話す権利も含まれるべきだ。しかし見てきたように、多くの新しい知的財産ルールはリバースエンジニアリストが触った後に学んだことを共

有する権利を制限している。これらの制限は深刻な合衆国憲法修正第一条のフリースピーチ問題を提起するだけでなく、著作権と特許法の憲法上の基盤である芸術と科学の進歩の核心に触れる。DMCA が提起する主要な問題の一つは科学者への萎縮効果だ。47

注

1. Joel Miller, Reverse Engineering: Fair Game or Foul?, IEEE Spectrum, Apr. 1993, at 64, 64.
2. 17 U.S.C. § 1201–1204.
3. 18 U.S.C. § 1831–39.
4. U.S. Const. Art. I, § 8, cl. 8. 憲法が書かれた当時、「science」は「knowledge (知識)」の同義語としてしばしば使われていた。
5. See *Diamond v. Diehr*, 450 U.S. 175 (1981); *In re Alappat*, 33 F.3d 1526 (Fed. Cir. 1994).
6. ヨーロッパでは著作権は伝統的に作品の創作者の固有の譲渡不能な個人権を保護するものとして見られてきた。
7. “The economic philosophy behind the clause empowering Congress to grant patents and copyrights is the conviction that encouragement of individual effort by personal gain is the best way to advance public welfare through the talents of authors and inventors in ‘Science and useful Arts.’” *Mazer v. Stein*, 347 U.S. 201, 219 (1954).
8. *Feist Publications, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 348–49 (1991).
9. See 17 U.S.C. Sec. 109.
10. たとえばフォノレコードとスタンドアロンコンピュータプログラムは Sec. 109 の下で書籍とは異なる扱いを受ける。
11. 現行著作権法では、MC 複製著作権は強制ライセンス規定によって制御されており、法定料率を支払うことで自動的に許可が得られる。
12. 17 U.S.C. § 117.
13. *Mirage Editions, Inc. v. Albuquerque A.R.T. Co.*, 856 F.2d 1341, 1344 (9th Cir. 1988).
14. See, e.g., *Lee v. A.R.T. Co.*, 125 F.3d 580 (7th Cir. 1997).
15. 最初の著作権法の下では保護期間はわずか 14 年だった。
16. 17 U.S.C. § 107.
17. *Sony v. Universal City Studios* 464 U.S. 417, 429 & 432 (1984).
18. See generally Neil Weinstock Netanel, *Locating Copyright Within the First Amendment Skein*, 54 *Stan. L. Rev.* 1 (2001).
19. See generally L. Ray Patterson, *Free Speech, Copyright, and Fair Use*, 40 *Vand. L. Rev.* 1 (1987).
20. See generally Pamela Samuelson & Suzanne Scotchmer, *The Law and Economics of Reverse Engineering*, 111 *Yale L. J.* 1575 (2002).

21. The Court went on to say that “[w]here an item in general circulation is unprotected by a patent, ‘[r]eproduction of a functional attribute is legitimate competitive activity.’”
22. U.S. Const. art. VI, cl. 2.
23. Atari Games Corp. v. Nintendo, 975 F.2d 832, 842 (Fed. Cir. 1992).
24. Id., quoting U.S. Const. Art. I, § 8, cl. 8.
25. Feist Publications, Inc., v. Rural Telephone Serv. Co., Inc., 499 U.S. 340, 350 (1991).
26. 977 F.2d 1510 (9th Cir. 1992).
27. Id. at 1526-1527.
28. Id. at 1518.
29. See Pioneer Hi-Bred Int’l, Inc. v. DeKalb Genetics Corp., 51 U.S.P.Q.2d (BNA) 1797 (S.D. Iowa 1999).
30. See Roche Prod. v. Bolar Pharmaceutical Co., 733 F.2d 858, 858-63 (Fed. Cir. 1984).
31. これらの州法は Bonito Boats で最高裁によって無効とされた。
32. Semiconductor Chip Protection Act, Pub. L. No. 98-620, 98 Stat. 3347 (1984) (codified at 17 U.S.C. § § 901-914 (1994)).
33. Agreement on Trade-Related Aspects of Intellectual Property Rights (TRIPS), Apr. 15, 1994.
34. See, e.g., <http://www.sethf.com>.
35. Universal City Studios v. Reimerdes, 111 F.Supp. 294 (S.D.N.Y. 2000), aff’d 273 F.3d 429 (2d Cir. 2001).
36. DMCA はそれらが保護するものに基づいて二種類の技術をカバーする。著作権のある著作物へのアクセスを「有効に制御する」技術 (Sec. 1201(a)(2)) と著作権者の権利を「有効に保護する」技術 (Sec. 1201(b)(1))。
37. これは奇妙に見えるかもしれないが、セキュリティの多くの学術論文にはコンピュータプログラムコードが含まれていることを考えてほしい。
38. E.g., Technicon Data Sys. Corp. v. Curtis 1000, Inc., 224 U.S.P.Q. (BNA) 286 (Del. Ch. 1984).
39. 847 F.2d 255 (5th Cir. 1988).
40. 302 F.3d 1334 (Fed. Cir. 2002).
41. The court relied on an earlier case, ProCD, Inc. v. Zeidenberg, 86 F.3d 1447, 1454. (7th Cir. 1996).
42. EFF は Baystate の再審査申立てを支持するアミカスブリーフを提出した。
43. Economic Espionage Act of 1996, Pub. L. No. 104-294, 110 Stat. 3488 (codified at 18 U.S.C. § § 1831-1839 (Supp. V 1999)).
44. See generally Lydia Loren, Digitization, Commodification, Criminalization: The Evolution of Criminal Copyright Infringement and the Importance of the Willfulness Requirement, 77 Wash. U. L.Q. 835, 840 (1999).

45. See Professor Larry Lessig, “Jail Time in the Digital Age,” N.Y. Times (July 30, 2001).
 46. See Brad King & Michelle Delio, “Sklyarov, Boss Plead Not Guilty,” Wired News (Aug. 30, 2001).
 47. See generally Electronic Frontier Foundation, Unintended Consequences: Four Years under the DMCA (2003).
-

第13章 前進！

コミュニティへの貢献

Xbox ハッキングコミュニティに貢献する方法を探しているなら、まずこのことを覚えておいてほしい——ほとんどのハッカーは、自分が最も興味を持っている分野に対応した独自のスキルや強みを持っており、楽しみのためにハックしているということだ。

たとえば、僕は特にハードウェアが好きで、何かを作ることが求められるプロジェクトに強くひかれる。コードを書くことはできるが、それほど楽しいとは思えない。だから、ハッキングコミュニティへの僕の貢献は主としてハードウェアプロジェクトを通じたものになっている。同じ理由で、ソフトウェアプロジェクトに自分を奮い立たせることも難しい。だから多くの場合、ただ後ろに引いて、他の人たちが Xbox で何をやっているかを眺めている。それはとても勉強になるし、楽しくもある。

Xbox ハッキングコミュニティに何ができるかを考えるとき、急いで飛び込もうとしたり、夢中になれないプロジェクトや、自信を持って実行できないプロジェクトに無理に取り組もうとしたりしないでいい。自分の番が来たときには自然にわかる——プロジェクトが自分の名前を呼ぶような感覚があり、自然にハックせずにはいられなくなるだろう。

コラム：Dan Johnson（別名 SiliconIce）プロフィール

ご自身のことと、ハッキングへの関わりについて教えてください。

以前からエレクトロニクスのハッキングに興味がありましたが、それはほとんど読んで楽しむだけのものでした。セガの Dreamcast や Netpliance I-Opener といったハックされたデバイスが時々目に入ってくる程度でした。エレクトロニクスハッキングで最初の実体験となったのは「ePods」——販売が打ち切られたインターネットアプライアンス/ウェブタブレット端末——でした。このデバイスのことをオンラインで読み、Ken Segler の I-Appliance BBS (<http://www.linux-hacker.net>) にたどり着きました。有名な I-Opener ハックの拠点です。タブレットを使ってみんながやっていることを読んで興味を持ち、当時の貯金からかなりの金額 (\$200) を使ってユニットを1つ入手しました。届いたばかりの玩具で、文書化されたハックの多く——主にソフトウェアベースのもの——を实践しました。これがエレクトロニクスハッキングの世界への最初の一步でした。ePods の後、別のインターネットアプライアンスである Gateway Connected Touchpad に移りました。400MHz Crusoe プロセッサと RAM 96MB を搭載した 10 インチタッチスクリーン。見栄えも良く、楽しいプロジェクトにぴったりに見えました。そのデバイスは AOL 向けにカスタマイズされた Linux を 32MB の CompactFlash カードから起動していました。それを Microdrive と入れ替え、ラップトップ

と USB リーダーと Gateway 間でディスクを何度もやり取りした末に、「98Lite」を使ったスリムダウン版 Windows が起動しました。指で操作できるネットワーク接続型 mp3 ジュークボックス（その他用途も）として最適に思えたため、タッチパッドで操作しやすいカスタムプレーヤーの開発に取り掛かりました。Gateway 全体の経験は、高校 3 年と 4 年の間の夏のプロジェクトとしてとても面白いものになりました。

XboxHacker BBS はどのように生まれたのですか？

まさにこの時期に、XboxHacker.Net のアイデアが浮かびました。僕の目には、Xbox はテレビのそばに置ける理想的なコンピューティングデバイスになる可能性があるように見えました——プログラムできるようになりさえすれば。当時のハードウェアとしては非常に印象的で、実現したいことには十分すぎるほどでした。他のコンソールと違い、Xbox にはハードドライブとイーサネットポートが内蔵される予定でした。ハックされれば、Xbox は NES、SNES、N64、PSX などのエミュレーション、mp3 や DivX などのメディアファイルの再生、ホームネットワークからのストリームメディア再生、さらには基本的な PC としての利用にも使えるはずでした。\$299 という価格から、近いスペックの PC をハックなしで組めるとの理由で Xbox をハックするのは割に合わないという意見もありました。しかし Xbox には、Xbox ゲームも遊べてテレビと一緒に置けるという強みがありました。ハックはあくまでも付加価値です。電子機器ハッキングの世界に少し足を踏み入れた後、Xbox ハッキングはテレビのそばでさまざまなタスクを実行できる便利な箱を手に入れるための興味深いプロジェクトを整理するのに役立てられると思い、取り組んでみようと考えました。実際に XboxHacker.Net のドメインを取得したのはそれから何ヶ月も後のことでした。

XboxHacker BBS を成長させ運営した経験はどうでしたか？

最初から XboxHacker.Net はいくつかの主な目標に集中していました——Xbox に関する技術情報を提供・共有すること、そして Xbox ハッキングに関連する技術的な議論をする場を提供することです。技術的な知識と経験が限られていたため実際のハッキングへの貢献は少なかったですが、関連情報を集めて配信し、ディスカッションボードをモデレートするという形でその取り組みを支援するには十分な知識がありました。

サイトの立ち上げ後まもなく、Mike Magee の The Inquirer や Van Smith の Van's Hardware といった有名サイトからリンクをもらえる幸運に恵まれました。XboxHacker.Net BBS は Xbox ハッキングに関するあらゆる話題を議論するためのインターネット上の主要な場となり、XboxHacker.Net のニュースページは最新の Xbox の状況を伝えるものとなりました。サイト公開から数週間後に CNET の記事で言及され、それ以降は活動が着実に増えていきました。ほどなく XboxHacker.Net は利用していた小さな共有サーバーを使い切り、より大きなアカウントに移行しましたが、それも数週間で使い切ってしまいました。フォーラムへのトラフィックは急速に増え続けたため、帯域幅の数 MB や同時接続ユーザー数を心配せずに成長できる専用サーバーに移転することにしました。

XboxHacker.Net の活動はほどなく Microsoft の注目を集めました。サイト初期に、素材の削除を求める連絡を数回受けました。最初はディスクの「セキュリティセクター」を示す開発者ツールのスクリーンショットについて、もう 1 つはフォーラムに誰かが投稿した Xbox の BIOS イメージへのリンクについてでした。これらの小さな問題を除けば、Microsoft と XboxHacker.Net の間の接触はほぼ皆無でした。ゲームのバックアップについての議論、著作

権のある素材へのリンク、BIOSからのMicrosoftコードの投稿など、法的に疑問のある問題には距離を置くことを早い段階でポリシーとしていました。

最初は関心が高かったものの、進展は比較的ゆっくりとしていました——セキュリティが破られる前はXboxでできることが多くなかったのです。初期から注目すべき貢献をした人たちがいました。Xboxハッキングの世界の最初期の貢献者の中には、ファイルフォーマットや他の貴重なXbox情報を多数掲載したWebページを持つAndyとLukeがいました。Steve「SurferDude」GehlbachはXbox向けVGAコンバータ回路の設計に貢献し、Ken GasperはそれをもとにXbox向けの真のVGAアダプタを作り上げました。bunnieの初期の解析とハードウェア情報ページはSlashdotへの掲載後、Xboxハッキングへの関心を高めました。その他にも多くの人が足跡を残しました。そのような優れたハッカーたちをボードの貢献メンバーとして迎えられたことで、XboxHacker.NetはXboxの技術情報・議論・ニュースのハブへと成長していきました。

XboxHacker.Netの目玉は間違いなくXboxHacker BBSでした。サイトはフォーラムとメンバーの活動を中心に成り立っていました。最新のXboxハッキングニュースは、メンバーが投稿したフォーラムの話題へのリンクや引用であることもよくありました。フォーラムの目的は、Xboxハッキングに関心を持つ多国籍の多様なグループが共通の目標に向けて協力できる場を作ることでした。フォーラムはハッカー仲間と情報を共有しアイデアを議論する場として機能し、そうでなければ協力する手段を持てなかった多くの優れたハッカーを結びつけました。フォーラムが可能にした進展を見てみると、とても良い気持ちになりました。特に印象に残っている議論が数日間に渡って展開されたことがありました。複数のメンバーが第2世代Xboxの新しいセキュリティシステムを調べ、欠陥と回避方法を探していました。ハッカーたちが解答に近づき、やがて改訂版Xboxのセキュリティを突破するまでの経緯を目の当たりにするのはとてもわくわくするものでした。数百人の貢献メンバーが、XboxHacker.Netの成功と全体的なXboxハッキング世界の進展に大きく貢献しました。

XboxHacker.Net BBSの活動のほかにも、秘密裏にXboxをハックしていたグループが多数あったのは間違いありません。知っていたグループの1つはIRC上にあり、フォーラムの貢献メンバーや電子機器ハッキングのアンダーグラウンドからの人々も含まれていました。ここでは名前は伏せます。独立したグループや個人から、時間をかけて多くの情報が報告のために僕に届けられました。しかしXboxハッキングへの関心と関与者の数が増えていたにもかかわらず、大きなブレイクスルーが公に知られるようになるまでにはまだ時間がかかりました。

Xboxハッキングの場面は、2002年夏頃にモッドチップが公に入手可能になってから急速に盛んになりました。その頃にはXboxのセキュリティシステムは複数のグループによって突破されており、モッドチップが一般に広く入手できるようになるのは時間の問題でした。この時点から、取り組みはハードウェアハッキングから離れ、ソフトウェア開発へと移っていきました。XboxHacker.Netの姉妹サイトXboxDeveloper (<http://www.xboxdeveloper.net>) は、Xbox向けに公開されたソフトウェアのカタログ作りを目的に設立されましたが、XboxHacker.Netほどの規模には成長しませんでした。MicrosoftのXbox SDKの流出コピーを使い、プログラマーたちはMAMEなどのメディアプレーヤーやエミュレーターを含むさまざまなXbox向けアプリケーションの作成とポートに取り掛かりました。Michael Steilが率いるXbox-Linuxプロジェクト (<http://xbox-linux.sourceforge.net/>) は、XboxでLinuxオペレーティングシステムを動かすことに成功しました。僕はOpenXDKプロジェクト (<http://sourceforge.net/projects/openxdk/>) の立ち上げを働きかけました——法的問題

に縛られることなく Xbox 向けソフトウェアを開発できるオープンソースの開発キットです。ただし、このプロジェクトは成功と失敗が混在しています。現在は Case Western 大学の CS 学生「Caustik」(Aaron Robinson) が率いています。ハードウェアハッキングと比べてソフトウェア開発がより身近であるため、Xbox ハッキング全体の取り組みに貢献するメンバー数は急速に増えていきました。今日の Xbox 向けホームブリュソフトウェアの量はすばらしく、メディアプレーヤー、コンソールエミュレーター、ユーティリティから、オリジナルゲームまで揃っています。

トラステッドコンピューティング

信頼はセキュリティの礎であり、セキュリティシステムを信頼するためには、実行しているハードウェアとソフトウェアを信頼できなければならない。トラステッドコンピュータとは、マシンの信頼性を損なう恐れのある攻撃に対して耐性を持つよう設計されたマシンのことだ。

信頼性の高いコンピュータを構築するアプローチはさまざまあり、ATM（現金自動預け払い機）方式の物理的なセキュリティや耐タンパー性から、Xbox で採用されているようなハードウェアの負担が少ない方式まで多岐にわたる。Xbox は、最初期の広く普及したトラステッド PC 実装のひとつとして、トラステッドコンピューティングという大きな絵の中に位置づけられている。ある意味で Xbox は、トラステッドコンピューティングの将来について示唆を与えてくれる存在だ。

トラステッドコンピューティングは、潜在的に破壊力を持つ新興技術だ。インターネットのようなアドホックなネットワーク上でトラステッドクライアントが広まれば、安全でプライベートなオンライン金融取引を可能にし、コンピュータウイルスやスパムメールを減らすまたは完全になくすという可能性を秘めている。トラステッド PC は、医療・金融記録や個人的な秘密といった機密データを安全に保管するためにも使える。また、デジタルコンテンツのアクセス権と管理ポリシーをユーザーに対して確実に適用するためにも利用できる。トラステッドコンピューティングのデジタル著作権管理（DRM）の側面は、今日のコンピュータの使い方を根本から変える可能性がある。僕たちの多くは、事実上フリーなコンテンツと柔軟な著作権実装の恩恵を享受している。根本的な問題はコンテンツ管理ポリシーの存在そのものではない——コンテンツ管理ポリシーは消費者にとって有益になり得る。本当の問題は、ユーザーの権利を決めるポリシーがコンテンツ提供者によって一方的に設定されうるときだ。現在のトラステッド PC の提案では、ユーザーはマシン内部の特定の重要な秘密に対するコントロールを信頼されていない。代わりに、ユーザーのマシンの認証情報（信頼性の確立に必要な情報）の一部が第三者によって管理される。選挙で選ばれたわけでもなく、規制もされていない、ビジネス上の利害を持つ第三者が、誰がビジネスを行い、メールを送り、信頼できる存在として認識されるかを定める権限を持つことを、僕たちは信頼できるのだろうか？

トラステッドコンピューティングは銃のようなものだ。引き金を自分が握っている限り、持っていることはすばらしい。しかし多くのトラステッド PC 反対派が恐れるのは、実際には、消費者にとって不利な方向を向いた既定の権利・ポリシーとサードパーティの信頼確認リソースを設定した状態でシステムが展開されるということだ。企業は好景気の

ときにはより大きな顧客基盤を引きつけるためにユーザーに有利なプライバシーとセキュリティのポリシーを設定するかもしれないが、いったん倒産の危機が訪れたり、企業が売却されたり経営が変わったりすれば、ポリシーは変わりうる。当初はユーザーフレンドリーなポリシーでトラステッドコンピューティングを普及させ、突然、消費者の財布を絞る DRM モードに切り替えるビジネスマンを倫理が止められるだろうか？

一歩引いて考える

「トラステッドコンピューティング」というフレーズには問題がある——これが暗号学的に保護されたトラステッドコンピュータと同義語になってしまっているのだ。一歩引いて、トラステッドコンピュータを構築する代替アプローチについて話してみよう。

信頼性はコンピュータにとって常に重要だった。しかし、コンピュータの黎明期には機器があまりに高価で、強力な信頼ポリシーを適用するためのハードウェアは消費者の手の届くところになかった。たとえば、初期のマシンの多くはハードウェアの MMU (Memory Management Unit、メモリ管理ユニット) チップ用のソケットを備えていた。MMU は信頼性の高いハードウェアメモリモデルへの最初の一步のひとつであり、MMU の仕事の一部はページレベルのメモリアクセス保護を適用することだ。当時はかなり高価だったため、MMU はオプションとして販売されていた。残念ながら、信頼性の高いハードウェアへの動きは MMU で止まってしまった。主な理由の一つは、コンピュータネットワークが大規模に存在するようになったのが比較的最近だからだ。ネットワークのない世界では、データはプログラムのエラーと、マシンへの物理的なアクセスを持つ少数の特定ユーザーによるアクセスからのみ保護できればよかった。今日のコンピュータには、MMU 以上の何か——ウイルスや、悪意あるコードを実行するためにソフトウェアの微妙な脆弱性を突こうとするリモートの攻撃者に立ち向かえる信頼性——が必要だ。

MMU のハードウェアによってページング化された仮想メモリモデルの自然な拡張として、タグ付きメモリモデルを使ったアドレスケイパビリティが考えられる。メモリタグとは、メモリ位置に格納されているデータやコードのタイプを記録するビットの集合だ。タグ付きメモリモデルでは、すべてのメモリ位置に一連のタグビットが付与される。これはちょうど、従来のエラー訂正メモリ実装では各メモリ位置に ECC ビットが関連付けられているのに似ている。タグビットはハードウェアがデータ型管理ポリシーを適用するのを助ける。たとえば、データとしてタグ付けされたメモリ位置は、誤って意図的にコードとして実行されることがない。ケイパビリティとは、信頼されたカーネルによって付与された、偽造不能なポインタだ。偽造不能性はタグビットによってハードウェアで適用されるのが好ましい。多くのアーキテクチャでは、ハードウェアケイパビリティの一部としてアクセス境界を適用する機能も含んでいる。1

ケイパビリティとメモリタグは新しいアイデアではない。1961 年には、Burroughs B5000 がケイパビリティ (当時は「ディスクリプタ」と呼ばれた) とタグ付きメモリを使って、バッファオーバーフロー攻撃からハードウェアで保護し、コードとデータを分離していた。2 MIT PDP-1、Intel i432、IBM System/38、Mach および Amoeba OS もある形でケイパビリティを実装しており、このリストはもちろん網羅的ではない。ケイパビリティのセキュ

リティ特性は、EROS (Extremely Reliable Operating System) など多くの学術研究でも実証されている。3 残念ながら、ケイパビリティとタグ付きメモリはメインストリームのPC アーキテクチャの核心に入り込むことはなかった。セキュリティと信頼性はコスト・後方互換性・パフォーマンスの前に常に後回しにされてきた。

このような簡単な歴史の教訓が示すのは、トラステッドコンピューティングが今日 Palladium や TCPA (Trusted Computing Platform Alliance、トラステッドコンピューティングプラットフォームアライアンス) によって提案されている暗号的アプローチを必要とするわけではないということだ。実際、暗号化それ自体はいかなるセキュリティも提供しない。安全な鍵管理こそが、Palladium/TCPA におけるすべてのセキュリティを提供している。暗号アルゴリズムは単に、鍵のセキュリティをユーザーの領域に移し替えるだけだ。

こう考えると、ケイパビリティと暗号鍵の間には類似点が見えてくる。どちらも、その生成・配布・破棄を管理するために信頼された OS を必要とする。どちらも、システムが偽造された鍵やケイパビリティから保護できなければ同様に弱い。大きな違いは、秘密鍵が漏洩した場合、セキュリティはすべて、永遠に失われるという点だ。一方、ケイパビリティは動的に生成・破棄されるため、ケイパビリティが漏洩してもセキュリティ侵害の範囲と期間は限られる。この点で、ケイパビリティはコンピュータセキュリティにおいてより堅牢な解決策を提供する。

ハードウェアセキュリティのために暗号技術だけに頼ると、古典的なバッファオーバーラン型攻撃やプログラムエラーによるセキュリティホールに対してマシンが開かれたままになることに注意しておく必要がある。ソフトウェアの状態の「計測」は、セキュリティクリティカルな操作を実行する前にコードの変更を検出することでこの弱点を軽減するが、計測は完璧な解決策ではない。一方、ハードウェアによるケイパビリティと境界チェックを使用するシステムではバッファオーバーラン攻撃は不可能だ。

メモリタグは、純粋に暗号的なアプローチでは実現不可能なセキュリティ機能を実装するためにも使える。一例として、コンパートメント化された秘密情報の信頼性の高い並行処理が挙げられる。4 この例では、さまざまなセキュリティクリアンスレベルを持つ複数のスレッドが1つのプロセッサ上で動作している。ハードウェアは、すべてのスレッドがアクセスするデータにそれぞれのセキュリティレベルを刻印するというポリシーを適用する。つまり、すべての計算は通常の算術演算のドメインとセキュリティドメインという2つのドメインで同時に動作する。

たとえば、非機密スレッドが2つの非機密の数を加算して「foo」というデータを作成するとする。foo のセキュリティタグは加算の算術演算と並行して計算される。この場合、セキュリティタグの結果は「非機密」だ。今度は、最高機密スレッドがfoo にアクセスするとする。foo のセキュリティタグは「最高機密」に変わる。非機密スレッドは、foo への有効なポインタを持っていてもfoo を読み取れなくなる。foo は明示的に再分類されない限り、非機密スレッドが再び読み取ることができない。このような厳密にコンパートメント化されたセキュリティシステムは、たとえば、カーネルの内部構造が（メモリの解放と

ユーザープロセスへの再割り当てという手順のない場合を含むバグやメモリリークがある状況でも) ユーザープロセスからアクセスされないことを保証するために使える。このスキームは、セキュリティ侵害の根本原因をプログラマーが追跡するのに役立つセキュリティ監査証跡を確立するためにも、またセキュリティ侵害が発生した場合のダメージコントロール手段としても使用できる。

公正を期すると、暗号学的なトラステッドコンピューティングアプローチには1つの利点がある。もしある悪意ある者が何らかのハードウェア的手段でデータを入手しても——ハードウェアの盗聴やハードドライブの盗難など——そのデータは解読できない。しかし、ユーザーはハードウェアケイパビリティとタグ付きメモリを使った実装を含め、あらゆるコンピュータ実装でデータ保護のために暗号化を選択できる。安全な鍵管理の問題は依然として難しい問題だが、PCに暗号スマートカードリーダーを統合することで部分的に解決できるかもしれない。

トラステッド PC の実装が Palladium や TCPA で提案されている暗号技術を使わなければならない本質的な理由はない。このセクションで見てきた技術——すなわちケイパビリティとタグ付きメモリ——は、ユーザーが自分自身のアクセスポリシーを設定する能力を失うリスクを伴わない形で、安全で信頼性の高い PC を実装するために使える。ユーザーは常に自分のマシンと秘密情報の完全なコントロールを持つことができる。

Palladium と TCPA の比較

現在のトラステッド PC 提案、すなわち Microsoft の Palladium と TCPA (Trusted Computing Platform Alliance) についての混乱が多い。2つの提案には十分な類似点があるため、多くの人が同じものだと思っているが、それぞれのイニシアティブの目標は異なる。

TCPA は、ある程度の名目的な信頼性を持つコンピュータを作るための複数企業による連合だ。特筆すべきことに、その仕様の多くは PC 以外のプラットフォームにも適用できる。TCPA では、信頼性は TPM (Trusted Platform Module、トラステッドプラットフォームモジュール) と呼ばれる安全なハードウェアモジュールに根ざしている。TPM には、モジュール内に格納された秘密情報がソフトウェア攻撃によって漏洩しないことを保証するための機能が含まれている。また、TPM に含まれる信頼性をホストマシンに移転しようとする、セキュリティの「計測」などの機能も含まれている。

Palladium は一方で、Microsoft が単独で作った PC 中心のシステム全体のセキュリティコンセプトだ。構成要素の一つは TPM に似たセキュリティモジュールだが、Palladium はハードウェアチップセットと I/O ポートの実装方法への大幅な変更も求めている。チップセットは、グラフィックカードなどの潜在的なすべての DMA ソースからのメモリセキュリティポリシーを適用することが求められる。Palladium はキーボードとビデオサブシステムへの I/O の暗号化も求めている。

チップセットベンダー・OEM・Microsoft 間の協力を求める Palladium の要件は、潜在的に大きな欠陥だ。現在のコモディティ PC ハードウェア業界には、大規模な暗号化セキュリティの刷新を支えるだけの余裕がない。また、多くのチップセットベンダーには安全な

システムの実装経験がない。多くの海外チップセットベンダーが抱える言語の壁に加え、チップセットは通常、タイトなスケジュールで予算を厳しく管理しながら開発される。こうした状況で開発されたチップセットが機密情報を保護することは期待できるだろうか？

Xbox は、セキュリティポリシーをある組織が定義し、まったく異なる別の組織が実装する場合に何が起りうるかの例だ。Microsoft は、信頼性の高いハードウェア——Xbox を——の仕様を書いた。強力な暗号アルゴリズムが Xbox では至るところで使われており、システムのマスター鍵は複雑なシリコンの奥深くに封じ込められている。しかし経験が示すのは、Xbox のセキュリティシステムが、非保護のデバッグポート、ハードウェア初期化スキームの欠陥、CPU の命令ポインタの境界ケース処理のバグという組み合わせを使って迂回できるということだ。3 つの独立した当事者（組み立て業者、Xbox ファームウェア設計者、Intel）によってそれぞれ犯された 3 つの小さな見落としが重なり、Xbox のセキュリティを破る便利な方法を提供することになった。

これらの見落としはそれぞれ単独では重大なセキュリティ問題にはならない。このことが、ある特定の Palladium 実装において、複数の無害な欠陥が積み重なることでどれほどのセキュリティ侵害が引き起こされるかという不安な問いを生む。複雑な家庭用電子機器システムは、特に主たる関心が利益を出すことにある複数の独立した組織が作ったコンポーネントで構成されているとき、必ず小さなバグや設計上の見落としがある。

家庭用電子機器業界では、完璧な製品を出荷するか、お金を稼ぐかのどちらかだ。お金にならない製品は早々にキャンセルされる。したがって、あらゆる面で技術的に完璧な消費者製品を見つけることは非常にまれだ。その結果、Palladium ほど複雑な家庭用電子機器システムのセキュリティを保証する唯一の実際的な方法は、野に放ち、すべての大きなセキュリティホールが発見・修正されるまで何年もハッカーたちに好きにさせることだ。

一方で TCPA の TPM は、Palladium より範囲が小さい特定の問題を解決するために作られたデバイスだ。したがって TPM は市場的な観点からはそれほど魅力的ではないが、意図された目的のためにはより実用的で使いやすかもしれない。TPM と Palladium のどちらもハードウェア攻撃には弱い。TPM はセキュリティ要件をサードパーティのシステム設計の領域にそれほど深くまで伸ばそうとしない。TPM は主として、ホストシステムに対するほとんどの変更や侵入を検出できる安全な鍵管理モジュールだ。このサブストレートの上に構築されたソフトウェア層が残りの重労働をこなす——買い手は用心せよ（caveat emptor）。

トラステッド PC をハックする

現在提案されているトラステッド PC は、統合上の見落としやバグ関連のバックドアがなくとも、かなりシンプルなハードウェア攻撃に対して弱い。

最初の攻撃は、僕が「Surreptitious BIOS」すなわち SPIOS（「スパイ OS」と発音する）攻撃と呼ぶものだ。SPIOS は、ユーザーへの不正アクセスを防ぐためにトラステッド PC の暗号化されたストレージ機能に依存する DRM ポリシーを破るために使える。基本的なアイデアは、変更されていない BIOS で PC をトラステッドモードで起動し、目的のデー

タをすべてシステム RAM に取り出し、BIOS イメージを差し替えながらウォームリセットを実行するというものだ。

変更された BIOS イメージを使ってシステム RAM から目的のデータを読み取ることができる。目的のデータとは、プログラムがデータをメモリにどのように構造化してキャッシュするかによって、メモリに格納されたセッション鍵か、実際に復号されたデータ自体かもしれない。現在のトラステッド PC 仕様が LPC バスベースの BIOS を採用しているため、安価な代替ファームウェアデバイス (Xbox で使われるものと同様) でこの攻撃を実行できる。この攻撃を複雑にするためにアプリケーションプログラマーが使える技法として、毎回システムメモリに 1 ブロックのデータだけを復号するというものがあるが、これらの技法の多くはシステムパフォーマンスを大幅に低下させる。特にファイルキャッシュとプリフェッチが無効化された場合、パフォーマンスの低下はとりわけ顕著になりうる。

もう 1 つの攻撃は、「Surreptitious RAM」すなわち SPAM 攻撃と僕が呼ぶものだ。この攻撃の目的は、システム状態の適合性を計測する信頼されたルーティンを欺くことだ。FPGA (Field Programmable Gate Array) や ASIC などのデバイスが、DRAM チップとメモリコネクタの間にある差し込み式メモリカードに取り付けられる。このデバイスはアドレスのパターンを監視するか、システムを認証する暗号モジュールの I/O ピンに接続されたワイヤの状態をスニッフする追加コネクタを持つかもしれない。

いずれにせよ、システム計測が進行中のとき、SPAM デバイスは変更されていない信頼されたシステム状態と一致するメモリイメージを提示する。しかしその他のすべての動作モードでは、SPAM デバイスはユーザーの望む通りに変更されたメモリイメージを提示する。この変更は非常に微妙なものでよい——セキュリティカーネルの重要な分岐命令を変更するのに必要なのは、適切な位置でのわずかなビットのフリップだ。

このデバイスは SPIOS より強力だ。電源が入っており信頼されているはずのシステム上で動作するためだ。より広範な DRM スキームや、ローカルマシンとサーバー間の認証済みトランザクションの一部に対しても有効に適用できる。ただし、SPAM だけでは、システムを別の登録済みトラステッドシステムとして偽って識別させることには使えない。SPAM は、ローカルマシンの耐タンパー性セキュリティ暗号モジュールと認証サーバーの間で共有された秘密情報を持たないからだ。偽のシステム識別は、耐タンパー性セキュリティ暗号モジュールから鍵を抽出するか (可能ではあるが自明ではなく、モジュールを破壊する可能性が高い)、別の登録済みトラステッドマシンの安全な暗号モジュールをなんとか騙して偽の識別情報を提供させるかのどちらかが必要だろう。

SPAM デバイスは比較的少ない費用で製造できる (現在、高性能 FPGA は 1 個単位で \$50 程度)、取り付けも非常に簡単だ。SPAM はメモリモジュールに直接統合されるか (その場合、信頼性侵害デバイスとメモリ拡張デバイスを兼ねる)、マザーボードのメモリスロットと既存のメモリデバイスの間にスタック構成で取り付けるデバイスとして提供される。一部のメモリカード構成、特にヒートシールドを採用しているものでは、SPAM デバイスを隠して通常のメモリ拡張デバイスとして通すことができるかもしれない。手の込

んだ方法ではあるが、トラステッド PC ベースのサーバーに高価値の秘密情報を保管している大企業や銀行に対する攻撃としては十分に価値があるかもしれない。

展望

トラステッドコンピューティングの見通しを考えると、まず現在提案されているスキームが約束するすべての利益をもたらすかどうかを検討し、それを消費者の権利への潜在的な損害や犯罪者にとっての潜在的な利益（強化されたプライバシーは善にも悪にも使える）と比較考量する必要がある。もしトラステッドコンピューティングがオンラインビジネスに完璧なセキュリティを提供できるなら、潜在的なリスクに見合うかもしれない。しかし本章で概説したシナリオは、トラステッド PC のセキュリティが完璧ではない可能性を示している。

Xbox を考えてみよう。Xbox はわずか \$50 のはんだ付け不要のモジュールでハックできるトラステッド PC 実装だ。これは Xbox に信頼できる秘密情報の価値についてかなり強い制約を置く。ハードウェアモッドチップはコピーされた 1 タイトル分——またはチップの取り付けを誰かに依頼する場合は 2 タイトル分——のゲームのコストでペイできるほど安価だ。

もちろん、コンテンツを盗むことの道徳的・社会的な意味もある。また、そのような行為を一部犯罪とすることを目指す DMCA などの新しい法律もある。残念ながら、現在提案されているトラステッド PC も同様に安価なハードウェア攻撃に対して弱い。したがって、高価値またはきわめて恥ずかしい秘密情報に必要なレベルのセキュリティを提供する可能性は低い。

実のところ、ハッキング技術は合法であるかどうかに関わらず、また意図が善であるか悪であるかに関わらず開発される。したがって、ハッキングについて人々を教育し、誰もが自分の「トラステッド PC」の限界を理解することが、消費者と企業双方の最善の利益になる。最悪のシナリオは、トラステッドコンピューティングに数十億ドルが投資されても、消費者にとって安全性やプライバシーがまったく向上せず、一方でコンテンツポリシーの貧弱な実装によって消費者の権利が大きく損なわれるという結果になることだ。

トラステッド PC の支持者にとって朗報なのは、集積回路の微細化が進み、PC 全体の統合が進んでいることだ。10 年以内に、今日の PC は 1 枚のシリコンに収まるようになるだろう。RAM と BIOS が完全に 1 枚のシリコンに統合されれば、システムのハッキングははるかに難しくなる——ただし不可能ではない。FIB (Focused Ion Beam、集束イオンビーム) 装置は、チップ設計者と故障解析ラボが使うツールで、ナノスケールの配線を切断・ジャンプできる。トラステッドマシン設計者にとってのもう 1 つのプラス面は、公開鍵プロセッサがチップに統合するほど小さく安価になりうることだ。そうなれば、個々のチップ、特にメモリチップが強力な暗号を使って I/O を認証・暗号化するようになり始めるかもしれない。

トラステッド PC の実装を助けるもう 1 つの技術は、耐タンパー性または改ざん検知機能の統合だ。たとえば、TDR (time-domain reflectometer、時間領域反射率計) をチップの I/O セルに組み込むことができる。TDR はワイヤの電気的特性への特定の変化を認

識することで、ワイヤ上の盗聴者の存在を検出できる。盗聴者を検出する能力に加え、統合 TDR デバイスは高性能 I/O にとっても望ましい——マルチギガビット速度の通信に必要な駆動インピーダンスと等化/プリエンファシスフィルタのキャリブレーションに使えるからだ。

おわりに

本書は Xbox ハードウェアハッキングの駆け足旅行に連れ出してくれた。はんだ付けと分解の基礎から最新のプロジェクトと技術まで。また Xbox ハッキングの社会的側面——ハッキングをする人々と、社会・法律・ハッキングの相互作用——も紹介してきた。

Xbox への LED の取り付け方の詳細は数年後には時代遅れになるかもしれないが、取り付けを実行することで身につくスキルは一生ものだ。さらに、ハッカーと消費者が直面している社会的・法的な問題は、Xbox を超えて、エレクトロニクスに満ち、コントロールされ、依存した僕たちの新たな情報中心の生き方のあらゆる部分にまで及んでいく。

本書の内容はただの出発点にすぎない。探求を待つハードウェアの世界がそこにある。本書が、エレクトロニクスがますます満ちあふれ、コントロールされ、依存されていく世界で、探求者・修繕者・革新者になるための強力な出発点を初心者読者に提供できていれば幸いだ。

ハッピーハッキング！

— bunny@xenatera.com

注

1 ケイパビリティのハードウェア実装の効率的な高性能実装については、Jeremy Brown、J.P. Grossman、Andrew Huang、Tom Knight によるテクニカルノート「A capability representation with embedded address and nearly-exact object bounds」(<http://www.ai.mit.edu/projects/aries/Documents/Memos/ARIES-05.pdf>) を参照。

2 Alastair J.W. Mayer による Burroughs B5000 の設計の分析については「The Architecture of the Burroughs B5000 — 20 Years Later and Still Ahead of the Times?」(<http://www.ajwm.net/amayer/papers/B5000.html>) を参照。

3 ペンシルバニア大学の Jonathan Shapiro が当初考案 (<http://www.eros-os.org/>)。

4 この種のセキュリティシステムの詳細については、Jeremy Brown と Tom Knight によるテクニカルノート「A Minimal Trusted Computing Base for Dynamically Ensuring Secure Information Flow」(<http://www.ai.mit.edu/projects/aries/Documents/Memos/ARIES-15.pdf>) を参照。

付録 A ハッキング機器の入手先

ハードウェアハッキングを始める際の心理的な障壁のひとつが、「ハッキングに必要な道具が手に入らない」という思い込みだ。Radio Shack でも部品や工具は少し揃っている

が、値段は高いし、今の小型化された部品に対応できる工具は少ない。この付録では、豊富な品揃えをリーズナブルな価格で提供しているコンポーネントおよびツールのベンダーをいくつか紹介する。また、本書のプロジェクトに必要な基本工具のJamecoの注文番号も掲載した。これらの番号をJamecoのセキュアな注文サイトに直接入力するだけで、すぐに始められる。

ホビイスト向けベンダー

コンポーネントや機材のディストリビュータは多種多様で、大企業を主な顧客とするものから、個人・ホビイスト・修理業者向けのものまで幅広い。大企業向けのディストリビュータを相手にすると、苦勞することが多い。そういったところは大口取引優先で担当営業が対応するため、小口注文の場合は総合カタログや価格・在庫情報を入手しにくいことがある。このセクションで紹介するベンダーはいずれも個人注文に対応しており、それらを合わせれば必要なパーツのほぼすべてが揃う。

ベンダー名	連絡先	特徴・備考
Digi-Key	www.digikey.com 1-800-344-4539（電話）1-218-681-3380（FAX）	正規メーカー品を幅広く揃える。ホビイスト対応、迅速なサービス。25ドル未満の注文には5ドルの手数料。ウェブサイトではリアルタイムの在庫・価格確認、データシート、パラメトリック検索が使える。
Jameco	www.jameco.com 1-800-831-4242（電話）1-800-237-6948（FAX）	ホビイスト向けのコストパフォーマンスに優れたベンダー。正規品とサンプラス品が混在。工具の品揃えも良く価格も手頃。20ドル未満の注文には5ドルの手数料。自作キットも豊富。
MCM Electronics	www.mcmelectronics.com 1-800-543-4330（電話） 1-800-765-6960（FAX）	家電サービス・修理向けの専門工具と部品、および汎用工具・コンポーネントを扱う。入手困難なセキュリティビットや交換部品も販売。手数料は一律2.95ドルで、インターネット注文には最小注文額なし。
Mouser	www.mouser.com 1-800-346-6873（電話）1-817-804-3899（FAX） orders@mouser.com	正規品コンポーネントの良好な品揃えを持つ工具・コンポーネントベンダー。ホビイスト対

		応、最小注文額なし、手数料なし。
Newark	www.newark.com1-800-463-9275（電話）	規模の大きい伝統的なコンポーネントディストリビュータ。幅広い品揃えだが、すべてが在庫されているわけではなく、一部の注文は数週間かかる場合がある。他のホビイスト向けベンダーで見つからないコンポーネントを探す際に便利。25ドル未満の注文には5ドルの手数料。
Fry's Electronics	www.frys.com	コンポーネントと工具をそれなりに揃えている家電量販店。カタログ通販が苦手な人が実際に足を運んで見て回るのに最適。カリフォルニア、テキサス、オレゴン、アリゾナ州のみ展開。
McMaster Carr	www.mcmastercarr.com	機械部品、金属材料、工作機械のベンダー。プロジェクトの仕上げるに使う板金素材やネジ類が揃う。ウェブサイトも充実していて使いやすい。
FindChips	www.findchips.com	自動パーツ検索エンジン。複数のディストリビュータを横断して部品を検索できる。ホビイスト向けのベンダーの多くがこの検索エンジンに登録されている。

表 A-1: コンポーネントおよび機材のベンダー一覧

機材注文フォーム

このセクションでは、Jameco で購入できる厳選した工具リストを紹介する。「基本」注文フォームには、Xbox を開けるのに必要なビットとドライバ類、および基本的なはんだ付け工具がすべて含まれている。「上級」注文フォームには、あると便利だが、予算が限られている場合は必要になってから個別に購入することも検討できる高めの工具が含まれている。

これらの注文フォームにある工具に加えて、第I部で説明するプロジェクトにはいくつかの小さな部品が必要になる。必要な部品のリストは、それぞれの章の導入部を参照してほしい。

基本注文フォーム

以下の表は、本書のプロジェクトを行うのに必要な基本工具の一覧だ。Jameco のウェブサイト www.jameco.com にアクセスし、左側にある黄色い「Quick Order」ボタンをクリックして、ここに掲載されている Jameco の部品番号を入力するだけでハッキングを始められる！（価格は変動することがある。）

説明	Jameco 部品番号	価格
16~30 ワット可変式はんだごて	116572	\$18.95
はんだごて用 1/32" コニカルベベルチップ	35326	\$4.49
60/40 Kester フラックスコアはんだ（小パック）	73576	\$2.59
2oz Kester ロジンペーストフラックス	73584	\$1.59
はんだ吸い取り線	175986	\$2.49
はんだごてチップクリーナー&コンディショナー	132986	\$4.95
26 ピース SAE ホビーツールセット	170069	\$14.95
ワイヤーストリッパー（22~30 ゲージ）	127870	\$4.95
合計		\$54.96

26 ピース SAE ホビーツールセットには、T-10、T-15、T-20 トルクスビットを含む、Xbox の分解に必要なビット、ドライバ、スクリュードライバがすべて揃っている。注文フォームに 1/32" コニカルベベルチップを追加したのは、付属のはんだごてに付いているチップが 1/8" という巨大なチゼルチップだからだ。1/8" チップは、小さな国をはんだ付けするくらいには使えるかもしれないが、ファインピッチの部品全体にはんだショートを量産する元凶にもなる。コニカルベベルチップとチゼルチップには素早い熱伝達を可能にする平らな面があるため、先細りのコニカルシャープチップより使いやすい。はんだごてのチップクリーナーとロジンペーストフラックスも含めた。少し値が張るが、作業がぐっと楽になる——信じてほしい。

上級注文フォーム

以下の表は、あると便利だが、ハッキングをたっぷりやるつもりがある場合にのみ購入を検討すべき高めの工具の一覧だ。

- はんだごてスタンドとスポンジは、少しでもはんだ付けをするつもりなら必須だ。スタンドに熱いこてを安全に置いておけるので、やけどや火災のリスクを減らせる。
- 1/32" セミチゼルチップではんだ付けしてみしてほしい。コニカルベベルチップより気に入るかもしれない。

- ・ 静電防止リストストラップは、特に静電気が発生しやすい服や靴を着用する傾向がある人には常に手元に置いておくべきものだ。
- ・ SMT 除去キットは、小さな SMT 部品を取り外すときに重宝する。
- ・ マルチメータは、いつでも使える万能な計測・診断ツールで、自宅や寮でさまざまな場面で活躍する。多くのモデルがあるが、ここにリストしたものは最も安い価格で最多機能を提供している。実際に試してみたが、よく使える。
- ・ ここに掲載されているワイヤーストリッパーとクリンプ工具は、電源ユニット交換プロジェクトに非常に役立つ。クリンプ工具はJamecoではなく Digi-Key から購入する。少し高めだが、10 ドル以下の工具と違い、成形クリンプダイが付いていて高品質なはんだ不要の圧着が可能だ。

説明	部品番号	価格
はんだごてスタンド	36329	\$4.25
はんだごてスタンド用スポンジ	134631	\$2.99
セミチゼル 1/32" チップ	35078	\$4.49
調節可能な静電防止リストストラップ	159257	\$7.95
SMT コンポーネント除去キット	141305	\$21.95
マルチメータ（電圧・抵抗・電流・静電容量・温度・周波数・ダイオード/トランジスタ）	177480	\$49.95
ワイヤーストリッパー（16～26 ゲージ）	127861	\$4.96
クリンプ工具（Digi-Key より注文）	WM9999-ND	\$36.19

付録 B はんだ付け技術

この付録では、はんだ付けの基礎と、より高度な表面実装はんだ付け技術について説明する。実際に手を動かしながら読み進めると最大の効果が得られる。基本的なはんだ付け練習用のスルーホール組み立てキットを幅広く扱っている Jameco Electronics (<http://www.jameco.com>) がおすすめだ。SMT 組み立て技術を練習したい場合は、TopLine (<http://www.topline.tv>) が部品ブランクを使ったエコノミー練習キットを提供しており、MCM Electronics (<http://www.mcmelectronics.com>) には実践的な Surface Mount Solder Practice Kit がある。（実際に使いたい SMT 部品を取り付ける前に、SMT はんだ付けスキルを練習しておくことを強くお勧めする。）

はんだ付けの基礎

基本的なはんだ付け技術はさほど難しくない。はんだごての先端を部品のリードと基板のパッドの間に差し込み、両方を十分に熱する。十分に温まったら、滑らかなフィレットができるまではんだワイヤをゆっくりと送り込む。実際には、はんだ付け箇所が千以上

ある典型的な基板を、はんだ不良をひとつも出さずにはんだ付けできるようになるには、少々の練習と経験が必要だ。

きれいなはんだ付けをするには、高温で溶けたはんだが対象物に「濡れる（ウェット）」必要がある。つまり、手はんだ付けの真の技術とは、はんだの濡れを確実に実現する方法を理解することにある。

溶けたはんだが金属を濡らしているかどうかは、見た目でははっきりわかる。うまく濡れているときは、溶けたはんだが表面張力を失ったように見え、溶けたはんだが光り輝き、作業面をなめらかに流れていく。逆にうまく濡れていないときは、はんだが鈍い光沢を帯び、外に広がらずにはんだごての先端に丸く固まってしまう。

フラックスを使う

はんだが金属に濡れない原因は、空気中の酸素や汚れ・油脂が金属と反応しているからだ。この場合、フラックスを塗布することでこれらの異物を分解できる。「フラックス（flux）」という言葉はラテン語の fluxus（流れる）に由来する。ほとんどのはんだには、はんだ付けのしやすさを高めるためのフラックスコアが内蔵されている。切断したはんだをよく見ると、はんだ合金に囲まれたフラックスコアが見えるはずだ。必ずフラックス入りはんだを使うこと。そうしないと、はんだが回らず苦勞することになる。電子工作向けに作られたはんだのほぼすべてにフラックスコアが入っているが、ホームセンターでは、フラックスなしの配管用はんだも売っているので注意が必要だ。フラックス入りはんだを加熱すると、気化したフラックスの煙が少し上がる。作業台の近くに小型ファンを置いて煙を吸い出し、吸い込まないようにしよう。

初心者がよく犯すミスは、フラックス入りはんだを過信することだ。フラックス入りはんだに含まれるフラックスだけでは、はんだが回るのに十分でないことが多い。その場合は、追加のフラックスを塗布する必要がある。生のフラックスは通常、液体またはペースト状で販売されているので、塗布は簡単だ。液体の場合は、爪楊枝を半分に折って先端をギザギザのままにし、折った端をフラックスに浸けると小さな一滴が先端に付く。広い面積に液体フラックスを塗布するには細先の絵筆が便利だが、使い終わったらすぐに洗浄すること——数日でベタベタになって使えなくなる。フラックスドロPPERも便利だが高価だ。フラックスドロPPERは先端に細いキャピラリーニードルが付いたボトルで、逆さにするとフラックスがゆっくりと滴る。ペースト状フラックスは、爪楊枝や固体ワイヤなどの端切れをフラックスペーストに浸けて塗布する。フラックスペンは、保管と塗布が一体になった便利で安価なパッケージで初心者に向いている。精度や品質は他の塗布方法に劣るが、たまに使うには便利だ。

多くのフラックスは使用後の洗浄が必要だ。フラックスは時間とともに硬化して将来の修理を困難にし、基板をゆっくりと腐食させ、水分を吸収して導電性になることがある。昔ながらのはんだフラックスは樹脂フラックスだ。樹脂フラックスの除去には引火性と毒性のある強力な溶剤が必要だ。そのため、僕は水溶性フラックスかノークリーンフラックスをお勧めする。水溶性フラックスは水で基板を洗い流すだけで除去できる。蒸留脱イオン水を使うのが理想だが、ほとんどの場合、温めた水道水でも十分効果的だ。大量の基

板をまとめて洗浄する必要があるときは、食洗機に入れてしまう（食べ物のくずはきれいにして洗剤は使わずに！）。洗浄後は、基板を導電性のきれいなクッキングシートまたはアルミホイルの上に置き、低温（約 200°F＝約 93°C）のオーブンで 1～2 時間、水分が完全に蒸発するまで焼く。オーブンの温度を上げすぎないこと——部品内部の細孔に入り込んだ水が蒸気になってクラックや破裂を引き起こすことがある。ほとんどの部品は「プロセスシール」されているので水で洗っても問題ない。ただし、コネクタやスイッチには注意が必要だ。水による汚染を防ぐためにカプトンテープで覆っておく必要があるかもしれない。

[!WARNING] 基板に酸性フラックスを使用しないこと。基板や部品を腐食させ、時間がたってから故障を引き起こす。初心者がよくやる失敗は、配管用のはんだやフラックスを電子工作に流用してしまうことだ。これをやると、あとで酸性フラックスによる腐食に悩まされる。

最初のコツ

リードが多い部品は、はんだ付けする前に位置を合わせて仮固定しておく必要がある。表面実装型部品の場合は、デバイスの対角にある 2 ピンをはんだ付けして仮固定できる。仮固定したら、仮固定中に部品がずれていないか再確認する。スルーホール型部品の場合は、基板を裏返している間に固定するためのマスキングテープが必要だ。部品の角のリードを仮固定し、全リードをはんだ付けする前に部品が基板に対して水平であることを確認する。（マスキングテープは少し遊びがあるため、部品を基板に対して水平にするために、仮固定した角の一つを加熱しながら部品を押さえる必要があることが多い。）

カプトンテープは、作業台のそばに置いておくと重宝する。DuPont の Kapton は 500°F（約 260°C）程度まで耐えられるので、溶けたはんだが触れそうな場所を一時的に保護するのに使える。はんだを付けたくない周囲の部品やパッドをマスクする用途に便利だ。ただし高価なので、普通の固定や仮止めには使わず、熱や溶けたはんだにさらされる場所にだけ使うのがよい。

温まっていない金属にははんだが回らない。これは大きなはんだ付け箇所や、大きな銅面に接続されたはんだ付け箇所がよく起きる問題だ。このような場合、接続された金属が熱を十分に伝導してしまい、接合部がはんだの融点に達しない。解決策は、より強力なはんだごてを使うか（ただし、非常に大きなはんだごての熱が基板の配線を剥がしてしまうこともあるので注意）、はんだを加える前にはんだごてを接合部に長めに当て続けることだ。作業エリアの加熱を早めるコツのひとつは、はんだごてがものを温めている先端にごく少量のはんだを送ることだ。基板ではんだは濡れないが、ごての先端に付いた液体はんだがごてと基板の実効的な接触面積を増やし、基板への熱伝達をより速くする。ごての先端にはんだを加えた後、はんだ付け箇所を維持したままごてを回転させて溶融はんだを基板に接触させる必要がある場合もある。

部品のリードや基板のランドが十分に加熱されているかは、反射光の変化を注意深く観察することでわかる。基板上の部品のリードやランドには通常、はんだなどのめっきが施されている。このはんだめっきは通常の状態では少し鈍い光沢を持つ。しかし、十分に加熱されると、光沢が鈍いものからほぼ完全な鏡面反射に変わる。この感覚をつかむには、

上述した技術を使って大きな正方形のランドをはんだごてで加熱してみるといい。はんだごてが基板を加熱するにつれ、溶融前縁がランドを横切って広がっていく様子が観察できるはずだ。

[!WARNING] 温度調節機能のないはんだごてを使う場合は、作業を完了できる範囲で最も低ワットのはんだごてを使うこと。過剰な熱で銅配線が基板から剥離することを防ぐためだ。

表面実装はんだ付け

表面実装はんだ付けのスキルを習得するには、少しの忍耐と練習、そして適切なツールが必要だ。基本的なはんだ付けセットに加えて必要な基本ツールは、ピンセットと拡大レンズだ。

良質な細先ピンセットは、表面実装部品に不可欠なはんだ付けの工具だ。ピンセットははんだ付け中の表面実装部品の安全な取り扱いに必要で、小さな部品はすぐに熱くなって指を火傷するほどの温度になる。また、ピンセットは小さな部品を所定の位置に保持するためにも必要で、溶けたはんだが固まるときの表面張力で部品が引っ張られて動くのを防ぐ。ピンセットの先端は、扱う予定の最も細かい表面実装部品のピンの間に入れるのに十分なほど小さくしなければならない。そうすれば、はんだ付けと検査の際に個々のピンを操作するためにピンセットを使える。

ピンセットにはさまざまなグレードがある。グレードは先端の鋭さ・品質・耐久性、先端の位置合わせ精度、ばねの動作に基づいて決まる。高グレードのピンセットは少し高価だが、SMT はんだ付けを多くやるつもりなら十分に投資する価値がある。Future-Active Electronics (<http://www.future-active.com>) のような量産部品の流通に特化した会社では、良質なピンセットを揃えている。

表面実装はんだ付けで最も難しいのは、作業対象をよく見ることだ。人間の目は、裸眼では見えないほど小さなものを繰り返し操作できる能力を持っている。はんだ付けの理想的な拡大ソリューションは、生物標本の検査に使われるような光学立体顕微鏡だ。しかし残念ながら、これらの顕微鏡は非常に高価で、良いモデルは中古車1台分ほどの値段がする。

より安価な解決策は卓上型拡大レンズを使うことだ。多くの製図・美術用品店ではこの種のレンズを販売しており、ほとんどのオフィス用品店でも拡大レンズ付きランプを少なくとも1種類販売している。これらの拡大レンズは組み立てを助けてくれるが、はんだ付けの徹底的な検査には不十分だ。完成したはんだ付けの検査には、宝石鑑定用ルーペや校正用拡大鏡のような比較的安価な高倍率ルーペを使うべきだ。

単純な部品の手順

抵抗器、キャパシタ（コンデンサ）、インダクタ、トランジスタやダイオードなどの小型半導体デバイスのような単純な表面実装部品は、基板への取り付けが容易だ。これらの部品の取り付け手順を見ていこう。

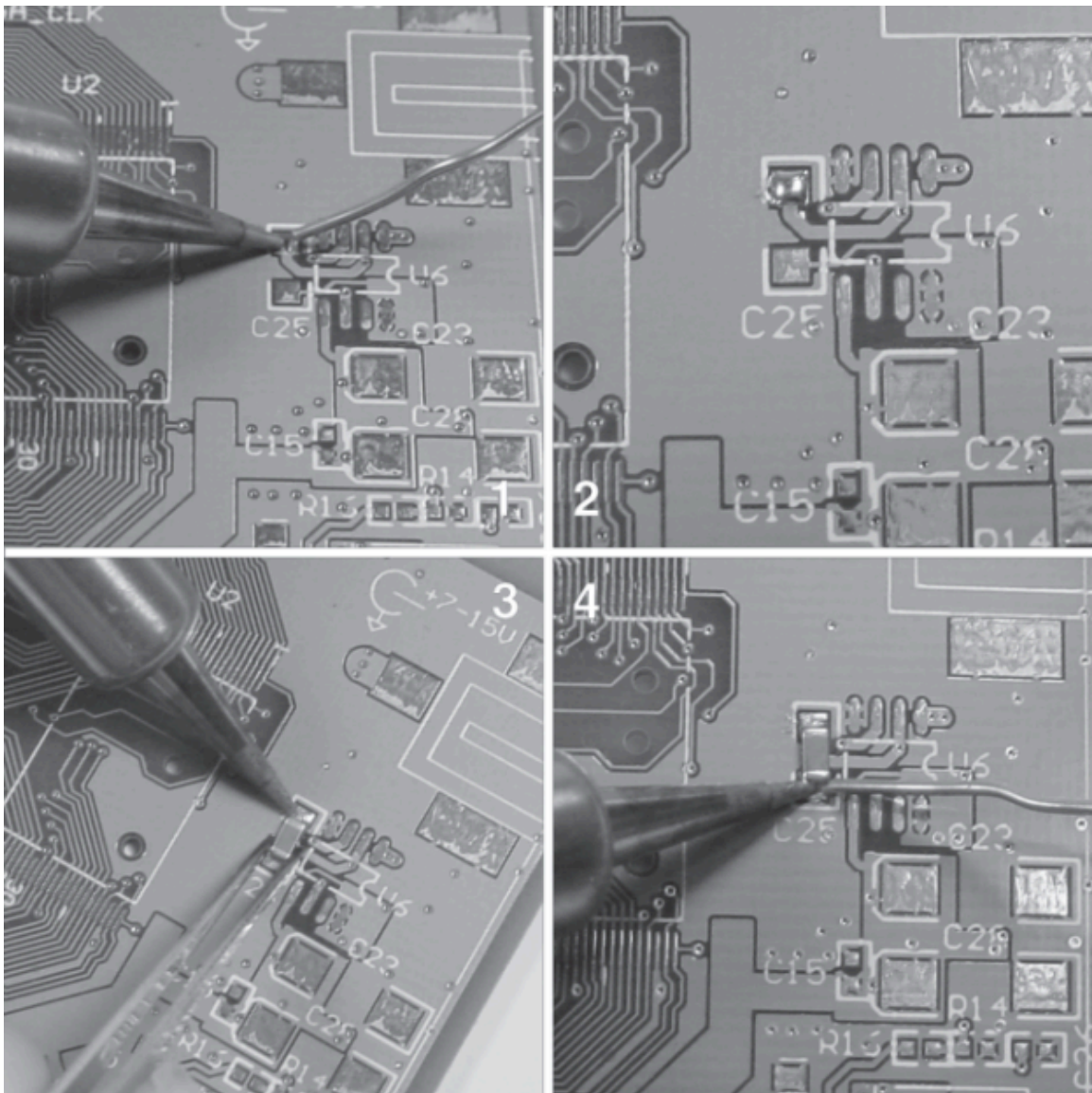


図 B-1: (1) 対象部品のランドの一つにはんだを少量塗布する。この例では対象部品はC25。(2) はんだを盛ったあとの状態。(3) ピンセットで部品を位置合わせし、最初に盛ったはんだが部品のリードに流れるまではんだごてで熱を加える。(4) 残りの部品のランドをはんだ付けする。

まず、部品のランドの一つにはんだを盛り、ピンセットを使って部品をランドの上に置いて位置合わせする。部品の配置に問題がないと判断したら、盛ったはんだが溶けて部品がランドの上に沈み込むまで熱を加える。アライメントを微調整しながら熱を加え続ける。熱を離してはんだが冷えて固まるのを待つ。部品のアライメントを再確認してから、残りの部品の接点を基板にはんだ付けする。最初のはんだ付け部が鈍く見えたり弱そうに見える場合は、他のピンをすべてはんだ付けした後で、少量のはんだを加えながら再

加熱する（図 B-1 参照）。この手順は 2～3 ピンの部品や、50mil(1.27mm)ピッチの SOIC パッケージデバイスのような広いピッチのマルチピン部品に有効だ。

これらの部品をはんだ付けする際に最も難しい状況は、部品のリードの一つ以上が電源プレーンのような広い銅面に接続されている場合だ。このような場合、はんだが基板のランドに濡れて密着するまでに多大な熱を加える必要がある。はんだが酸化して丸い塊になり基板との電氣的接触が悪くなることがあるので注意しよう。これが起きたら、フラックスを少量追加して溶けたはんだの濡れ性を高めよう。

[!TIP] 「乾いた」はんだごての先端は、大きな銅箔や電源プレーンに接続されたランドを加熱するのに苦労する。はんだごてからの熱伝達率を上げるひとつの方法は、ごての先端に溶けたはんだの塊を乗せた状態で基板に触れることだ。この塊が広がって基板を局所的に加熱し、より効果的に熱を伝達する良好な熱接触を確立する。これにより、さらにはんだを加えたときによりきれいなはんだ付けができる。

複雑な部品の手順

今日、ほとんどの IC 部品は何らかの細ピッチ表面実装パッケージになっている。このようなパッケージは最初は敷居が高く感じるかもしれないが、いくつかの簡単な手順とコツを少し練習するだけで、高い確信を持ってこれらのパーツを取り付けられるようになる。細ピッチ表面実装部品の取り付けプロセスは単純な表面実装部品の取り付けに非常に似ており、典型的な中程度のサイズの表面実装パッケージなら数分以内に完了するはずだ。

最初のステップは、チップの 2 つの対角にある各 1 ピンをはんだ付けして基板にチップを仮固定することだ。最初の試みでアライメントが正しくない場合は、2 つのコーナーのうち 1 つを加熱しながらチップを希望する方向に押すだけで簡単に修正できる（図 B-2 参照）。このアライメントの精度には十分注意すること——位置ずれしたチップはチップのはんだ付けに際して際限のない苦労を招く。

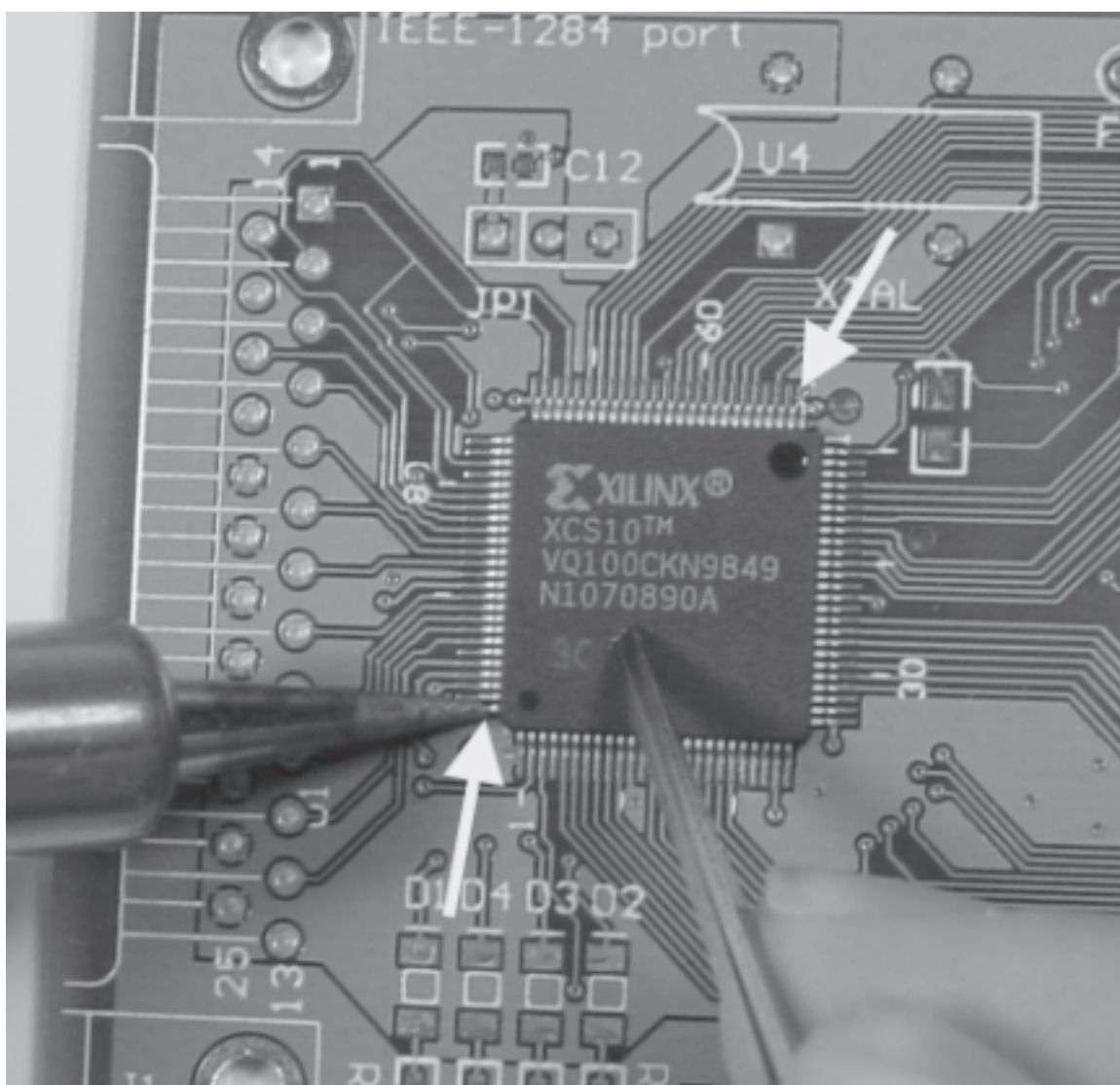


図 B-2: まず、チップの対角にある2つのコーナーにそれぞれ1ピンをはんだ付けする。この図の矢印はこの例で使用するコーナーを示している。

次のステップは、チップの周りに薄いはんだフラックスの膜を塗布することだ。これにより、チップのピンへのはんだの濡れ性が高まり、ショートを引き起こすリード間ではなく部品ランドにはんだが引き込まれる（図 B-3 参照）。

[!TIP] 基板レイアウトの際には、細ピッチ表面実装部品のランドを少し長めに設計すること。ランドを長くすることで手はんだ付けが楽になるが、配線のルーティングが少し難しくなり基板サイズが若干大きくなる。余分なランドの長さがチップのリードから余分なはんだを引き込んで逃がす役割を果たし、はんだブリッジが発生しにくくなる。

すべてのリードにはんだフラックスが均一に塗布されたら、はんだごての先端に少量のはんだを乗せ、それをはんだ付けされていないリードに押し当てる。はんだが部品のリー

ドの下や周りの空間に浸透していく。すべてのリードにはんだが塗布されるまでこの作業を繰り返す。この時点で過剰なはんだが複数のピンをブリッジしていても気にしないこと。すべてのピンのはんだ付けが終わったら、銅のはんだ吸い取り線（ソルダーウィック）を使って、図 B-4 に示すようにはんだブリッジを除去する。

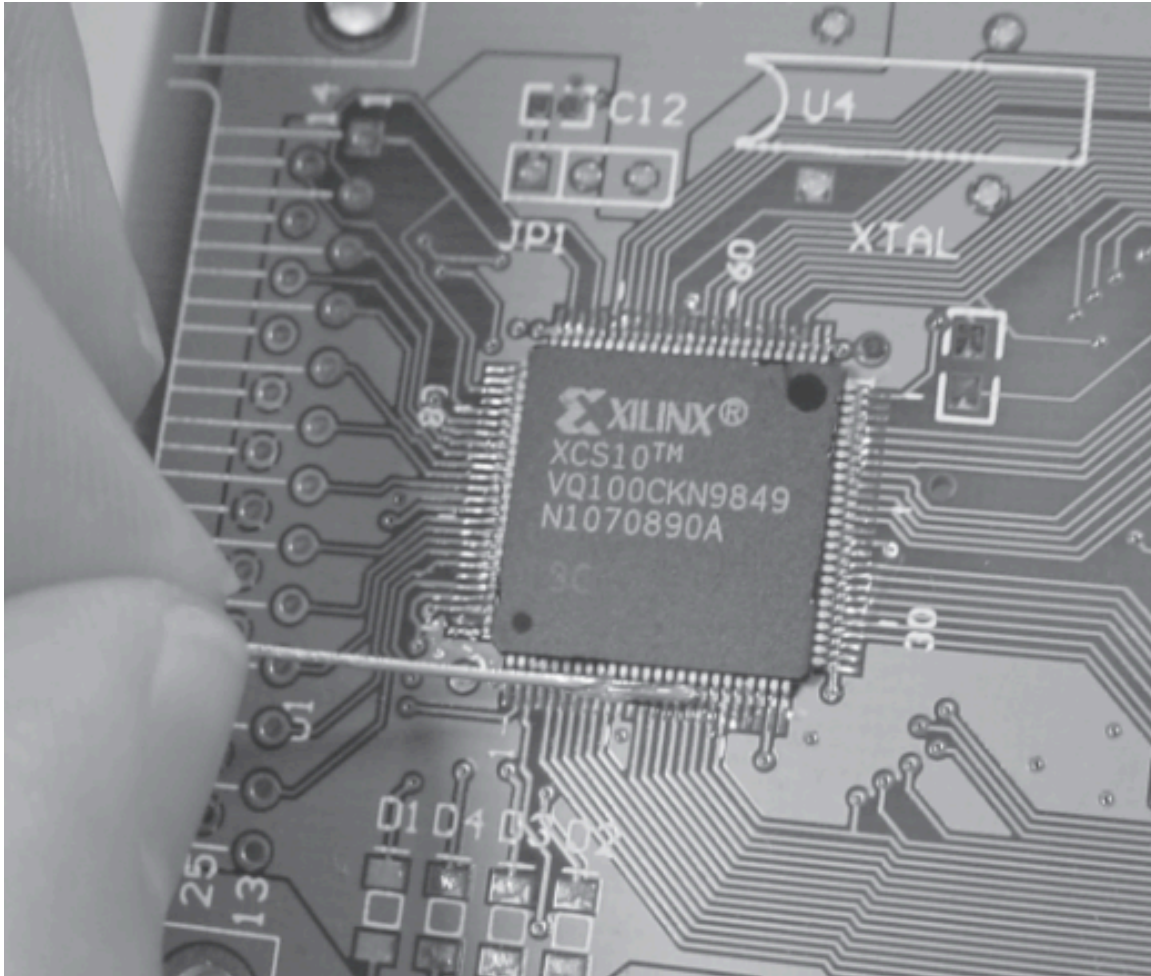


図 B-3: チップ周りのピンに薄いはんだフラックスの膜を塗布する。この図では、フラックスの容器に何度もつけた端切れワイヤを使ってペーストフラックスを塗布している。

ほぼ完成だ。最後に、すべてのピンがしっかりとのはんだ付けされていることを確認する。顕微鏡なしに目視でこの検査を行うのは難しい。代わりに、ニードルまたはピンセットの先端でピンを引っ張り、図 B-5 に示すようにしっかりとした一定の力でピンに沿ってピンセットを動かす。はんだ付けされていないピンは少し動く。動いた場合は、ピンを元の位置に押し戻し、そのピンに少量のはんだを加えて修復する。

[!CAUTION] 基板のはんだ付けが終わったら、必ず電源ショートを確認すること。はんだブリッジの中には顕微鏡的な樹枝状構造のものや、チップの下のピンの裏側に形成されるものがある。マルチメータを抵抗測定モードに設定して電源ショートを確認する。導通チェック

モードは使わないこと——大型高速コンピュータ基板では電源レール間に数オームという低抵抗があることが多く、多くの導通チェッカーで短絡として検出されるほど低い場合がある。また、電源ライン間の抵抗を測定する際には、測定値が安定するまで1~2秒待つこと。基板の電源ラインに載っている大型キャパシタが充電される間、電源レール間の初期抵抗は非常に低くなる。

部品がはんだ付けされたら、イソプロピルアルコールなどの穏やかな（低刺激性の）溶剤をつけた綿棒で余分なフラックスを洗浄する（図 B-6 参照）。洗浄は目視検査を容易にするためだけでなく、デバッグ中のチップピンのプロービングも容易にするため重要だ。また、多くのフラックスは時間とともに硬化して汚染物質を閉じ込め、将来の修理の妨げになるためでもある。

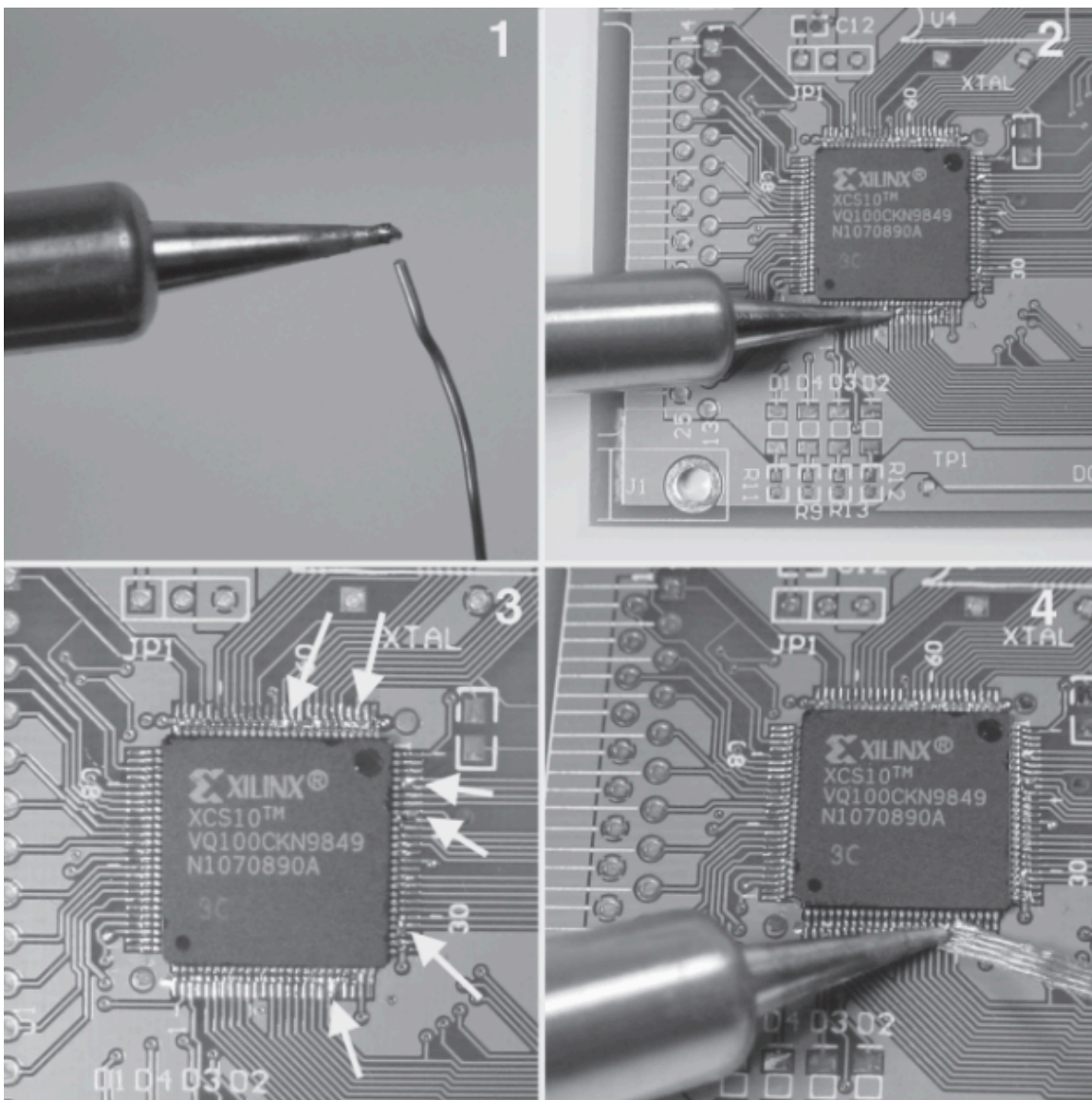


図 B-4: (1) 少量のはんだを熱いごての先端に溶かし込む。(2) このはんだを部品のピンに転写する。すべてのピンがはんだ付けされるまで (1) と (2) を繰り返す。(3) 矢印で示したはんだブリッジが多くのピンに形成されている。(4) 銅のはんだ吸い取り線を使ってはんだブリッジを除去する。

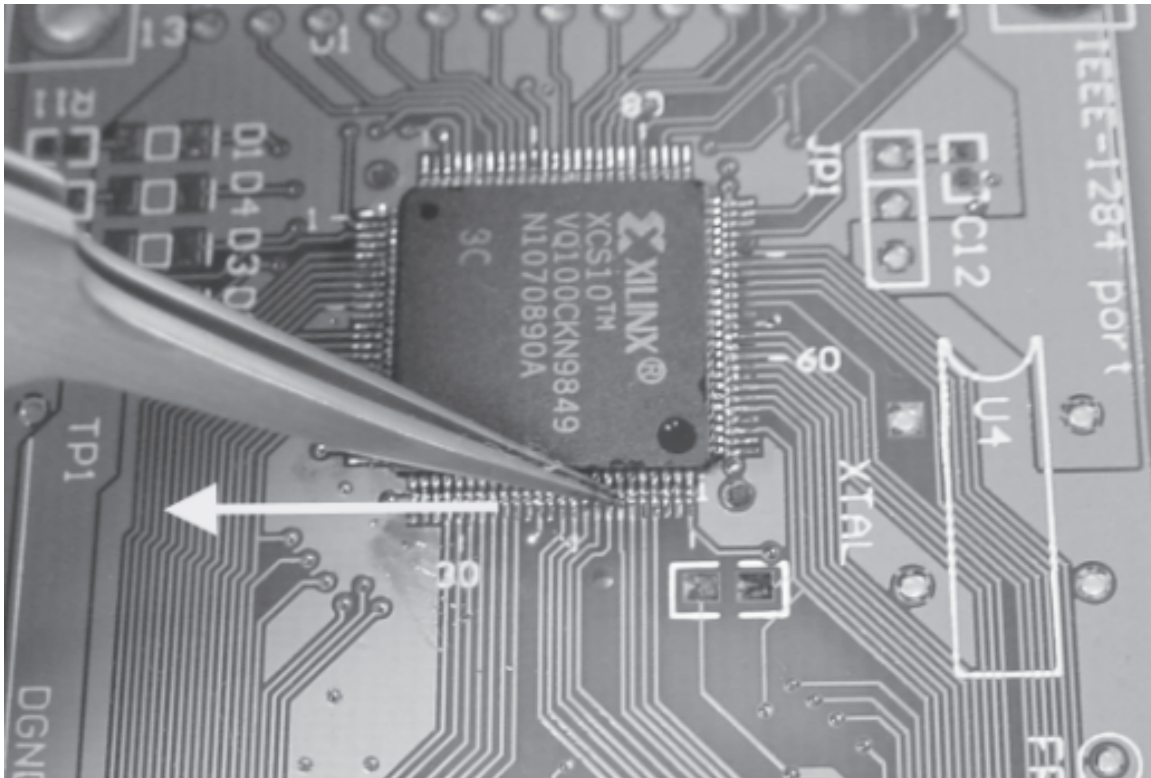


図 B-5: ピンセットまたはニードルを使って、しっかりとした一定の力でチップの側面に沿ってピンをなぞる。しっかりはんだ付けされていないピンは少し曲がる。矢印は動作方向を示している。

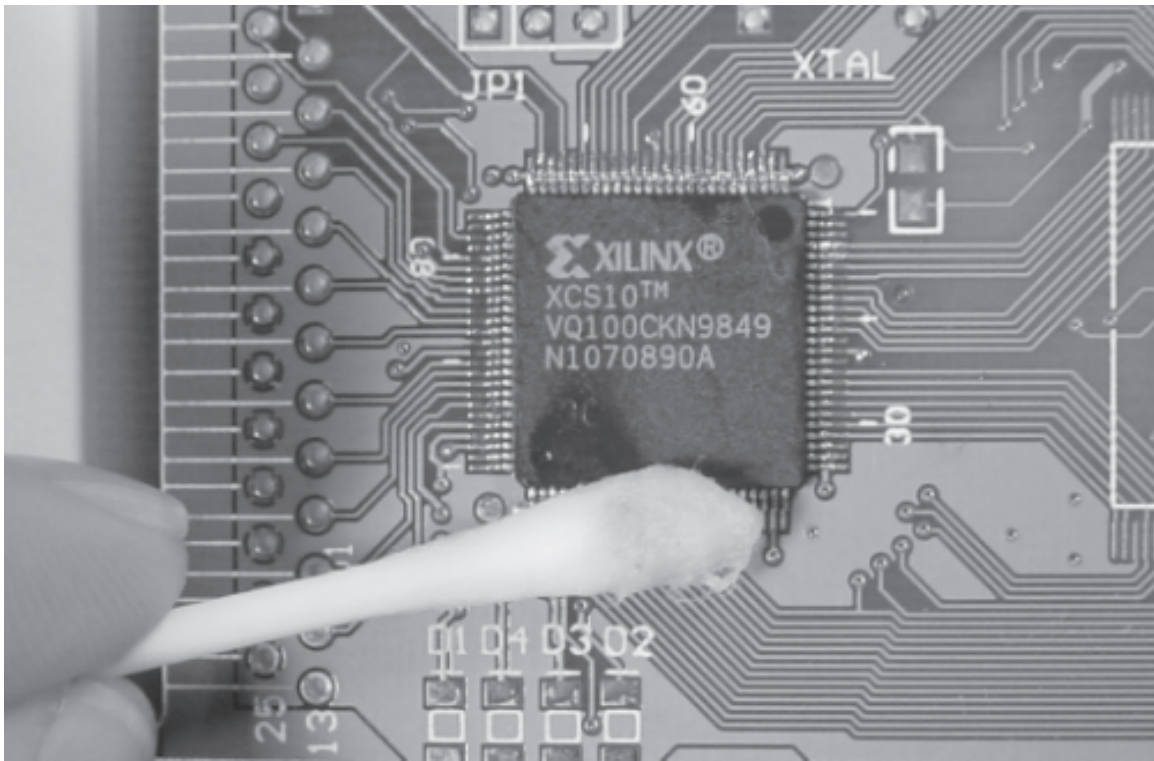


図 B-6: 穏やかな溶剤に浸した綿棒でフラックスの残留物を洗浄する。

部品の取り外し手順

表面実装部品の取り外し技術は多数あり、その多くはピンセット型はんだごてやホットエアガンなどの特殊ツールを必要とする。ピンセット型はんだごては、特に基板が密集してパッケージされている場合に、小型の表面実装部品の取り外しに適している。かなり素早く効率的で、適切なピンセット型はんだごてを使えば部品のピンを比較的良好な状態で保てる。しかし、かなり高価で、チップを取り外す際の適切なタイミングと圧力を把握するには練習が必要だ。

より簡単で安価な解決策は、リノリウムの床タイルの除去のためにホームセンターで販売されているようなヒートガンを使うことだ。ヒートガンは基板の広い領域を加熱し、部品はリードへのダメージが最小限でランドから浮き上がってくる。この技術の欠点は、あまり精密ではないため、再使用する予定の基板からチップを取り外すには理想的ではないことだ。そのため、ヒートガンは壊れた基板から良いチップをサルベージ（回収）するのに最も効果的だ。ヒートガンを使うもうひとつの問題は、熱が基板を反らせたり、パッケージ内に水分を吸収したチップに破壊的な故障を引き起こしたりすることだ。「ポップコーン現象」と呼ばれるこの故障モードは、チップ内部に閉じ込められた水分が沸騰しても逃げ場がなく、圧力が積み上がって最終的に破壊的な放出として終わるときに起こる。湿った環境で動作していた、または水で洗浄した基板やチップは、ヒートガンではんだ取り外しをする前に 200～250°F（約 93～121°C）のオーブンで数時間焼いておくべきだ。

はんだを再合金化してはんだが非常に低い温度（300°F＝約 149°C以下）で溶けるようにするという、特にホビイストにとって魅力的なオプションもある。Chip Quik (<http://www.chipquikinc.com>) がこの技術の特許を持っており、除去キットは Jameco（注文番号 141305）を含む多くの販売業者から購入できる。この技術は特別な機器が不要で、図 B-7 に示すように簡単で安全なため魅力的だ。

まず、チップのピンにはんだフラックスを塗布し、取り外す部品のピンにチップ除去合金を溶かし込む。これにより、チップの周り全体に低融点合金の大きなビードが形成される。次に、はんだごての先端を合金のビードに沿って引きずるようにして合金全体を加熱する。チップと合金に蓄えられた熱がチップ全体を十分長い時間溶融状態に保ち、チップをランドから簡単にスライドさせて外せる。最後に、はんだごてで加熱して綿棒で拭き取ることで、基板とチップの両方から低融点合金を洗浄する。合金は基板とチップの両方からかなり簡単に拭き取れる。

この再合金化技術は取り外したチップのピンの完全性と基板のランドを保護する。欠点は、合金の洗浄が少し面倒で、隣接するチップのピンの間に合金の微粒子が詰まることがある点だ（洗浄時に注意することでこの問題は防げる）。もうひとつの欠点は、チップを取り外すたびに除去合金を消費するため、多数のチップをこの合金で取り外すと長期的にコストがかかることだ。幸い、チップの取り外しはほとんどの趣味家にとってかなりまれな作業だ。

「ボールグリッドアレイ（BGA）」パッケージは、高い電气的性能と高密度のため普及が進んでいる表面実装パッケージの一種だ。このパッケージは、パッケージの下の大きなはんだボールアレイを通して基板に取り付けられる。明らかに、この種のパッケージは従来のはんだごてでははんだ付けできない。さらに、BGA パッケージはほとんどののはんだ接合部がパッケージの下深くに埋もれているため検査が非常に難しい。これらの部品を取り付ける唯一の実用的な方法は、すべてのはんだボールが溶けるまで基板と部品全体を加熱するオーブンを使うことだ。通常、これらの BGA パッケージの位置合わせは何らかの機械の助けを借りて行われ、BGA パッケージの検査は X 線または超音波イメージングで行われる。

これらの種類のパッケージを取り付けるのに必要な設備は非常に高価なので、ホビイストが BGA パッケージデバイスを取り付ける最良の方法は、専門の組み立て会社に依頼することだ。Naprotek (<http://www.naprotek.com>) のようなクイックターン少量生産のプロトタイピングを専門とする会社は、BGA パッケージデバイスの取り付けをかなり合理的な料金で行ってくれる。BGA を取り付けてもらうときは、必ず部品の検査写真のコピーを要求すること——写真による安心感は追加コストに値する。BGA パーツの取り外しや BGA リボールも可能だが、このプロセスは取り付けよりかなり高くつくことがある。

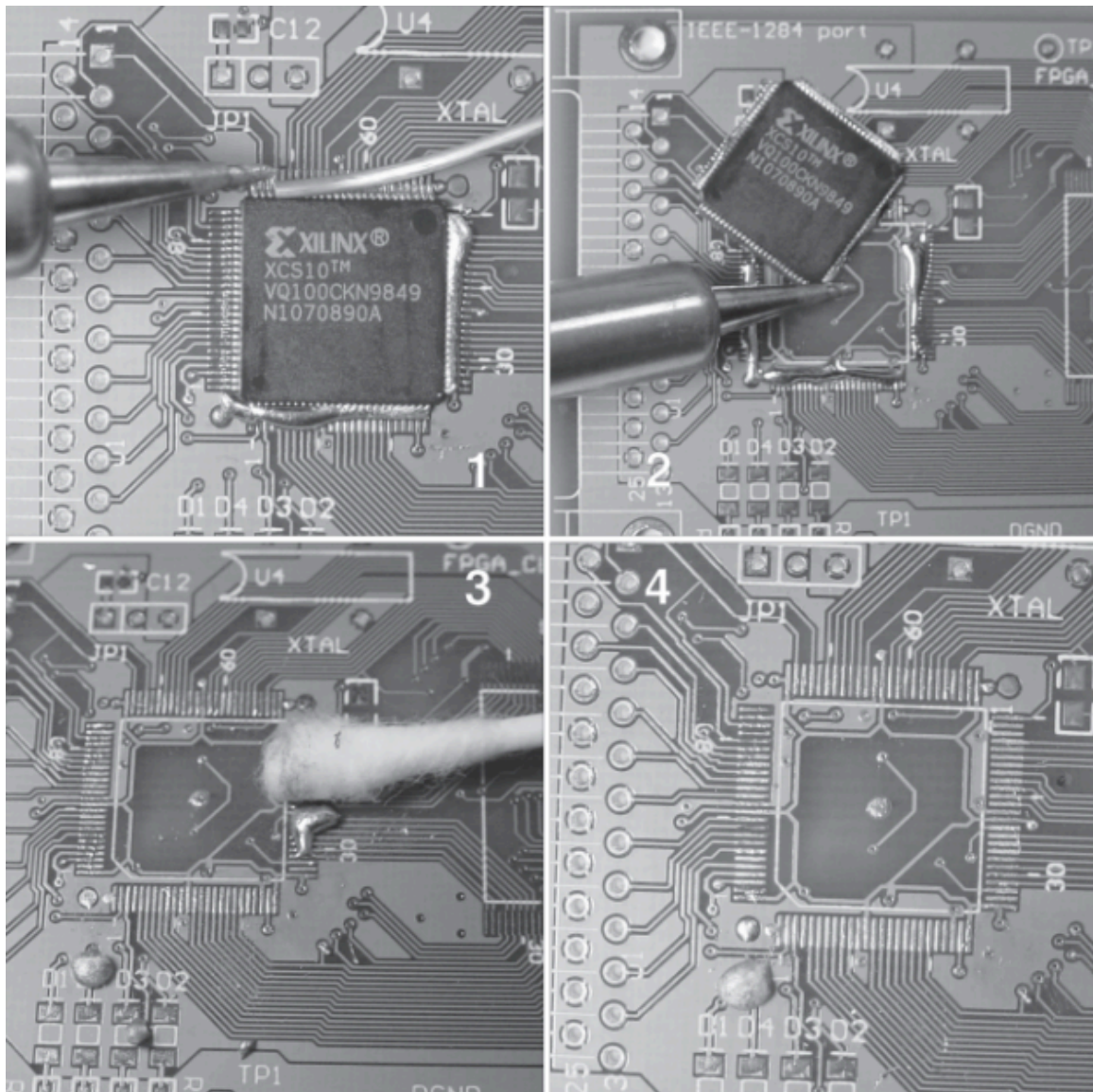


図 B-7: はんだを再合金化してチップを取り外す。(このプロセスを始める前にチップのピンにはんだフラックスを塗布すること。)(1) 再合金化化合物をチップのすべてのピンに溶かし込む。再合金化化合物はワイヤ状になっている。(2) はんだごての先端を金属のビードに沿って引きずるようにして、チップ周りの金属の塊を加熱する。再合金化されたはんだすべてが溶融状態になったら、チップをランドから押してスライドさせられる。(3) 加熱して綿棒で拭き取ることで、基板とチップの両方から再合金化化合物を洗浄する。(4) 結果: きれいなチップの取り外し。

付録 C PCB レイアウト入門

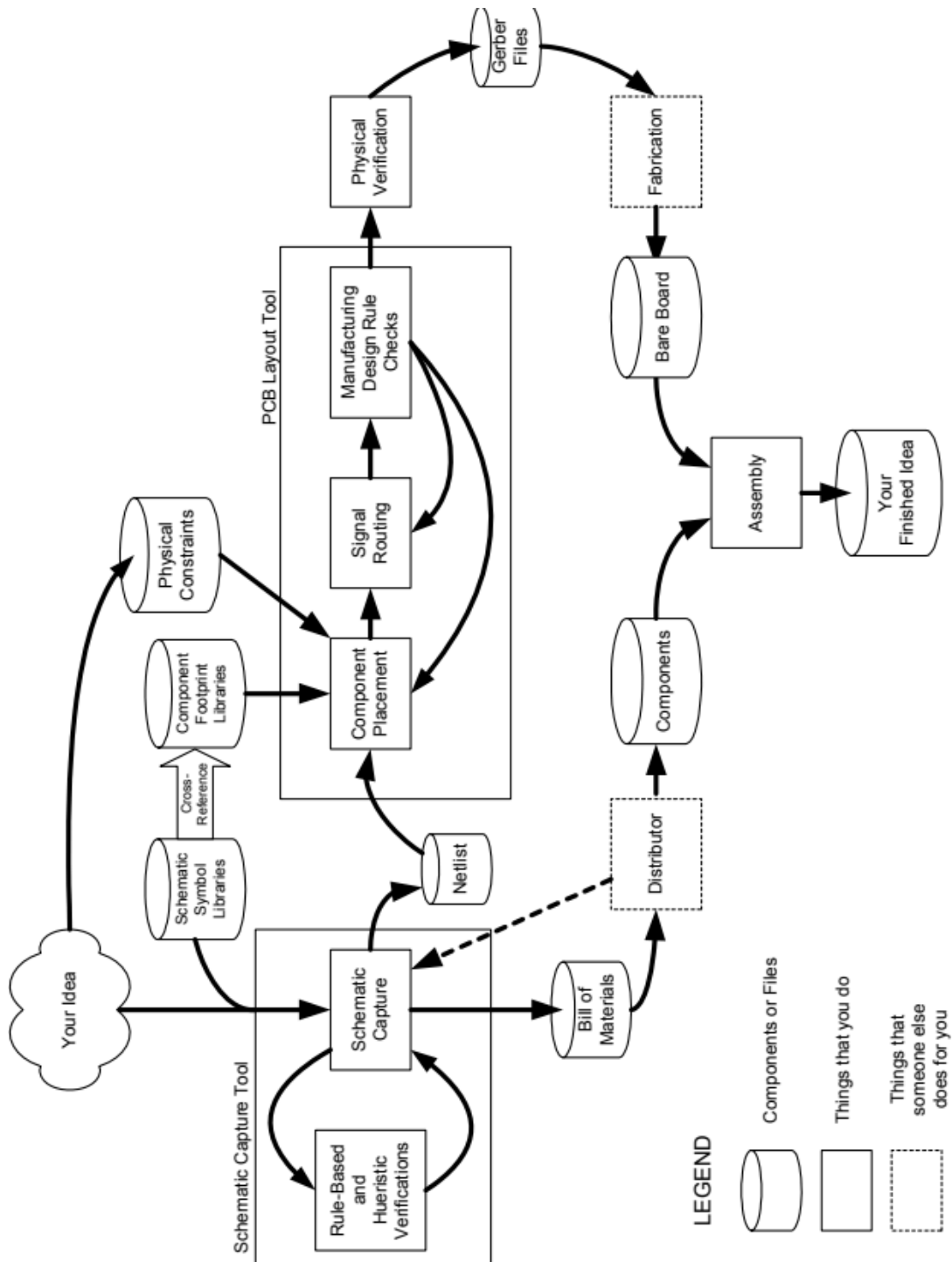


Figure C-1: The board design process, from idea to finished product.

図 C-1: アイデアから、(誰かが代わりに製造して) 完成した製品として届くまでの、基板設計の全工程。

設計ルールチェッカー(DRC)は回路図のエラーを検出するのに役立つが、実行されるチェックはたいいてい基本的なもので、最も明らかなミスしか見つけられない。一般的なチェック項目には、部品指定番号の重複、未接続のネット、フローティング入力などがある。

従来の DRC に加えて、独自の簡単なチェックを設けることもできる。たとえば、多くの設計ツールは「部品同士の接続関係(ネットリスト)の接続数チェック」を行わないが、これは Perl スクリプトやスプレッドシートを使えば比較的簡単に実装できる。このヒューリスティックなチェックは、各ネットリストに接続されているコンポーネント数を集計し、接続数でソートするものだ。ソートされたリストの先頭に来るのが「ピン数1のネット」だ。ピン数1のネットは、ほぼ例外なく回路図入力時のタイプミス(ネット名のスペルミス)を示している。ソートされたリスト全体をざっと見渡して、各信号グループが同じ数の接続を持っているか確認するのも有効だ。ほとんどの信号バスでは、バス内のすべての信号が同じ数の接続を持つ。このようなネット接続数チェックツールは半日で作れるが、それによって見つかるバグのおかげで、数え切れないほどの時間とお金を節約できるだろう。

回路図を PCB レイアウトツールにエクスポートする前に、回路図の部品表(BOM)に記載されているすべての部品を発注しておくべきだ。重要な部品が品薄になっていることがあり、その場合は不足を補うために回路図を再設計しなければならない。多くの場合、必要な再設計は部品のフットプリント指定を変更するだけで済む。メーカーの在庫不足を考慮してあらかじめ回路図を修正しておくことで、完成した PCB レイアウトの段階で変更を加えるという困難な作業を回避できる。

PCB レイアウト

PCB レイアウトプログラムへの入力は、部品のフットプリント情報が付与された部品同士の接続関係(ネットリスト)だ。ネットリストは、すべての部品とピン、およびその接続関係を中間的に表現したものだ。ネットリストの抽出は自動化されているが、回路図シンボルと PCB フットプリントの対応関係は必ずしもうまく管理されているとは限らない。

シンボルとフットプリントの対応が難しいのは、一つの部品に対して複数のパッケージが存在するためだ。たとえば、トランジスタのシンボルは、小型の SOT-23 パッケージのデバイスも、大型の TO-3 パッケージのデバイスも同じように表すことができる。ネットリスト抽出時に適切なパッケージが選択されているかどうかを確認するのはあなた自身の責任だ。すべてを一度に確認するよりも、部品を回路図に配置した時点でフットプリントを確認する習慣を持とう。ネットリスト変換時や、さらに悪いことには配置・最終設計レビュー時にまとめてチェックするのは避けたい。

ネットリストが完成したら、PCB レイアウトを開始する準備ができています。

PCB レイアウトプログラムへのもう一つの外部入力、設計ルールだ。設計ルールは基板製造会社によって設定されており、最小配線幅、配線間の最小スペース、最小穴径、スルーホール最小アニュラーリング、電源レイヤーおよびルーティングレイヤーの枚数などの仕様が含まれる。具体的な設計ルールは選択するプロセスによって異なり、さらに予算に

よっても左右される。最高水準のプロセスでは2ミル（1ミルは1000分の1インチ、すなわち25.4マイクロメートル）という細さの配線や、レーザードリルによるブラインドビア・埋め込みビアも利用できるが、製造費用は一般的なホビイストの予算（100ドル未満）をはるかに超える。ホビイストが利用しやすいプロセスは、6ミルの配線/スペース設計ルール、最小仕上がり穴径15ミル、銅箔2層または4層が一般的だ（基板製造会社のリストはこの付録の後半に掲載する）。

PCBレイアウトは、配置（プレースメント）と配線（ルーティング）の2段階で構成される。部品の配置を適切に行うことで、ルーティング作業が大幅に楽になる。一般的な目標は、ノイズ・遅延・信号損失を最小化するために、すべての部品をできるだけ接続が短くなるよう、ビアの数を最小限に抑えながら配置することだ。コネクタ、スイッチ、電源部品などは配置場所がほぼ決まっており、選択の余地が少ない。残りの部品については、設計の理解が最適な配置の判断に役立つ。

配置が完了したら、1対1のスケールで設計を印刷し、実際の部品をその上に当ててみることで、部品がそれぞれのフットプリントに収まるかどうかを確認しよう。ソケットと組み合わせて使う予定の部品がある場合は、部品単体ではなくソケットを使って1対1の確認を行うこと。ソケットは部品本体よりも多くのスペースを必要とするからだ。このチェックにより、すべての部品が正しいパッケージタイプかどうか、部品の輪郭がすべて正確かどうか、組み立てが容易になる十分なクリアランスが各部品間にあるかどうかを保証できる。1対1プロットで確認するもう一つの重要な点は、すべてのコネクタの向きとピン配列だ。コネクタを逆にしまったり、コネクタのオス・メスが逆のフットプリントを使ってしまうことは非常によくある。また、特にファインピッチの表面実装（SMD）リードを持つチップを扱う際には注意が必要だ。リードを曲げないように気をつけ、ESD（Electro-Static Discharge、静電気放電）対策の手順を必ず守ること。

一般的な配置・配線のガイドライン

以下は、配置と配線に関するガイドラインの簡単なリストだ。これらはあくまでも一般的な提案であり、当然ながら適用できない状況もあるだろう。

表面実装デバイスのビアファンアウト領域を確保する

表面実装デバイスは、かつてデファクトスタンダードだったスルーホール部品と比べて、高密度実装が可能という大きな利点を持つ。しかし、表面実装部品も、特に複雑な設計やオートルーティングを使った設計では、配線のためにスルーホールビアが必要だ。このようなルーティングビアは、SMDランドの「ファンアウト」ビアと呼ばれる。図 C-2 と図 C-3 は、表面実装部品のファンアウトビアの使用例を示している。

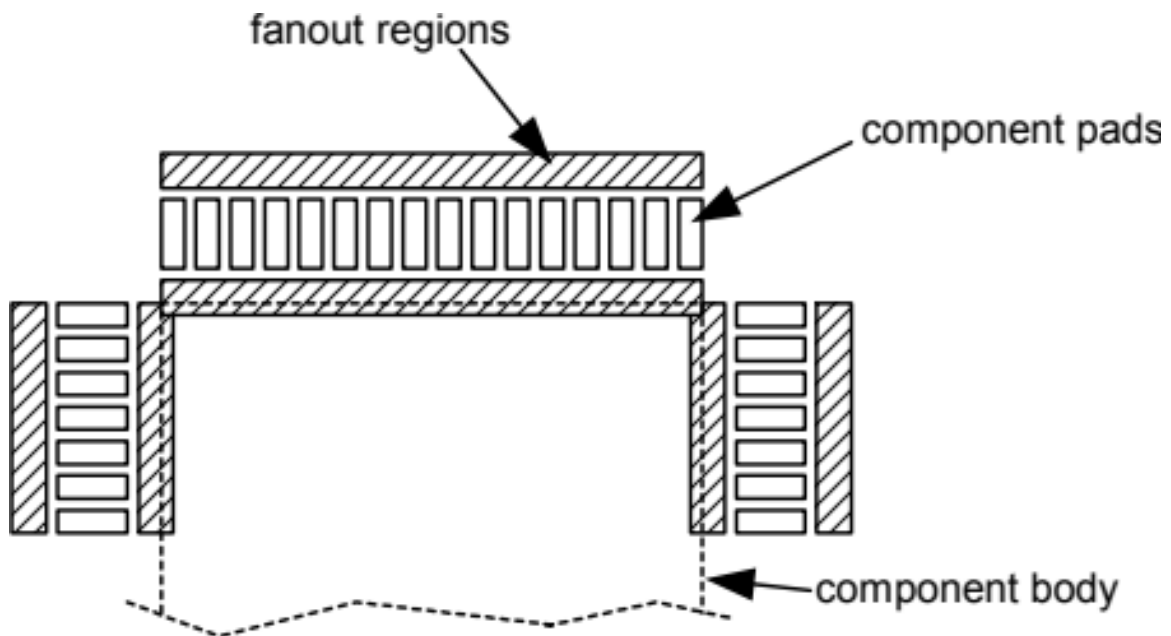


図 C-3: SMD 部品のパッドプリント周囲のファンアウト領域。

デカップリングコンデンサを SMD ランドの下に配置する

一般的なデジタル設計において最も多く使われる受動部品は、デカップリングコンデンサ（通称：パスコン）だ。これらの小さなコンデンサは至る所に存在し、適切に配置されなければ配線とビアファンアウトのスペースを消費してしまう。両面実装基板を使う場合、デカップリングコンデンサをターゲット部品のランドの反対側の面に配置することができる。部品のランドスペースの下にこれらの部品を置けば、ビアファンアウト領域を消費しない。

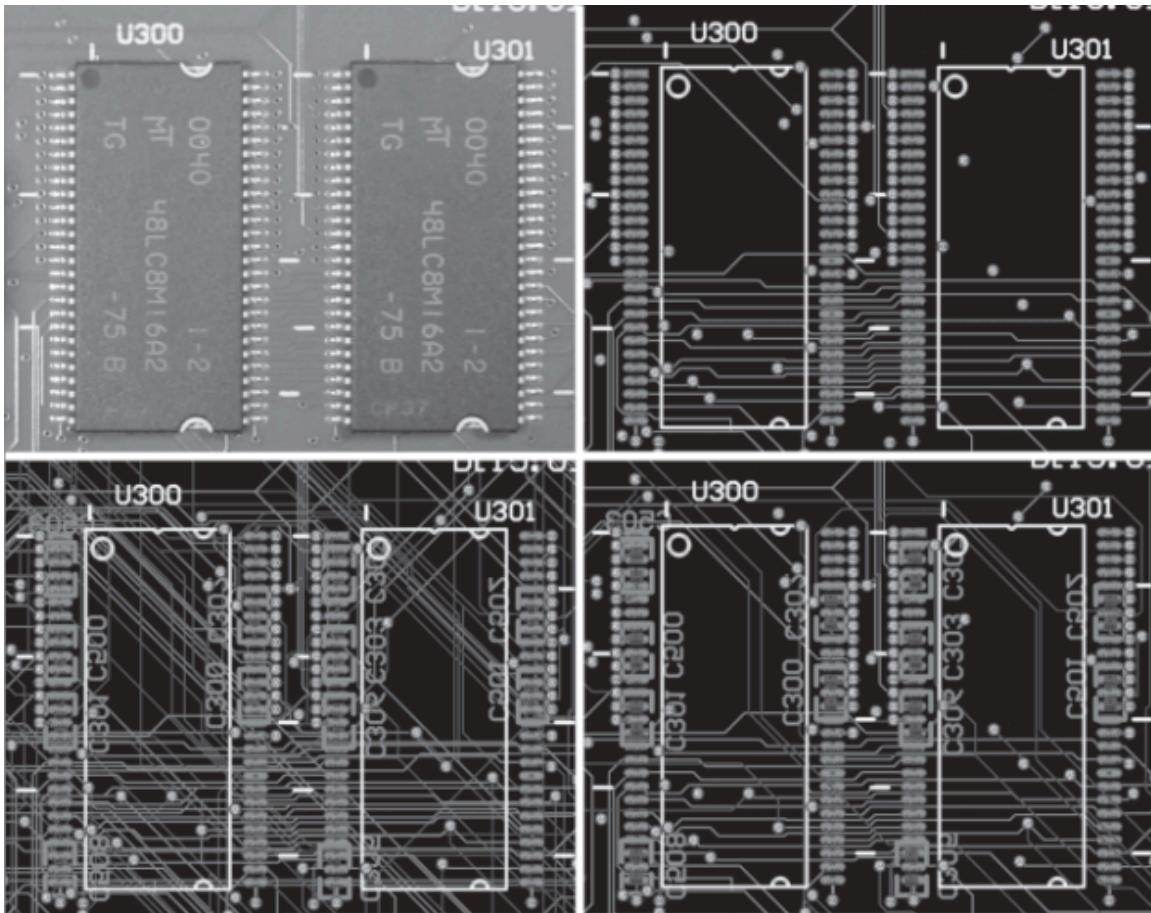


図 C-2: 回路基板レイアウトの4つのビュー。左上から時計回りに：部品を実装した完成基板；PCBレイアウトプログラムのトップレイヤービュー；PCBレイアウトプログラムの全レイヤービュー；両面 SMT レイアウトを示すトップ・ボトムレイヤーのみのビュー。

うまく配置されたデカップリングコンデンサは、部品の電源ピンが使う電源ビアを共用できる。図 C-2 の左下のビューがこの技術をわかりやすく示している（この方法が適切でない特殊なケースもあるので、次のセクションで説明する）。

特殊な配線を把握する

デジタル回路基板のレイアウトの良いところは、アナログ基板と違って、ほとんどの配線があまり深く考えなくても処理できることだ。悪いところは、残りの配線を正しく処理しないと、デバッグが困難な奇妙で厄介な挙動が現れることだ。こうした特殊な配線のルーティングは、ある種の職人芸と言える。このセクションではいくつかのガイドラインを示すにとどめるが、興味のある読者は、これらのテクニックを深く学ぶためにボードレイアウト専門の書籍を探すことをお勧めする。僕が推薦する2冊は、William J. Dally と John W. Poulton による Digital Systems Engineering (Cambridge University Press) と、Howard W. Johnson と Martin Graham による High Speed Digital Design: A Handbook of Black Magic (Prentice-Hall PTR) だ。

回路基板の配線で特別な注意が必要な配線の種類は、一般的に以下のとおりだ：

- 電源配線
- タイミング基準（クロック）配線
- 高速信号配線
- アナログ/混合信号配線

一般的なルールとして、電源配線は平均的な信号配線よりも太くする必要がある。これは特に、細い配線幅（約5ミル）を提供するハイエンドの製造プロセスを使用する場合に重要だ。電源配線は、抵抗成分による発熱と寄生成分のインダクタンスの両方を低減するために太くしなければならない。細い電源配線、特に重要な電源分配点付近の配線は抵抗のように機能して発熱し、供給電圧を低下させ、回路に不定動作を引き起こすことがある。

電源配線の適切なサイズは銅箔の厚さによって異なる。一般的な基板には「1オンス銅」（厚さ1.35ミル——1平方フィートの1.35ミル厚の銅箔が重さ1オンスに相当）が使われる。1オンス銅の外層における12ミル幅の配線は、10度の温度上昇を伴って1アンペアの電流を通過させることができる。同等の電流容量を持たせるためには、埋め込みレイヤーではさらに太い配線が必要だ。

レイヤー間で電源配線を配線する際には、ビアにも抵抗があることを忘れずに。重要な電源配線をレイヤー間で接続するのに、1本のビアだけでは不十分だ。寄生成分の抵抗とインダクタンスを抑えるために、重要な電源配線はレイヤー間を複数のビアで接続すべきだ。また、複数レイヤーにまたがる分散型電源プレーンには、共通の電位を維持するためにビアを広く分散させる必要がある。

[!NOTE] 高性能・低ノイズアプリケーションでは、デカップリングコンデンサと電源ピンの間にビアを置くことが、配線作業の利便性に対して電源インテグリティのコストとして高すぎる場合がある。ビアは高速（数百MHz）の信号の伝搬を乱す。したがって、このようなアプリケーションにおけるデカップリングコンデンサの最適な配置は、部品のピンと電源ビアの間だ。

タイミング基準信号にはクロックとストロブが含まれる。多くのメモリデバイスは、デリケートなタイミング要件を持つ非同期制御ストロブを必要とする。これらの信号は適切にターミネートし、ターミネーション戦略（通常は「デイジーチェーン」ルート）に合わせた方法で配線すべきだ。デイジーチェーンルートには分岐がないため、信号の波面が伝わるパスは一本だけだ。

電気信号は回路基板上で光速の約4分の1、つまりおよそ1ナノ秒で3インチ進む。したがって、高速信号の配線パターンは長さを揃えなければならない。そうしないと、信号がタイミング基準に対して大幅に位相がずれて到着する可能性がある。配線長は、短い配線を最も長い配線の長さに合わせて延長することで揃える。配線長の延長には、蛇行させた配線パターン（サーペントイン配線・ミアンダ配線）を使い、両端点の位置を変えずに有効な長さを増やす。

アナログおよび混合信号のルーティングは、この付録の範囲をはるかに超えている。一般的なホビイストのデジタル設計では、アナログ回路のほとんどは電源部分に限定されるだろう。特定の電源部品に必要な特殊なレイアウト要件は、通常、その部品のデータシートに詳しく記載されている。

電気信号は怠け者で節操がないということを覚えておこう。信号電流は常に最小抵抗のパスを通り、隣接する配線に結合する。さらに、電流は保存されるので、すべての信号電流パスには、明示的かどうかに関わらず、リターン電流パスが存在する。回路基板のアナログ部分をレイアウトする際には、このシンプルなルールを念頭に置いてほしい。

回路基板はヒートシンクにもなる

電源レギュレータや高性能マイクロプロセッサなどの高電力部品をルーティングする際には、回路基板の銅が優れた熱伝導体であることを思い出してほしい。ターゲット部品のヒートスラグやグランドピンに接続された大きな銅の領域をレイアウトするだけで、状況によってはヒートシンクが不要になる。電源プレーンを持つ多層基板を使用している場合は、複数のビアを使って内部レイヤーへ熱を逃がすとよい。

回路基板のヒートシンク特性は、手作業での組み立て時には厄介にもなる。銅の高い熱伝導性により、大きな銅の領域にも接続されている部品のピンを加熱するのが難しくなるからだ。低電力部品を電源プレーンに接続する際は、サーマルリリーフ付きのビアの使用を検討しよう。サーマルリリーフとは、ビアと電源プレーンの接続部分に設ける小さなギャップのセットで、接続の電気的性能に大きな影響を与えずに熱伝導を低減する。なお、密集したサーマルリリーフ付き電源ビアの大きなグループが銅の領域周辺にあると、接続されていない、または接続が不十分な銅のアイランドが生じる場合があることに注意しよう。

各レイヤーの優先ルーティング方向を決める

各レイヤーに主要なルーティング方向を確立することで、密度の高い基板のルーティングが楽になる。たとえば、トップ（表面）レイヤーを水平ルーティングレイヤー、ボトム（裏面）レイヤーを垂直ルーティングレイヤーにする。ボード上で対角に位置する2つの部品間に信号をルーティングする場合は、まずトップで水平に走らせ、次にボトムで垂直に走らせて2つの部品を接続する。代替策として、たとえばトップレイヤーを対角に走る配線を一本通しただけで、ボード両半分のルーティング可能性が半減してしまう。一方の半分からもう一方へのルートがボトムだけになるからだ。このルールの例外は許容される。特に信号インテグリティとルーティング可能性のトレードオフが必要な場合はそうだ。

直交レイヤーで基板をスタックする

各レイヤーの優先ルーティング方向を確立したら、隣接するどの2層も優先ルーティング方向が平行にならないようにレイヤーをスタックする。この直交性により、レイヤー間の信号干渉を最小限に抑えられる。電源レイヤーがある場合は、信号レイヤー間の干渉を遮蔽するために、信号レイヤーの間に電源レイヤーを挟むよう心がけよう。

2 層基板ではフィンガーを使って電源をバス配線する

2 層基板では、基板の外周に電源とグランドをリングとして走らせたいことがある。しかしこれは理想的ではない。リングは基板の中心部を電源から遠ざけ、回路性能を劣化させる大きな寄生電流ループの可能性も高める。代わりに、インターデジタル型（くし形）や積層型の電源フィンガーを使おう。これらのフィンガーは各レイヤーの主要なルーティング方向を確立するものであり、信号をルーティングする前にレイアウトしておくべきだ。

オートルーターの使い方のヒント

オートルーターは良し悪しある存在だ。数時間のルーティング作業を節約できる一方で、数時間にわたる厄介な問題を引き起こすこともある。オートルーターを使う際の第一のルールは、回路基板設計ファイルの唯一のコピーに対して使わないことだ。代わりに、設計のコピーを作成して、そのコピーにオートルーターの魔法をかけよう。

第二のルールは、最終設計に適用する前に、シンプルなテスト設計でオートルーターのバグを把握しておくことだ。オートルーターには、ツールを使い始める前に理解しておかなければならない重大なバグや制限が存在することが多い。オートルーターのバグを確認する際は、ロックされた配線、流し込みポリゴン、変則的な配線サイズの処理方法に特に注意を払おう。オートルーターの中には、ロックされた配線（手動で引いて移動不可に設定した配線）を実際に削除するものもあれば、それらを無視したり、それらが存在すると動作しないものもある。オートルーターを起動する前に重要な電源配線とタイミングネットのレイアウトに数時間かけた後でそれが判明すると、特に悩ましい。

最後に、複雑な基板をオートルーターが完全にルーティングしてくれることを期待しないこと。オートルーターは基板の最初の 90% を素早くルーティングするのに優れているが、基板が混雑するにつれて急激に遅くなる。部品の配置の微妙な変化がオートルーターの成否を左右することも覚えておこう。多くのオートルーターは、特別なアノテーションや理想的な部品配置がなければ、バスやストレートスルー接続を認識しない。

CAD ツール

基板設計ツールの価格は、ここ数年で大幅に下がった。僕が基板設計に最もよく使うツールは Protel 99SE だ（新しいバージョンの Protel DXP はまだ購入していない）。Protel は高度に統合されたツールで、回路図入力、シミュレーション、ライブラリ管理、DRC とオートルーティングを備えた PCB レイアウトがすべて一つのツールに統合されている（ソフトウェアのリリースごとに新機能が設計環境に統合されているようで、良い面もそうでない面もある）。

Protel のソフトウェアの 30 日間完全機能デモは、ウェブサイト www.protel.com からダウンロードできる。フルライセンスは数千ドルだが、同等の機能と深さを提供する他のソフトウェアパッケージと比べれば、まだ競争力がある。その他のハイエンド PCB CAD ベンダーには、Mentor (PADS)、Cadence (OrCAD)、Altium (P-CAD) などがある。興味深いことに、Altium は Protel のソフトウェアスイートも所有している。

始めたばかりで気軽に PCB レイアウトをやってみたいなら、基板製造会社の中には自社の製造サービスを使うクライアント向けに、フル機能の設計ツールを無料で提供しているところもある。ExpressPCB (www.expresspcb.com) は、自社の製造サービスを使うクライアントに無料の回路図入力と PCB レイアウトツールを提供している。ツールは機能的だが、設計ルールチェックの面では多少制限があり、現実的に使える設計の複雑さにも限界がある。しかし、ExpressPCB は初心者向けの優れた入門ツールであり、週末のハードウェアプロジェクトのほぼすべてに対応できる。

完成した設計を製造に出す前に、エクスポートしたファイルをサードパーティのファイルレビューでプレビューして、設計ツールのバグから製造への投資を守ろう。基板製造に最もよく使われるファイル形式は「Gerber」フォーマットだ。僕が信頼している優れた無料 Gerber プレビューは、Graphiccode (<http://www.graphiccode.com/>) が作成している。

基板製造会社

基板製造会社は、CAD ツールと同様に幅広い能力を持っている。大量生産注文のみを受け付ける会社もあれば、クイックターンのプロトタイプとホビイスト市場を主なビジネスとする会社もある。以下は、僕のお気に入りの基板製造会社とその基本サービスの概要だ。

Sierra Proto Express

カリフォルニア州サンノゼに位置する Sierra Proto Express は、最も競争力のあるクイックターンプロトタイプ料金を提供する会社の一つだ。本書執筆時点では、「ノータッチ製品」プロセスを展開している。これらの製造プロセスには厳格な設計ルール要件があるが、非常にリーズナブルだ。たとえば、2 層回路基板を 4 日で 1 枚 34 ドル（最低 2 枚注文）、または 4 層回路基板を 4 日で 1 枚 51 ドル（最低 2 枚注文）で製造できる。これらの価格で提供される技術は、6 ミルの配線/スペース設計ルール、最小仕上がり穴径 15 ミルだ。さらに速いターンタイムのプロセスも提供しており、そちらは配線幅 5 ミル、仕上がり穴径 10 ミルに対応している。詳細は www.sierrapROTOexpress.com を参照。

Data Circuit Systems

同じくカリフォルニア州サンノゼに位置する Data Circuit Systems は、厳格な設計ルールや、安価なクイックターン会社には合わない特殊な処理オプションが必要な設計における、僕の定番ベンダーだ。ウェブサイトからダウンロードできる「プロセス能力調査」は包括的かつ明確に書かれており、設計ルールの解釈に伴う推測作業を大幅に省いてくれる。また、提出された設計に対してかなり厳格な工場検査を行い、後に問題を引き起こす可能性のある微妙なレイアウトエラーを検出することも多い。スタッフは有能で親切だ。価格はほとんどのエンジニアリングプロトタイプメーカーより若干高いが、よく文書化されたプロセスと設計ルールチェックにより、厳格な設計ルールを必要とする設計のリスクを軽減できる。長い目で見れば追加コストも十分に価値があるだろう。www.datacircuits.com を参照。

Advanced Circuits

コロラド州オーロラの Advanced Circuits (www.4pcb.com) は、ウェブサイトインスタント見積もり機能を備えている。この機能だけでも、割引クイックターンプロセスのガイドラインに合わない中程度の複雑さの基板には最適な選択肢になる。インスタント見積もり機能を使えば、価格に合わせた実装技術の選択を最適化できる。さらに、割引クイックターンスペシャルも頻繁に提供している。

Alberta Printed Circuits

カナダ・アルバータ州に位置する Alberta Printed Circuits (AP Circuits) は、迅速なサービスで知られる老舗のクイックターンプロトタイプ PCB 会社の一つだ。彼らが提供する P1 プロセスは基本的なもので、はんだレジスト (soldermask) もシルクスクリーンもない。そのため、組み立て段階でハンダがあちこちに付いてしまい、ファインピッチの表面実装設計の実行が難しい。しかし、P1 プロセスで基板を 1 日で製造・発送してくれる上に、最低注文数量もない。本書執筆時点での基本料金は約 45 ドルで、これに加えて 1 平方インチあたり約 0.65 ドルが加算される。技術仕様は最小ドリルサイズ 20 ミル (最安オプションなら 28 ミル) の 8 ミル配線/スペースだ。AP Circuits は、急ぎで厳格な予算の中で仕上げなければならない基板、特にはんだレジストなしでも組み立てやすいスルーホールや粗い SMT 部品を使う基板に最適だ。 www.apcircuits.com を参照。

スターターポイント：ATX 電源交換アダプタ基板

第 5 章「壊れた電源の交換」では、Xbox の ATX 電源を標準の ATX 電源と交換する方法を説明した。一つの問題は、Xbox と ATX 電源の電源オン信号の極性が逆であることだ。その章で提案したハック的な解決策は、電源を常時オンのままにして、まず電源を入れてから Xbox の電源ボタンを押すという方法で Xbox のオン・オフを切り替えることだった。

電源信号の極性を反転させ、Xbox のフロントパネルだけでスタンバイ電源の状態を制御できるようにする基板を設計・レイアウトすることは、かなり簡単だ。2 本のダイオードを使う代わりに、スタンバイ電源を適切にレギュレートすることもできる。そのような基板は、74HCT04 などのインバータチップと、LM317K などのレギュレータで構成される。LM317K は調整可能なレギュレータで、ATX 電源が提供する +5V スタンバイ電圧を Xbox が要求する +3.3V スタンバイ電圧まで下げるよう設定できる。この基板の回路図例を図 C-4 に示す。

この基板へのコネクタの選択はあなた次第だ。最も簡単な方法は、穴を開けてそのままリード線をはんだ付けすることだろう。接続は 5 本しかない。3 本は電源に接続する：+5VSB (紫) 線、グランド (黒) 線、電源オン出力 (緑) 線だ。残りの 2 本、+3.3VSB (電源コネクタのピン 6) と電源オン入力 (電源コネクタのピン 11) は、Xbox に接続する。

完成した基板を取り付ける前に、レギュレータの電圧出力を必ずテストすること。抵抗値を間違えたりピンを取り違えたりすることは十分あり得る。その場合、Xbox に危険なほど高い電圧が流れ込む可能性がある。また、基板を固定設置する際には、Xbox のケースや Xbox の他の部品との偶発的な接触を防ぐため、基板の上下面を必ず絶縁すること。

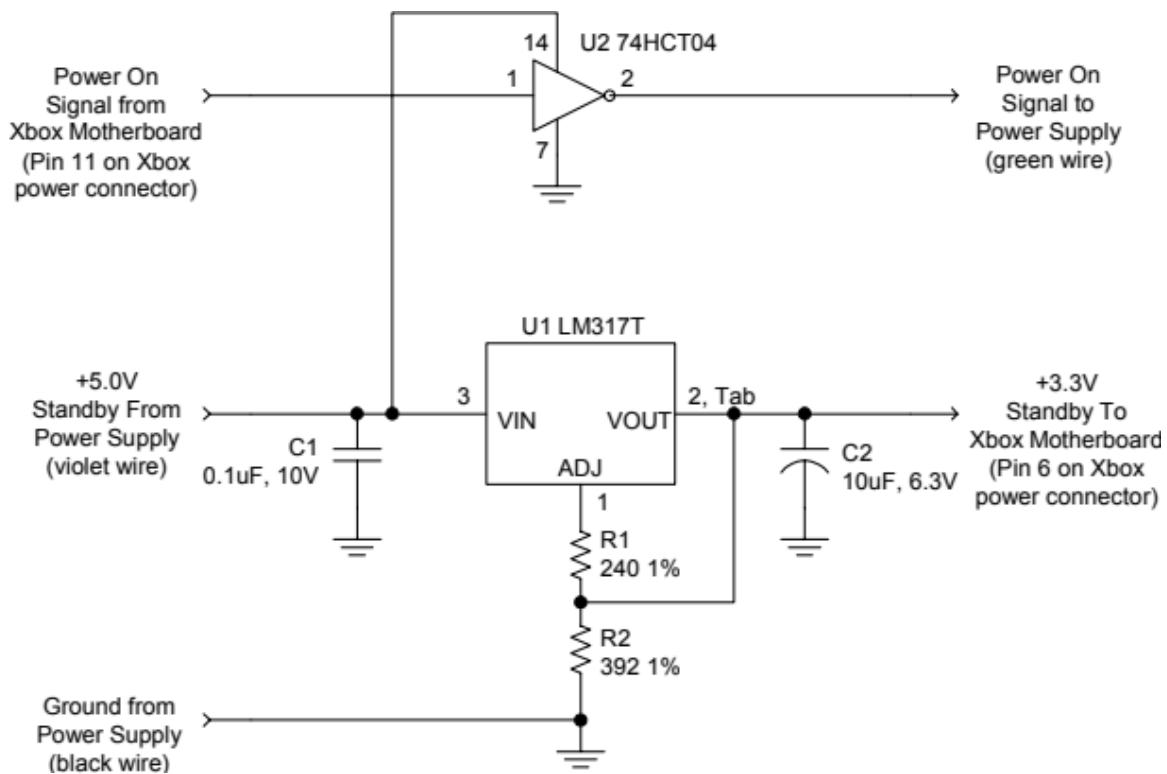


図 C-4: ATX 電源交換アダプタ基板の回路図例。抵抗 R1 と R2 は電圧レギュレータ U1 の出力電圧を+3.3V に設定する。

付録 D FPGA 入門

集積化はハードウェアハッカーの宿敵だ。僕たちはものを分解し、改造し、改善することが好きなのに、時代の流れはすべてを1~2個のASIC (Application Specific Integrated Circuit、特定用途向け集積回路) に詰め込む方向へ向かっている。こうした集積化は凡人の手の届かない領域にある。チップ上の回路パターンを定義するためのマスクセット1組が、あっという間に100万ドルに近づいているからだ。つまり、チップを1種類リビジョンするたびに100万ドルがかかる。間違いを犯して新しいマスクセットが必要になれば、また100万ドルを出さなければならない。

幸いなことに、チップ1種類に100万ドルをキャッシュで前払いするのは、多くの企業にとっても無理な話だ。そのため、FPGA (Field Programmable Gate Array、フィールドプログラマブルゲートアレイ) という市場が生まれた。FPGA とは汎用の「再構成可能な」プログラマブルハードウェアデバイスであり、多くのアプリケーションでASICの代替として使える。

FPGA とは何か

FPGA とは「フィールドプログラマブルゲートアレイ」の略だ。つまり、エンドユーザーが現場でプログラムできるゲートの配列だ。FPGA は「自宅で作れるカスタムシリコン」と考えることもできる。ただし、近年の部分的再構成や文脈依存の再構成への対応が、ASIC にはない次元を FPGA に加えている。ASIC は量産時の単価では安く、クロック速度も高くできるが、FPGA は少量～中程度の量産用途やプロトタイプ開発において定番のツールとして確立している。

FPGA の基本アーキテクチャは、柔軟な配線ネットワークに埋め込まれたハードウェアプリミティブの配列だ。FPGA の強みは、複雑な計算をより単純な論理関数の連鎖に分解できるという点にある。

sequence of simpler logic functions. These simpler functions can each be broken down in turn, until the entire computation is described by nothing more than a sequence of basic logic operations that can be mapped into the FPGA's hardware primitives. Thus, the same FPGA can be used to implement a microprocessor, a video controller, or a tic-tac-toe game just by changing the configuration of the hardware primitives and the routing network.

The kinds of hardware primitives implemented by an FPGA architecture strongly influence the FPGA's implementation efficiency for a given target application. Modern FPGAs provide designers with mostly one bit wide primitives: a 4 or 5 input to 1-bit output lookup table, and a single bit of time-synchronized storage known as a flip flop. Lookup tables are used as the logic primitive because they can be programmed to perform any logic operation with as many terms as there are inputs to the lookup table. These primitives are then wired into a vast programmable network of wires; a typical high-end FPGA might have many tens of thousands of these primitive elements.

It turns out that while single bit-wide structures are very general, they can be very resource-inefficient in applications where the natural data width is large. In particular, the area dedicated to the actual logic primitives is around 1 percent in many cases, with the remainder being configuration memory

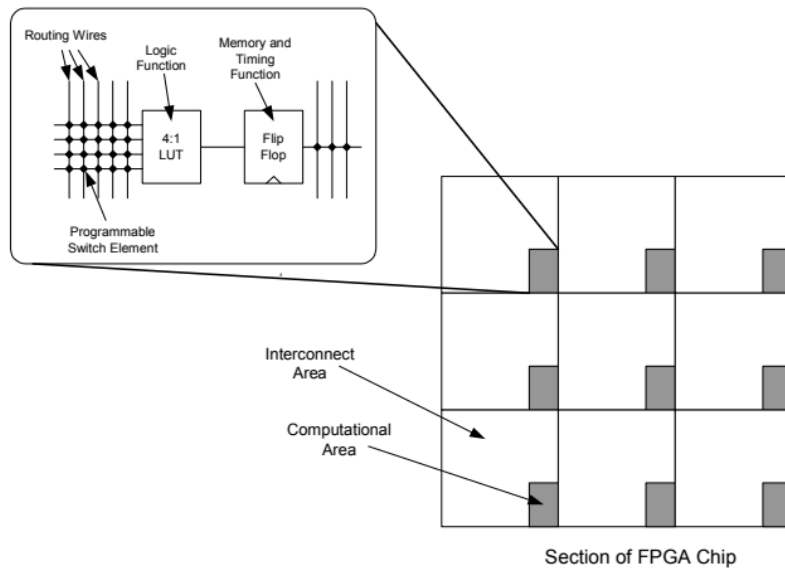


Figure D-1: Block diagram of a typical FPGA structure, illustrating the disparity between the amount of wire on an FPGA versus the amount of computational logic. A typical modern FPGA will contain several tens of thousands of these basic cells.

図 D-1: 典型的な FPGA 構造のブロック図。FPGA が持つ配線量と演算ロジック量の差異を示している。現代の高性能 FPGA には、このような基本セルが数万個含まれている。

これらの単純な関数はさらに分解でき、最終的に計算全体が FPGA のハードウェアプリミティブにマッピングできる基本論理演算の連鎖で記述される。同じ FPGA でも、ハードウェアプリミティブと配線ネットワークの構成を変えるだけで、マイクロプロセッサにもビデオコントローラにも○×ゲームにもなる。

FPGA アーキテクチャが実装するハードウェアプリミティブの種類は、特定のターゲットアプリケーションに対する FPGA の実装効率に大きく影響する。現代の FPGA が設計者に提供するプリミティブは、主に1ビット幅のものだ。具体的には「4入力または5入力から1ビット出力へのルックアップテーブル」と「フリップフロップ」と呼ばれる1ビットのタイミング同期ストレージ素子だ。ルックアップテーブルを論理プリミティブとして使うのは、入力数と同じ数の項を持つどんな論理演算でもプログラムできるからだ。これらのプリミティブは巨大なプログラマブル配線ネットワークに接続される。高性能な FPGA だと、こうした基本要素が数万個含まれる。

単ビット幅の構造は非常に汎用的だが、データ幅が大きいアプリケーションでは面積効率が悪くなる。実際、多くの FPGA では面積の約1パーセントしか実際の論理プリミティブに使われておらず、残りはコンフィギュレーションメモリと配線に使われている。この膨大な配線は、1ビット幅の用途で必要となる多数のルーティングの組み合わせを処理するために必要だ。

面積効率を高めるため、多くの FPGA は RAM の塊や乗算器ブロックなどの粗粒度プリミティブもいくつか搭載している。Xilinx の Virtex II-Pro FPGA はチップ上に複数の PowerPC コアまで搭載している。これは印象的に聞こえるが、そのコアが実際に消費する面積は驚くほど小さい。PowerPC プロセッサが消費するシリコン面積はおそらく 1mm^2 強程度で、FPGA 全体の面積（数百 mm^2 ）と比べると微々たるものだ。

最近の FPGA は非常に柔軟な演算ハードウェアに加え、非常に柔軟な I/O も備えている。典型的な FPGA は PCI、AGP、LVDS、HSTL、SSTL、GTL を含む最新の高速シグナリング規格のほぼすべてに対応できる。さらに DDR クロック信号も扱える FPGA がほとんどだ。これらの略語が何を意味するかわからなくても、基本的な考え方はシンプルだ。FPGA は Xbox のような典型的な PC マザーボードにあるあらゆるハードウェアと通信するために使える。これはハードウェアハッカーにとって最高のニュースだ。FPGA は PC 内のほぼあらゆるチップのエミュレーションやモニタリングに使えるからだ（もちろん、FPGA が PC の速度に追いつけない場合は PC をダウンスクロックする必要があるが）。

FPGA の設計

典型的な FPGA 設計フローには、複数の設計エントリ手法がある。グラフィカルに考えるのが好みなら、多くの設計フローはスキマティックキャプチャツールをサポートしている。スキマティックキャプチャはハードウェア設計にとって直感的なことが多いが、保守や修正が難しくなりがちだ。たとえば、ネット名のすべての出現箇所を変更しようとする、すべての配線をクリックして新しい名前を入力しなければならず面倒だ。さらに、設

計階層の1レベルの規模はスキマティックシートのサイズに制限されるため、複雑な設計ではスキマティックキャプチャだけでも相当な計画と先見の明が必要になる。

そのため、複雑な設計を実装する場合はHDL (Hardware Description Language、ハードウェア記述言語) が主流のツールだ。HDL は一見すると普通のプログラミング言語によく似ている。たとえば Verilog の構文は C や Java に非常によく似ている。しかし、言語のセマンティクス (意味論) の理解は少し難しい。

ハードウェアには C のような手続き型言語では表現できない固有の並列性がある。FPGA のすべてのゲートとすべてのフリップフロップは並列に計算できるのに対し、C プログラムでは基本的に単スレッドの実行が想定されている。そのため、HDL ではハードウェアを並列動作するプロセスの集合として表現する。コンパイラがプロセスをゲートに変換する方法を理解できるよう、すべての関数を正しいプロセスにグループ化するのはコーダーの仕事だ。

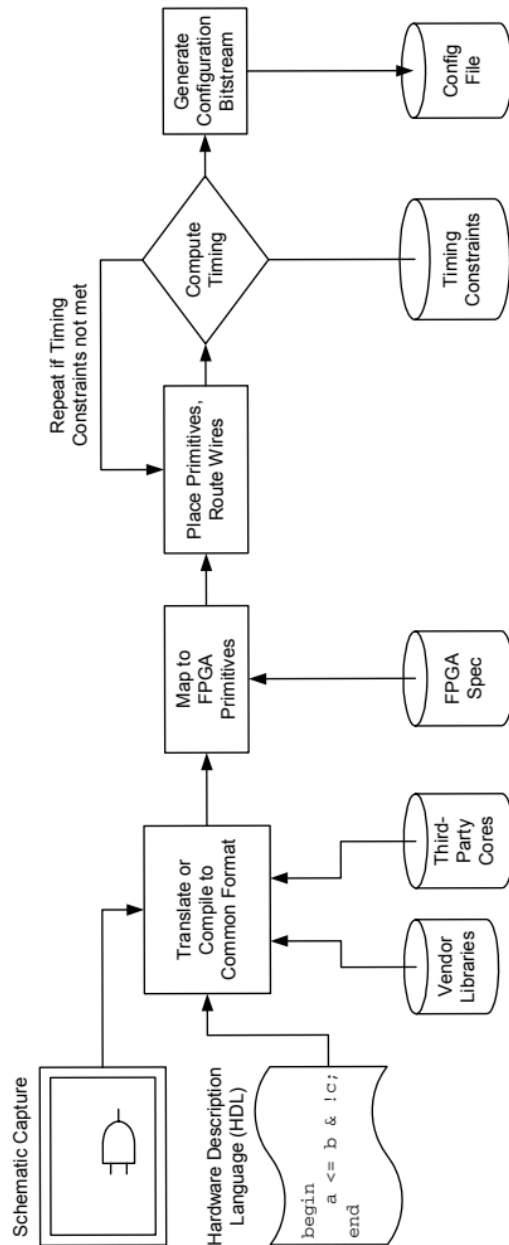


Figure D-2: Typical FPGA design flow.

図 D-2: 典型的な FPGA 設計フロー。HDL 記述やスキマティックキャプチャから始まり、コンパイル、マッピング、配置配線を経て最終的なコンフィギュレーションビットストリームが生成される。

たとえば、Verilog における単一のクロック付き記憶素子（フリップフロップ）は、通常次のような構造を持つ「プロセス」だ。

```
input inData; // 入力と出力を宣言する
input clock;
reg bitOfStorage; // ストレージビットを reg 型として宣言する

always @(posedge clock) begin
    bitOfStorage <= inData;
end
```

このコードは入力ポート inData の値を受け取り、クロックの立ち上がりエッジごとに inData を bitOfStorage という出力のフリップフロップに格納する。always @(...) begin ... end という構文で区切られた複数のプロセスを 1 つの設計に含めることができ、すべてのプロセスは並列に実行される。

組み合わせ論理もプロセスとして表現できる。たとえば、次の Verilog コードはクロックのない 2 入力マルチプレクサを実装している。

```
input a;
input b;
input select;
output out;
reg c;

always @(a or b or select) begin
    if( select == 1'b1 ) begin
        c <= a;
    end else begin
        c <= b;
    end
end

assign out = c; // assign文にも論理関数を含めることができる
```

この例では、always キーワードに続く括弧内のブロックが「センシティビティリスト」であり、出力に影響する可能性のあるすべての入力が含まれている。センシティビティリストからパラメータを省略すると、そのパラメータが変化しても出力は変化しない。たとえば a と b をセンシティビティリストから省略すると、select が変化したときにのみ出力が変化する。つまり、select の状態に応じて a か b かを格納するラッチを作ったことになる。しかし、マルチプレクサの本来の動作は select が変化しないときでも a または b の変化を常に出力に伝えることなので、a と b の両方がセンシティビティリストに含まれていなければならない。

📦 コラム：FPGA のオーバークロック

FPGA 向けのタイミングモデルはかなり保守的に設定されていることを知っておく価値がある。これは、タイミングアナライザが認める値よりはるかに高い周波数で FPGA が正常動作する可能性が高いことを意味する。実際、FPGA のロジックを丁寧に手動配置すると、スペックに明記された性能よりずっと高い性能を引き出すことができる。

たとえば、Xbox の HyperTransport バスタップの実装に使った FPGA (Xilinx Virtex-E) は、約 200 Mbits/s/pin のデータレートしか仕様に規定されていないが、このアプリケーションでは 400 Mbits/s/pin が必要だった。それを実現できた理由は、実際のロジックと記憶素子は非常に高速に動作できるのに、性能のほとんどがロジック素子間の信号を伝える配線とリピータで消費されているからだ。具体的には、400 Mbits/s では遅延が大きすぎてデータを 1 クロックサイクル分実質的に記憶してしまう配線が出てくる。

低速な配線がどれかを見つけるために、データのシーケンスをキャプチャしてオシロスコープで事前に発見したパターンと比較した。低速パスを特定したら、遅延が少なすぎるチャンネルのクロックを反転させたりフリップフロップを挿入したりした。最終的には、時間スキューが補正された信号のセットができあがった。それらの信号は、従来のコンパイル済み HDL 設計技術が使えるより低いクロックレートで簡単に分割できた。

この手法は非常に強力だが、汎用的ではない。配線による遅延はチップごとに異なり、周囲温度や電源電圧の品質といったパラメータに依存するからだ。ただし、制御された環境下で特定の 1 チップに限定すれば、定格の 2 倍の性能を引き出せた。このアプリケーションと一般的なアプリケーションとのもう一つの重要な違いは、数千回に 1 回程度のビットエラーが許容できたことだ。3 回トレースして XOR することでランダムノイズで失われた情報を回復できたからだ。しかし、通常のアプリケーションでは 1 万回に 1 回のビットエラーレートは許容できない。10 兆回に 1 回以下の回復不能エラーレートのほうがより標準的だ。これは僕がよく言う格言に通じる話だ。「何かを 1 回やるのは簡単だが、それを 100 万回完璧にやることは難しい。」

HDL の習得には本書の範囲を超えた多くの微妙な点があるが、上記の 2 つのコードセグメントがどんなものを期待すべきかの雰囲気を伝えていると思う。熟練したソフトウェアプログラマは初心者より HDL への適応に苦労することがある。ソフトウェアの世界で当たり前のように使うトリックの多くが、直接的なハードウェア実装にはうまく変換されないからだ。配列、構造体、乗算、除算プリミティブはソフトウェアの世界では当然のように使えるが、それぞれが潜在的に大きく非効率なハードウェアブロックに変換される。さらに、ハードウェア実装では、case 文のすべての可能なケースが意図するかどうかにかかわらず存在する。case 文をデフォルトケースで完全に指定しないと、暗黙のケースを処理するための余分なハードウェアが合成されることが多い。Verilog のチュートリアルやシンタックスリファレンスは Google で大量に見つかる。verilog syntax や verilog tutorial は構文リファレンスやチュートリアルを探すときの良い検索キーワードだ。Xilinx のウェブサイトにも FPGA 設計者向けの優れた Verilog リファレンスがあるし、Sutherland HDL 社は http://www.sutherland-hdl.com/on-line_ref_guide/vlog_ref_body.html で無料の Verilog クイックリファレンスガイドを公開している。

HDL 設計エントリのもう一つの利点は、無料・有料の「ソフトコア」が利用できることだ。www.opencores.org のようなウェブサイトでは、USB インタフェース、DES・AES

暗号エンジン、各種マイクロプロセッサなどの機能向けに GPL ライセンスの HDL コアを提供している。さらに、ほぼすべての標準機能についてサードパーティベンダーが有償コアを販売している。

設計エントリが完了したら、ハードウェアにコンパイルする前に設計をシミュレーションすることを強くお勧めする。コードを少し変更してハードウェアに書き込んでプローブするというやり方でバグを追跡しようとするのは非常に非効率だ。シミュレーションならボタン一つで回路のあらゆるノードをプローブできる。また、コード変更をシミュレーションするのに必要な労力は、変更をハードウェアに書き込むまでの手間と比べると圧倒的に小さい。

設計のエントリとシミュレーションが済んだら、共通のネットリスト形式にコンパイルまたは変換する必要がある。このネットリスト形式はターゲット FPGA のハードウェアプリミティブにネットリストプリミティブをマッピングするプログラムに入力される。マッピングされたプリミティブは次に配置配線される。得られた設計は、設計者が指定した制約セットへの適合をタイミング解析で確認する。設計が仕様を満たさない場合は、配置配線を繰り返して反復的に改善される。設計が制約を満たすと、コンフィギュレーションビットストリームジェネレータに渡され、FPGA の内部表現が自己コンフィギュレーションに使える 2 進ファイルに変換される（最新の FPGA 設計ツールでは、これらのステップすべてがボタン一つでほぼシームレスに実行される）。

プロジェクトのアイデア

FPGA とは何か、どうプログラムするかが少しわかったところで、実際に何ができるだろうか。

FPGA は今日、非常に印象的な範囲のタスクをこなせるだけの論理容量と性能を持っている。FPGA の明らかな産業的応用は、ハードワイヤードシリコン向けの設計エミュレーションだ。カスタムチップの製造コストは急騰しており、致命的なミス犯了した場合の修正コストが数十万ドル、あるいは数百万ドルに達することもある。

一方、FPGA の HDL 記述でミスを犯した場合の修正コストは基本的に時間と設計工数だけで済む。部品を捨てる必要もなく、新しい部品を買い直す必要もない。そのため、多くの企業が最終シリコンの製造（テープアウト）前に FPGA でモックアップ設計を完全シミュレーションするという戦略を採用している。このアプローチの副次的な利点は、カスタムシリコンが製造中（数ヶ月かかることもある）の間に、シリコンのソフトウェア・ハードウェアチームが FPGA モックアップを使って設計の検証を始められることだ。

ハッカーにとって FPGA はあらゆる複雑なプロジェクトの万能薬のようなものだ。総当たりキー探索や大量データの高速暗号化に関心があるなら、暗号関数の実装に最適な選択肢だ。無料の乗算器とデジタルフィルタのコアが存在することもある、信号処理関数の実装にも非常に有用だ。FPGA は DSP より低消費電力で高性能を達成でき、バッテリー駆動のロボティクスなどのアプリケーションで独自のニッチを持っている。FPGA は組み込みコントローラ用途にも役立つ。今日の FPGA には PIC 相当以上の小型マイクロプロ

セッサコアが簡単に収まる。シリアルポートやPWM タイミングジェネレータなどのカスタムハードウェアパブリックをすべて追加すれば、準備完了だ。

FPGA は数値演算に重点を置かない場面でも役立つ。FPGA はタイトなスペースでの論理設計エラーを修正する「グルーロジック」として優れており、うまく配置された FPGA があれば、論理設計エラーのパッチとして基板にジャンパ線を追加する必要をなくせる。また、1 万ドルの Tek TLA メインフレームを買えない僕たちにとって、FPGA は安価なロジックアナライザの代替にもなる。最新 FPGA の高速 I/O 能力と自動生成された FIFO 設定の組み込みメモリを組み合わせれば、信号キャプチャ・解析システムの設計は大幅に楽になる。

最後に、FPGA は一見すると明らかでないアナログ・デジタル混在 (ミックスドシグナル) の場面にも応用できる。最も一般的なミックスドシグナル用途はおそらく VGA モニタのアナログ信号を FPGA で駆動することだろう。抵抗分圧器を 2 つ用意するかよく選んだ出力ドライバタイプを使うだけで、カラー画像を生成するのに必要なタイミングとロジックはすべて FPGA 内部のロジックで処理できる。FPGA は PWM 方式の D/A コンバータとしても、シグマデルタ D/A コンバータや A/D コンバータの一部としても、ほとんど手間をかけずに使うことができる。

購入先

これほど汎用性が高くで強力なツールは高価に違いないと思っているかもしれない。10 年ほど前はそうだったが、今日では 10 万ゲートの FPGA を 50 ドル以下で購入でき、設計ツールは教育ユーザーや趣味のユーザーに対して無料で提供されることが多い。

もちろん、FPGA だけでは使えない。使うためには適切な接続が施されたボードに実装する必要がある。そのために XESS (www.xess.com) という会社が比較的手頃な FPGA スターターキットのラインナップを提供している。製品ラインナップは新しい FPGA が登場するたびに変わるが、執筆時点でのエントリーレベルの FPGA ボードは XSA-50 で、5 万ゲートの FPGA が搭載されて約 150 ドルだ。このボードには RAM 数メガバイト、パラレルポート、VGA ポート、PS/2 キーボードポート、その他いくつかの必須アイテムも含まれている。

もう一つの選択肢は、大胆に自分でゼロからボードを作ることだ。本書の他の付録でボードレイアウトと製造への入門方法、ファインピッチの FPGA デバイスを基板に取り付ける方法を説明している。自分でボードを作ってみるのは実際非常にやりがいがある。挑戦してみることをお勧めする。今日ではボードの製造コストは 100 ドル以下なので、ボードが動かなくても大きな損失にはならない。

自作ボードを作る場合は、Xilinx のディストリビュータから FPGA を購入する必要がある。Xilinx のウェブページ (www.xilinx.com) に最新のディストリビュータへのリンクがある。執筆時点で比較的便利なディストリビュータの一つは NuHorizons (www.nuhorizons.com) で、登録や特別な顧客アカウントなしに製品の在庫状況と価格情報をウェブページで確認できる。

FPGA 開発ソフトウェアは通常、低価格または無料で入手できる。たとえば Xilinx は Virtex-II (300K ゲートまで)、Spartan II-E、CoolRunner ラインの部品向けに無料の開発環境を提供している。この開発環境は Xilinx ISE WebPACK と呼ばれ、www.xilinx.com での登録後にダウンロードできる。この無料環境はスキマティック・HDL 入力、HDL 合成、フロアプランナ、タイミング駆動の配置配線、タイミング解析、消費電力解析ツールを含む印象的な機能リストを誇っている。

Xilinx はまた、Prentice-Hall 経由で「Xilinx Student Edition」と呼ばれるバージョンも提供している。このソフトウェアには FPGA 設計を学ぶのに役立つ多数のチュートリアルとドキュメントが同梱されている。Xilinx のウェブサイトでは Education タブの下に幅広いチュートリアルや講義が用意されている。

付録 E デバッグ：ヒントとコツ

たまにしか再現しない不安定動作や間欠不良を、絶対に見過ごしてはいけない。たとえば、基板のある箇所を触ると動いたり止まったりする、手をかざすだけで挙動が変わる、電源投入直後の一瞬だけ違う動作をする、といった現象がある。こうした観察結果を「たまたま起きた偶然」「些細なこと」として切り捨てたくなる気持ちはわかる。しかし、それが実際に起きたという事実には必ず説明があるはずだ。

具体例を挙げよう。基板を触ったら動作状態が変わった——どこを触ったか？どんな触り方をしたか？手に汗をかいていたか、乾いていたか？人体は、小さなキャパシタンスと大きな抵抗として機能する。そのため、信号をわずかに遅らせたり、接続されていないデジタル入力のような高インピーダンスのノードを放電させたりする。指で強く押した場合は基板をたわませることになり、クラックが入った配線や不良はんだ接合の電気的特性を変えてしまうこともある。

症状を原因と誤解してしまうケースは少なくない。焼き切れた配線や破損した部品は、通常は問題の「原因」ではなく「症状」だ。つまり、回路のどこかで起きた誤動作が部品の破損を引き起こしているのが普通であって、部品が突然単独で故障するケースは比較的まれだ。壊れたステレオをデバッグしているとしよう。焼けた臭いがして、黒焦げになった大きな抵抗器が見つかった。その抵抗器だけ交換しても、交換品がまた燃えるだけという可能性が高い。本当の原因はショートしたトランジスタや電源回路の損傷かもしれないが、焼けた抵抗器ほど目立たない。

もうひとつ強力な観察手法として、正常動作品との比較がある。壊れたデバイスをデバッグするなら、動作している同型品を見つけて、電圧やその他の動作特性を比較してみよう。自作回路のデバッグであれば、可能であれば回路のシミュレーションを作るか、似た設計の回路を探す。こうした「正常な基準」を使えば、異常な動作を素早く特定できる。さらに、コントロールされた方法で正常品に意図的に故障を誘発し、本当に根本原因を見つけられたかを確認することもできる。この手法は特にシミュレーション環境で有効だ。

よくあるバグ

自作プロジェクトにおけるハードウェアバグで最も多いのは、不良はんだ接合と、キャパシタ・ダイオード・IC・コネクタのような極性のある部品の実装ミスだ。コネクタは物理的な負荷が最もかかり、目視だけでは良否を判断しにくいいため、特に故障の原因になりやすい。以下は、発生頻度が高いおおよその順に並べたよくあるバグのリストだ。

1. **不良はんだ接合。** コールドジョイント、はんだブリッジ、接合忘れがこれに含まれる。丁寧な目視検査で多くの不良接合を発見できる。すべての接合部のはんだは滑らかで光沢があり、基板のランドや部品リードの上に「濡れたメニスカス」のような形で盛り上がっているはずだ。良い接合と悪い接合の写真は付録B「はんだ付けのテクニク」に掲載している。表面実装 (SMD) パッケージの不良接合は、ピンセットの先やゼムクリップのような硬い細い棒を、パッケージの長さ方向に沿ってピンの上をそっとなぞることで素早く発見できる。接触不良のピンは少し曲がる。基板をたわませてでも不良接合を発見しやすい。それでも見つからない場合は、オームメータで接合の品質を確認する必要がある (はんだ付け後に基板が汚れてしまった場合は、綿棒にイソプロピルアルコールなどの穏やかな溶剤を付けて拭き取ってから検査しよう)。最後に、「見るのが信じることを」忘れずに——拡大鏡を使って検査しよう。中程度の倍率のマイクロスコープが理想だが、ドラフター用ランプに付いているような固定式拡大鏡でも、宝石師が使うようなリングルーペでも大きく役立つ。
2. **不適切な部品の値。** 見た目が似ていて値が違う部品を誤って基板に実装してしまうと発生する。表面実装の受動部品はラベルがないか読みにくいことが多く、特に問題になりやすい。部品の値を正しくテストするには、基板から外してからでないと測れないことを覚えておこう。組み立て時に部品をはっきりラベル付けされた袋や箱に整理整頓しておけば、誤った部品の実装を防ぎやすい。
3. **不良コネクタ。** 逆向きに取り付けられたコネクタ、さらに悪い場合はピンの割り当て自体が間違っていて設計されたコネクタが含まれる。ピン1の位置と、コネクタが使う番号付け方式に注意しよう。ジグザグ方式を使うコネクタもあれば、円形方式を使うものもある。ワイヤー to ボードコネクタは手作業での製作が難しい。ワイヤーとコネクタ接点の接合部をすべて確認し、かしめ不良・絶縁被覆の残留・不良はんだ接合がないかチェックしよう。最終手段として、ボルトメータでコネクタの導通を確認する。
4. **データシートを読まないことによる設定ミス。** 複雑なチップは、一連のピンをハイまたはローのロジックレベルに固定する「ストラップ」によって複数の動作モードを選択できることが多い。また、正常動作のためにピンの負荷やバイアスを設定する外部抵抗が必要なチップも多い。内部機能を安定させるためにキャパシタ・抵抗・インダクタのネットワークを要求するチップもある。使わない入力ピンも、正常動作のために固定電圧へのターミネーションが必要なことが多い。特定の機能を使わないからといって、データシートのその部分を読み飛ばしてはいけない。
5. **設計上の問題または実装上の問題。** バグがそもそもの設計エラーから生じることもある。正しい回路図から基板レイアウトへの変換時に問題が起きることもある。

変換ミスは、回路図のネット名を指定するときのタイプミスや、回路図シンボル上の暗黙の電源名によって生じることが多い。暗黙の電源名はデジタル部品では便宜上よく使われるが、複数の電源電圧を使う設計では大きな問題になりうる。このような問題は、付録Cで説明したヒューリスティックなネットリストチェックプログラムを使えば、レイアウト前に検出できる。高速設計ルールの違反も実装上の問題のひとつだ。25MHz以上で動作する回路や、エッジレートが5ns未満の高速回路では、電氣的インピーダンスと伝送線路のターミネーションに特別な注意が必要だ。

6. **電源が仕様外。** 電源電圧は、できるだけ実際に使用する箇所に近い場所で測定しよう。配線によって実際に供給される電圧が低下することがある。回路に問題はなく、単に電源が設計を動かすのに十分な供給能力を持っていない場合もある。また、電源電圧の時間的な変動も確認しよう。電源への過剰なノイズが問題を引き起こすことがある。大量の高速CMOSロジックを使うシステムは電流消費の急激な変動があり、電源電圧に短いディップやスパイクを引き起こす可能性がある。
7. **PCBの配線の断裂・損傷。** 手作業で基板を組み立て、部品の取り付けに苦労した場合に起きやすい。組み立て時の過熱によって配線が基板から剥がれることがある。使用する基板メーカーにも注意しよう。一部の基板メーカー（特に短納期の格安プロトタイプ専門業者）は、基板のネットリスト電気テストを完全には実施しない。エッチングが過剰で公差外に細くなった配線を探し、すべてのビアホールの周囲に銀色のアニュラスリングがあることを確認しよう。基板穿孔時にドリルビットがずれたり斜めに入ったりすることがあり、そのような穿孔ミスが電氣的接続を断ってしまうことがある。
8. **ラッチアップまたは電源シーケンスの問題。** ラッチアップは、チップの基板（サブストレート）内部に電源とグランドの間の寄生的なショートが生じるという、潜在的に深刻な現象だ。電流がサブストレートに流れ込むことでラッチアップが引き起こされる。これは、チップの電源電圧より高い入力電圧が印加されるような混在電圧システムで起きうる。多くの場合、ラッチアップはチップの過熱を伴い、恒久的な損傷につながる。はじめてシステムに電源を投入するときに推奨される方法は、アンメータでシステムが消費している電流を監視しながら、すべての部品に触れて過度に熱くなっているものがないか確認することだ。ラッチアップが起きた場合、通常は数百ミリアンペアオーダーの過電流消費が観察される。
9. **熱の問題。** 主に線形電圧レギュレータや高電力デジタル回路で問題になる。すべての高電力部品に適切なヒートシンクが付いていること、そしてヒートシンクが電氣的に活性なチップパッケージの部分に接触する場合は適切に絶縁されていることを確認しよう。
10. **露出した銅へのショート。** コネクタやチップで、部品の底面に金属が露出しており、ビアのような基板の露出した銅部分にショートしてしまうケースがある。また、ネジで基板を固定している箇所でも問題になる。金属ネジの頭が、ネジ穴に近すぎる位置に配置されたビアと接触してしまうことがある。

11. **基板の汚染。**はんだフラックスの残留物やその他のプロセス残留物が基板に残り、微小な漏れ電流経路を形成することで引き起こされる。フラックス残留物の中には無視できない抵抗値(1MΩ未満)を持つものがあり、高インピーダンス回路——たとえば時定数の大きな RC 回路——では問題になることがある。
12. **不良の試験機器。**中古や古い試験機器を使う場合に特に問題になる。テストプローブは時間とともにキンクが入ったり、校正がずれたりするため、オシロスコープで見えるひどい信号が、実は不良プローブや接地の取り方が悪いせいということがある。試験機器を正常な信号に対してキャリブレーションし、試験機器側の問題を除外しよう。
13. **最も可能性が低いのは、チップ不良や部品の初期不良だ。**部品メーカーは、出荷する部品が機能していることを保証するために大変な努力をしている。シンプルから中程度の複雑さの部品では、典型的な故障率は百万分の一桁台で測定される。問題の原因がメーカーの不良チップにあるかと思いたがる気持ちはよくわかるが、それはほとんどの場合あてはまらない。不良品が見つかった場合、通常はその部品が製造プロセスの問題(粗雑な取り扱いや組み立てのトラブル)か、回路の別箇所の設計上の問題によって観察された故障を誘発されたことが多い。

剥がれた配線やランドからの復旧

PCB の銅箔配線が剥がれたり切れたりすることは、フライングワイヤーを使って改造を行おうとする人がよく遭遇する問題だ。この銅箔配線の剥離は、通常はんだごての過熱によって引き起こされる。もうひとつよくある原因は、配線を基板にはんだ付けした後で改造ワイヤーを引っ張ること——ワイヤーの端の絶縁被覆を剥くときにやりがちなことだ。幸い、この問題からの復旧は通常比較的簡単だ。

[!TIP] 最善の対策は予防だ。基板作業に過剰なワット数のはんだごてを使わないこと。温度制御式のごてが理想だが、低ワット(15W)の安価なごても十分使える。はんだが基板にうまく乗らないと感じたら、熱を加えるのをやめること。そのかわり、基板とワイヤーに少量のフラックスを塗り、チップコンディショナーまたは蒸留水で湿らせたスポンジではんだごての先端を清潔にしよう(水道水にははんだごての先端を劣化させる化学物質が含まれている)。こうすることで、はんだ付け性が向上し、接続するのに必要な熱や力を減らせる。

配線やランドが基板から浮き始めたときにまずすべきことは「止まる！」ことだ。問題を悪化させてはいけない——ワイヤーを引っ張り続けて配線全体を剥いでしまうのが最悪の事態だ。ワイヤーがまだつながっている場合は、はんだごてを接合部にほんの少し触れさせてワイヤーが落ちるようにして外す。図 E-1 はそのような惨事の現場を示している。

the insulation off the end of a wire, after it has been soldered to the circuit board. Fortunately, it is usually fairly easy to recover from this problem.

Tip



The best solution is prevention. Do not use an over-powered soldering iron for working on circuit boards. A temperature-controlled iron is preferred, but an inexpensive low-wattage (15 watts) iron will also work. Also, if the solder does not seem to be sticking to the board, stop applying heat. Instead, put a touch of flux on the board and the wire, and clean the soldering iron tip with tip conditioner or a sponge dampened with distilled water (tap water contains chemicals that can degrade soldering iron tips). This will enhance solderability so you do not need to apply as much heat or force to make the connection.

The first thing to do when you see a trace or pad lifting off of the circuit board is to STOP! Do not aggravate the problem further; the worst thing you can do is cause the entire trace to peel back by continuing to pull on



Figure E-1: Left, arrow points to the original pad that is being soldered. Right, pad has been torn off through excess heat and force.

the wire. Remove the wire, if it is still connected, by barely touching the soldering iron to the joint and letting the wire fall off. Figure E-1 illustrates such a disaster scene.

The strategy for recovering from a broken trace is to remove the soldermask, fix the trace with a jumper wire, and find an alternate point for soldering by following the trace to a nearby component or via.

Removing the soldermask reveals the underlying copper traces. A short jumper wire can be soldered to these bared traces to fix the discontinuity caused by the torn trace. The bare region also serves as a convenient starting point for using a continuity meter to find an alternate point for affixing the jumper wire. Remove the soldermask using either a fine-grit (200 or finer) sandpaper, or by scraping the surface with a sharp hobbyists knife. When removing the soldermask, be careful not to catch pieces of the broken trace and further tear the trace of the board. Once the soldermask has been removed, clean the region with a gentle solvent, such as rubbing alcohol,

図 5: Figure E-1: 左、はんだ付けしていた元のランドを矢印で示す。右、過剰な熱と力によってランドが引き剥がされた状態。

図 E-1 左、はんだ付けしていた元のランドを矢印で示す。右、過剰な熱と力によってランドが引き剥がされた状態。

断線した配線を復旧するための戦略は、はんだレジスト (soldermask) を除去し、ジャンパーワイヤーで配線を修復し、配線をたどって近くの部品やビアに代替のはんだ付け点を探すことだ。

はんだレジストを除去すると、下にある銅の配線が露出する。この露出した配線に短いジャンパーワイヤーをはんだ付けすることで、引き剥がされた配線による断線を修復できる。露出した部分は、導通メータを使って代替のはんだ付け点を探す出発点としても役立つ。はんだレジストを除去するには、細かい番手 (200 番以上) のサンドペーパーか、鋭いホビーナイフで表面を削る。はんだレジストを除去する際は、剥がれた配線の断片を引っかけてさらに引き剥がさないように注意しよう。はんだレジストを除去したら、綿棒に消毒用アルコールなどの穏やかな溶剤を付けてその部分を清潔にする。次に、露出した配線に非常に薄いはんだフラックス層を塗り、清潔なはんだごての先端を露出した配線に沿って擦る。ごての先端に付いた少量のはんだが基板に吸われて配線をコーティングし、露出した銅の酸化を防ぐ。ごての先端が清潔すぎる場合は、少量のはんだを付けて、濡れたスポンジで先端を軽く拭いてからやり直してみよう。溶けたはんだのボールを先端に乗せて露出した配線をティンニングしようとしてはいけない——余分なはんだが積もってショートにつながる。(はんだフラックスは配線に均一で薄いはんだコーティングをするために不可欠だ。フラックスの塗布を省略しないこと。) 図 E-2 は、ティンニング前後の配線の様子を示している。

using a cotton swab. Then, apply a very thin layer of soldering flux to the region and rub a clean soldering iron tip along the exposed traces. Small amounts of solder sticking to the iron's tip will wick onto the circuitboard and coat the traces, preventing oxidation of the bare copper. If the iron's tip is too clean, apply a drop of solder to it and lightly wipe the tip off on a wet sponge and try again. Do not attempt to tin the exposed traces with a ball of molten solder on the tip. Excess solder will be deposited that can lead to

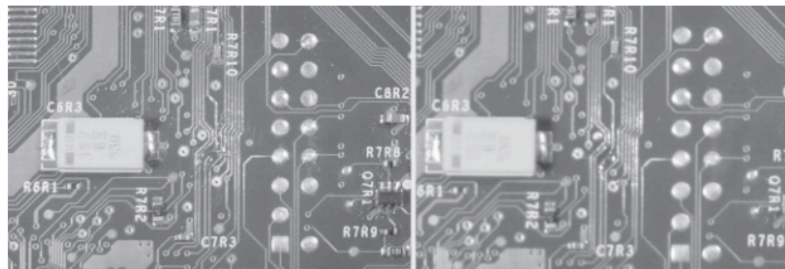


Figure E-2: Left, region after soldermask has been removed with fine-grit sandpaper. Right, region after it has been tinned (reconditioned for soldering).

shorts. (Note that the soldering flux is essential for getting a uniform, thin coating of solder on the traces. Do not skip the application of the solder flux.) Figure E-2 illustrates what the traces will look like before and after the tinning process.

At this point, you may want to use a continuity meter to determine an alternate point for attaching your modification wire. Most voltmeters come with an audible continuity meter function. When selected, a tone is emitted from the voltmeter whenever the resistance between the probes is very low.

Vias and component leads both make good alternate attachment points. If you decide to use a via, you must scrape the solder mask off and

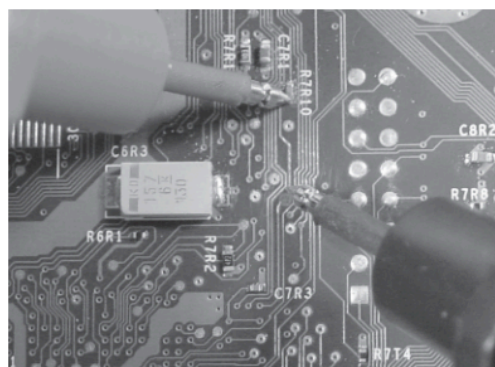


Figure E-3: Using a continuity meter to find an alternate attachment point. In this case, R7R10 turns out to be a good alternate.

図 6: Figure E-2: 左、細かい番手のサンドペーパーではんだレジストを除去した後の状態。右、ティンニング（はんだ付け処理）を施した後の状態。

図 E-2 左、細かい番手のサンドペーパーではんだレジストを除去した後の状態。右、ティンニング（はんだ付けしやすくする処理）を施した後の状態。

この時点で、導通メータを使って改造ワイヤーを取り付ける代替点を探してもいい。ほとんどのボルトメータには音の出る導通メータ機能が付いており、プローブ間の抵抗が非常に低いときにボルトメータから音が出る。ビアや部品のリードはどちらも良い代替接続点になる。

ビアを使う場合は、改造ワイヤーを取り付ける前にはんだレジストを削り取り、ビアを処理する必要がある。図 E-3 は導通メータを使って代替のはんだ付け点を見つけているところを示している。

図 E-3 導通メータを使って代替の接続点を探している様子。この場合、R7R10 が良い代替点として見つかった。

(図 E-3 は上の画像 (図 E-2 と同一ページ) を参照)

複数のビアをたどらないと最適な代替接続点が見つからない場合もあることを覚えておこう。

次のステップは、断線した配線の上に短いジャンパーワイヤーを取り付けることだ。断線した配線の周辺に、はんだフラックスをもう少し多めに塗る。問題の隙間とだいたい同じ長さの細いワイヤー (30 ゲージ程度) を切る。フラックスの粘着性を利用してワイヤーを位置合わせしながら、隙間の上にワイヤーを置く。ピンセットでワイヤーを押さえ、両側が断線した配線の端にくっつくまではんだごてで熱を加える。ピンセットでそっと押してワイヤーが所定の位置にあるか確認しよう——ワイヤーが動いてはいけない。また、導通メータで隣接する配線へのショートがないか確認しよう。ショートが見つかった場合は、ジャンパーが外れるまで加熱してやり直せばいい。図 E-4 は修理後の配線の様子を示している。

最後に、先ほど導通メータで見つけた代替のはんだ付け点に改造ワイヤーを取り付ける。

condition the via prior to attaching the modification wire. Figure E-3 illustrates using a continuity meter to find an alternate soldering point. Keep in mind that sometimes you will have to trace through several vias to find the best alternate attachment point.

The next step is to attach a short jumper wire across the broken trace. Apply a touch more soldering flux over the region of the broken trace. Cut a piece of fine wire (about 30-gauge) that is about the length of the gap in question. Place the wire over the gap, using the stickiness of the soldering flux to aid the placement process. Hold the wire in place with a pair of tweezers, and apply heat with the soldering iron until both sides have bonded to the edges of the broken trace. Verify that the wire is in place by gently pushing on it with the tweezers; the wire should not move. Also inspect for shorts to neighboring traces using your continuity meter. If a short is discovered, simply heat the jumper until it falls off the board and try again. Figure E-4 illustrates what the repaired trace looks like.

Finally, attach the modification wire to the alternate soldering point that was discovered previously using the continuity meter.

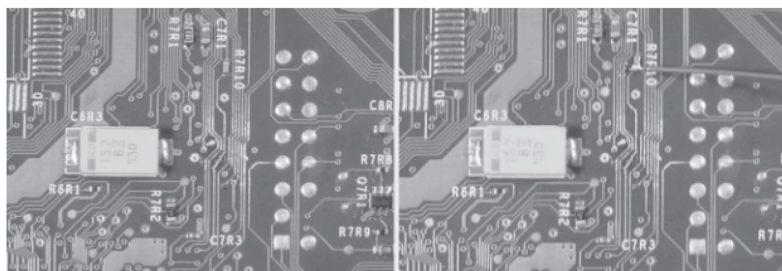


Figure E-4: Left, a jumper has been installed over the damaged trace. Right, the modification wire has been successfully attached to the alternate soldering point.

図 7: Figure E-4: 左、破損した配線にジャンパーを設置した状態。右、代替のはんだ付け点に改造ワイヤを取り付けた状態。

図 E-4 左、破損した配線にジャンパーを設置した状態。右、代替のはんだ付け点に改造ワイヤーを取り付けた状態。

付録 F Xbox ハードウェアリファレンス

この付録では、Xbox ハードウェアに使用されている主要なコネクタのピン配置をまとめる。

電源コネクタのピン配置

Xbox に使用されている電源は最大 96 ワット定格のスイッチング電源で、10 秒未満であれば 160 ワットのピーク出力能力を持つ。マイクロソフトはこの電源を Delta Electronics, Inc. (www.deltaww.com) をはじめ複数のベンダーから調達している。デルタ製は米国向け Xbox に使用されており、データシートは同社ウェブサイトまたはウェブ検索から入手できる。

ピン番号	説明	ワイヤーの色
1	+12V	黄
2	+5V	赤
3	+5V	赤
4	+5V	赤
5	+3.3V	オレンジ
6	+3.3V スタンバイ	茶
7	GND	黒
8	GND	黒
9	GND	黒
10	GND	黒
11	電源オン	白
12	電源 OK	青

表 F-1: メイン電源コネクタのピン配置。ワイヤーの色は Xbox に搭載されている電源モデルによって若干異なる場合がある。この表は Delta 製 DPSN-96AP A バージョン電源に適用される。

APPENDIX F

Xbox Hardware Reference

This appendix summarizes the pinouts of the major connectors employed in the Xbox hardware.

Power Supply Pinout

The power supply used in the Xbox is a switcher rated at 96 watts maximum, with a peak pulse capability of 160 watts for less than 10 seconds. Microsoft buys this power supply from multiple vendors, including Delta Electronics, Inc. (www.deltaww.com). Delta's is used in United States Xboxes, and you can find a datasheet for this part through their website or through a web search.

Pin	Description	Wire Color
1	+12V	Yellow
2	+5V	Red
3	+5V	Red
4	+5V	Red
5	+3.3V	Orange
6	+3.3V Standby	Brown
7	GND	Black
8	GND	Black
9	GND	Black
10	GND	Black
11	Power On	White
12	Power OK	Blue

Table F-1: Main power connector pinouts. Wire colors may vary slightly depending upon the specific power supply model used in your Xbox. This table applies to the Delta DPSN-96AP A version power supply.

ハードディスク電源コネクタのピン配置

説明	ワイヤーの色
+12V	黄
GND	黒
GND	黒
+5V	赤

表 F-2：ハードディスク電源コネクタのピン配置。

ビデオコネクタのピン配置

ビデオコネクタのピン配置は、一部の信号をプローブで確認してもわかりやすいパターンが現れないこと、また1つのコネクタで複数の表示モードに対応していることから、解析が少々難しい。ビデオコネクタのピン配置を掲載しているウェブサイトはいくつかあるが、掲載情報と実測値を照らし合わせると食い違いが見られる。僕はXboxHacker BBSとSourceforge.netのucon64ウェブページの2つの投稿を参照・照合して、できる限り正確な情報をまとめた。元の投稿は<http://www.xboxhacker.net/index.php?do=article&id=10&page=1>と<http://ucon64.sourceforge.net/ucon64misc/conn.html>で確認できる。また、MODE1～3 信号で選択できる 8 つのビデオモードの定義は、XboxHacker BBS のウェブページに記載されている。実測の結果、コンポジットビデオとオーディオ信号のマッピングはすべて正しいと確認できたが、XboxHacker BBS の投稿に記載されているSDTV、HDTV、RGBのマッピングは検証できなかった。これらの信号に誤りがあった場合はあらかじめお詫びする。

なお、ピン 12 とピン 24 は Xbox のコネクタ側のピンが長くなっており、Xbox の電源が入った状態でビデオコネクタに周辺機器を接続する「ホットインサクション」時に電力を供給するためのものである。ピンが長いことで、信号を受け取る前に周辺機器の回路が電源を確保できる。これにより、周辺機器のチップ内部で発生する可能性のある「ラッチアップ」と呼ばれる破壊的な状態を防ぐ。

Description	Wire Color
+12V	Yellow
GND	Black
GND	Black
+5V	Red

Table F-2: Hard disk power connector pinouts.

Video Connector Pinout

The video connector pinout is a little bit of a mystery because some of its signals show no obvious or recognizable signal patterns when probed, and because multiple display modes are supported by a single connector. There are a few websites that post pinouts for the video connector, but cross-checking the posted information with measurements reveals some discrepancies. I have done my best here to piece together and reconcile two separate postings from the XboxHacker BBS and the ucon64 web page at Sourceforge.net. The original postings can be found at <http://www.xboxhacker.net/index.php?do=article&id=10&page=1> and at <http://ucon64.sourceforge.net/ucon64misc/conn.html>. The definition of all eight video modes selectable by the MODE1-3 signals is listed on the XboxHacker BBS web page. My measurements indicate that all of the composite video and audio signal mappings are correct, but I was unable to verify the SDTV, HDTV, and RGB mappings as given by the Xboxhacker BBS posting. I apologize in advance if any of these signals are incorrect.

Note that pins 12 and 24 have longer pins on the Xbox's connector, which indicate that they are used to supply power to peripherals attached to the video connector during hot-insertion events. The longer pins allow a peripheral's circuitry to power up before receiving signals in the event that the peripheral is connected while the Xbox is powered on. This helps prevent a potentially destructive situation inside the peripheral's chips called *latch-up*.

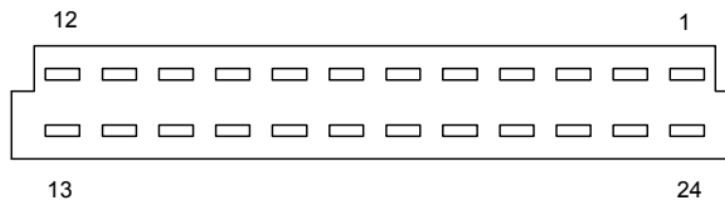


Figure F-1: Xbox audio-video connector, as viewed while looking at the Xbox back panel from the outside.

图 9: Figure F-1: Xbox audio-video connector

図 F-1 : Xbox のオーディオ・ビデオコネクタ。Xbox の背面パネルを外側から見た状態。

ピン番号	信号名	I/O	説明
1	Right Audio	Out	右チャンネル音声出力
2	GND	Power	
3	SPDIF	Out	Sony/Philips Digital Interface (S/PDIF) 音声出力
4	VSYNC	Out	垂直同期 (VGA 出力モード)
5	GND	Power	
6	GND	Power	
7	GND	Power	
8	GND	Power	
9	Pb / B	Out	HDTV モードでは Pb、RGB モードでは青
10	GND	Power	
11	Y / G	Out	SDTV および HDTV モードでは Y 信号、RGB モードでは緑
12	GND	Power	ホットインサーション用の長いピン
13	CVIDEO	Out	コンポジットビデオ出力
14	GND	Power	
15	C / Pr / R	Out	SDTV では C 信号、HDTV では Pr、RGB モードでは赤
16	GND	Power	
17	STATUS	Out	SCART (ス カ ー ト — Syndicat des Constructeurs d'Appareils Radio Récepteurs et Téléviseurs) ステータスピン
18	MODE3	In	ビデオ出力モード選択ピン 3
19	MODE2	In	ビデオ出力モード選択ピン 2
20	MODE1	In	ビデオ出力モード選択ピン 1
21	HSYNC	Out	水平同期 (VGA 出力モード)
22	GND	Power	左チャンネル音声ケーブルシールド
23	Left Audio	Out	左チャンネル音声出力
24	+5V	Power	+5V 電源、ホットインサーション用の長いピン

表 F-3：ビデオコネクタのピン配置。

表 F-3 の Xbox ビデオコネクタのピン番号は、図 F-1 のとおり。XboxHacker 由来のピン配置と ucon64 由来のピン配置では番号の付け方が異なっており、僕はその中間的な番号体系を採用した。興味深いことに、図 F-1 のピン 24 は Xbox ビデオコネクタ上で四角いパッドに対応しており、ここで採用した番号体系がメーカーの番号体系と一致していないことを示している（四角いパッドは通常ピン 1 を、丸いパッドはその他のピンを示す）。ピンの番号付けは任意のものであり、ピン定義表と一貫性があれば問題ないため、表の正確性には影響しない。

USB コネクタのピン配置

Xbox はゲームコントローラーポートにおいて、USB 派生の仕様を採用している。Xbox 前面には 4 つのゲームコントローラーポートがあり、すべて同一のピン配置となっている（図 F-2 参照）。

USB Connector Pinout

The Xbox uses a USB derivative for the game controller ports. There are four game controller ports on the front of the Xbox, and all of them have an identical pinout, as shown in Figure F-2.

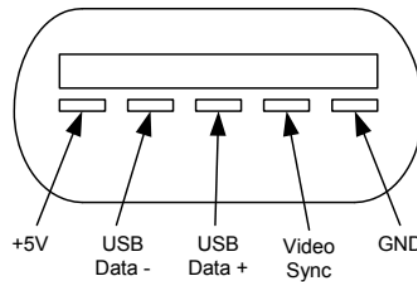


Figure F-2: Game controller pinout, as viewed from outside the Xbox case looking at the connector.

The “video sync” signal is a 3.3V CMOS or TTL-compatible signal. It is a basic 15.734 kHz positive polarity pulse train synchronized to the horizontal line time of the composite video output, with a single longer pulse at the beginning of every video field. This signal enables peripherals that are pointed at the TV screen, such as a light pen or a light gun for shooting games, to derive position information.

The game controllers are connected to the Xbox via an intermediate break-away connector. The purpose of this break-away connector is to prevent console damage (particularly hard drive damage) by dragging or jerking in the event that a cord becomes entangled around a user’s foot. The pinout of this break-away connector is depicted in Figure F-3.

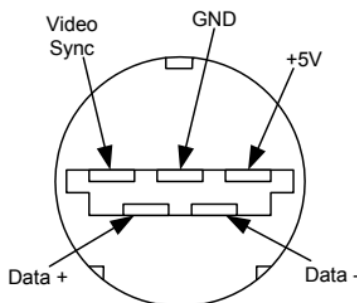


Figure F-3: Game controller break-away pinout, as viewed looking head-on into the connector closer to the Xbox.

図 F-2 : Xbox のケース外側からコネクタを見た状態のゲームコントローラーピン配置。
図 F-3 : Xbox に近い側のコネクタを正面から見た状態のゲームコントローラー着脱式コネクタのピン配置。

「ビデオ同期」信号は 3.3V CMOS/TTL 互換の信号である。これはコンポジットビデオ出力の水平ライン時間に同期した、15.734 kHz の正極性パルス列であり、各ビデオフィールドの先頭には 1 つの長いパルスが挿入される。この信号により、TV スクリーンに向けて使用するライトペンやシューティングゲーム用ライトガンなどの周辺機器が位置情報を取得できる。

ゲームコントローラーは Xbox と着脱式コネクタ（ブレイクアウェイコネクタ）を介して接続される。このコネクタの目的は、ケーブルがユーザーの足に引っかかって引っ張られたりした際に、コンソール（特にハードディスク）へのダメージを防ぐことにある。着脱式コネクタのピン配置を図 F-3 に示す。

ゲームコントローラー拡張スロットのピン配置

Xbox のゲームコントローラーにはメモリカード、マイクロフォンなどの周辺機器向けに 2 つの拡張スロットが搭載されている。これらのスロットも USB 準拠のインターフェースを提供する。ゲームコントローラーには USB ハブ（Atmel AT43USB401 ハブチップ）が内蔵されており、入力された USB 信号を拡張スロットへ中継する。拡張コネクタのピン配置を図 F-4 に示す。

The Xbox game controller features two expansion slots for memory cards, microphones and other peripherals. These slots also provide a USB-compliant interface. The game controller contains a USB hub (an Atmel AT43USB401 hub chip) that repeats the incoming USB signal to the expansion slots. The pinout of the expansion connector is illustrated in Figure F-4.

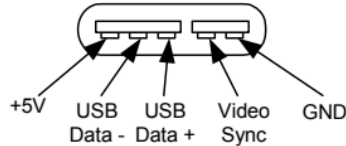


Figure F-4: Game controller expansion slot pinout, as viewed looking into the game controller slot with the buttons facing up.

Ethernet Connector Pinout

The Ethernet port on the Xbox is a standard 10/100 base-TX twisted pair RJ-45 connector. The pinouts and colors of Table F-4 are based on the EIA/TIA 568B standard. Figure F-5 illustrates the pin numbering of the connector.

Pin	Description	Wire Color
1	Transmit +	Orange stripe
2	Transmit -	Orange
3	Receive +	Green stripe
4	<i>Not connected</i>	Blue
5	<i>Not connected</i>	Blue stripe
6	Receive -	Green
7	<i>Not connected</i>	Brown stripe
8	<i>Not connected</i>	Brown

Table F-4: Ethernet 10/100 RJ-45 pinout.

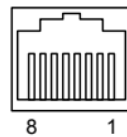


Figure F-5: Pinout of the Xbox ethernet connector, seen looking from the outside toward the back panel of the Xbox.

図 F-4：ボタンを上にした状態でゲームコントローラースロットを正面から見た状態の、ゲームコントローラー拡張スロットのピン配置。図 F-5：Xbox の背面パネルを外側から見た状態の Xbox イーサネットコネクタのピン配置。

イーサネットコネクタのピン配置

Xbox のイーサネットポートは EIA/TIA 568B 規格に基づく標準的な 10/100BASE-TX ツイストペア RJ-45 コネクタである。表 F-4 のピン配置とワイヤーカラーは EIA/TIA 568B 規格に基づく。コネクタのピン番号は図 F-5 のとおり。

ピン番号	説明	ワイヤーの色
1	送信+	オレンジストライプ
2	送信-	オレンジ
3	受信+	グリーンストライプ
4	未接続	青
5	未接続	青ストライプ
6	受信-	グリーン
7	未接続	茶ストライプ
8	未接続	茶

表 F-4：イーサネット 10/100 RJ-45 のピン配置。

ATA コネクタのピン配置

Xbox はハードドライブおよび DVD ドライブとの通信に標準的な ATA (Advanced Technology Attachment) バスを使用している。ATA バスは一般的に (ただし技術的には不正確な表現で) IDE (Integrated Drive Electronics) バスと呼ばれることが多い。現在の多くのドライブは IDE スタイルドライブに該当するが、例えば SCSI ドライブもドライブ内蔵エレクトロニクスを持つ。しかし長年の (誤) 用例により、IDE という言葉は ATA バスの同義語として定着している。

表 F-5 は ATA コネクタのピン配置で、Xbox マザーボード上で Xbox の背面を自分に向けた状態でコネクタを上から見たもの (コネクタは右側にあるはず) である。奇数ピンが片側に、偶数ピンがもう片側に並ぶジグザグ型のピン番号付けに注目してほしい。

ピン番号	信号名	説明	ピン番号	信号名	説明
1	Reset	リセット	2	Ground	GND
3	Data 7	データ 7	4	Data 8	データ 8
5	Data 6	データ 6	6	Data 9	データ 9
7	Data 5	データ 5	8	Data 10	データ 10

9	Data 4	データ 4	10	Data 11	データ 11
11	Data 3	データ 3	12	Data 12	データ 12
13	Data 2	データ 2	14	Data 13	データ 13
15	Data 1	データ 1	16	Data 14	データ 14
17	Data 0	データ 0	18	Data 15	データ 15
19	Ground	GND	20	Key	極性合わせ用の空きピン
21	DMARQ	DMA リクエスト	22	Ground	GND
23	DIOW-	I/O 書き込み	24	Ground	GND
25	DIOR-	I/O 読み込み	26	Ground	GND
27	IORDY	I/O 準備完了	28	CSEL	ケーブルセレクト
29	DMACK-	DMA 応答	30	Ground	GND
31	INTRQ	割り込みリクエスト	32	IOCS16-	16 ビット I/O
33	DA1	デバイスアドレスビット 1	34	PDIAG-	診断通過
35	DA0	デバイスアドレスビット 0	36	DA2	デバイスアドレスビット 2
37	CS0-	チップセレクト 0	38	CS1-	チップセレクト 1
39	DASP-	デバイスアクティブ/スレーブ存在	40	Ground	GND

表 F-5 : ATA コネクタのピン配置。

DVD-ROM 電源コネクタ

Xbox は DVD-ROM 専用の電源コネクタを使用している。このコネクタは電力を供給するだけでなく、いくつかの制御・ステータス信号も伝達する。これらの信号はドライブおよびドライブトレイの状態を伝える。ここに記載するピン配置は Xboxhacker BBS の情報に基づいており、元の投稿（Ken Gasper による）は <http://www.xboxhacker.net/forums/index.php?act=ST&f=5&t=1025&s=0755f2b600975b776552f93d0730e4b1> で確認できる。

Xbox マザーボードを上から見た状態のコネクタのピン番号は図 F-6 のとおり。

DVD-ROM Power Connector

The Xbox uses a proprietary DVD-ROM power connector. This connector not only brings power, but it also carries a few control and status signals. These signals convey information about the state of the drive and the drive tray. The pinouts given here are from the Xboxhacker BBS, and the original post by Ken Gasper that this pinout is based on can be found at <http://www.xboxhacker.net/forums/index.php?act=ST&f=5&t=1025&s=0755f2b600975b776552f93d0730e4b1>

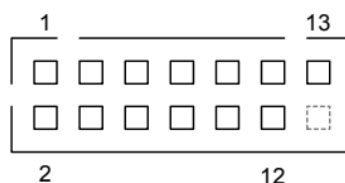


Figure F-6: DVD-ROM power connector pin numbering, as viewed looking down at the Xbox motherboard.

The connector pin numbering, as viewed looking down at the Xbox motherboard, can be found in Figure F-6, and the pinout in Table F-6.

Pin	Name	Comment	Pin	Name	Comment
1	12VDC	+12 Volts power	2	5VDC	+5 Volts power
3	GND	Current return, reference	4	EJECT-	Active low tray eject
5	TS0	Traystate status 0	6	TS1	Traystate status 1
7	TS2	Traystate status 2	8	ACTIVITY-	Disk seek/data transfer
9	12VDC	+12 Volts power	10	5VDC	+5 Volts power
11	GND	Current return, reference	12	GND	Current return, reference
13	Key	Not connected			Blank for polarizing

Table F-6: DVD power connector pinout (viewed on motherboard).

図 F-6 : Xbox マザーボードを上から見た状態の DVD-ROM 電源コネクタのピン番号。

ピン番号	信号名	説明	ピン番号	信号名	説明
1	12VDC	+12V 電源	2	5VDC	+5V 電源
3	GND	電流リターン・基準	4	EJECT-	アクティブロー ト レイジェクト
5	TS0	トレイステータス 0	6	TS1	トレイステータス 1
7	TS2	トレイステータス 2	8	ACTIVITY-	ディスクシーク／ データ転送
9	12VDC	+12V 電源	10	5VDC	+5V 電源
11	GND	電流リターン・基準	12	GND	電流リターン・基準
13	Key	未接続・極性合わせ用ブランク			

表 F-6 : DVD 電源コネクタのピン配置 (マザーボード上から見た状態)。

LPC コネクタ

Xbox には LPC (Low Pin Count) バスをベースとしたデバッグ・テスト用ポートが搭載されている。このバスは Intel が当初サウスブリッジチップセット向けに定義したもので、コストを抑えるためにピン数を削減しながら、レガシー PC I/O 機能のサポートを維持する。これらのレガシー I/O 機能はかつてほぼ廃止された ISA バス上に存在していたものであり、キーボード、マウス、シリアルポート、パラレルポート、ブート ROM などが含まれる。Intel の LPC バス仕様は <http://www.intel.com/design/chipsets/industry/25128901.pdf> で確認できる。

LPC デバッグコネクタは特に重要で、内蔵 ROM が存在しない場合や破損して ROM が存在しない・空白のように見える場合に、代替 ROM イメージを Xbox に供給できる。この機能は、インストールが容易な Xbox 用の代替ブート ROM を実現するために活用されてきた。

Xbox の LPC デバッグコネクタのピン配置は、Intel の Installable LPC Debug Module Design Guide (http://www.intel.com/technology/easeofuse/LPC_mod_spec72.pdf) をベースにしており、表 F-7 に記載する一部の変更が加えられているようだ。特にピン 16 の機能は不明であり、その対となるピン 15 が Xbox マザーボードではパワーピンに再割り当てされている。ピン 15 のパワーピンとしての割り当ては、そのピンに太いトレースと近くにデカップリングコンデンサが配置されていることから推定される。もしピン 15 が常時ハイの SPDA1 信号として使われるのであれば、電源コンディショニングのない細いトレースが使われていたはずだ。

ピン番号	信号名	説明	ピン番号	信号名	説明
1	LCLK	33 MHz クロック	2	VSS	電流リターン
3	LFRAME#	LPC トランザクションの開始・終了	4	KEYWAY	極性合わせ用ブランク
5	LRST#	LPC リセット	6	VCC5	+5V 電源
7	LAD3#	多重化アドレス／データ	8	LAD2#	多重化アドレス／データ
9	VCC3	+3.3V 電源	10	LAD1#	多重化アドレス／データ
11	LAD0#	多重化アドレス／データ	12	VSS	電流リターン
13	SCL	I2C シリアルクロック	14	SDA	I2C シリアルデータ
15	VCC3	+3.3V 電源 (Intel 仕様では SPDA1)	16	SPDA0	シリアル EEPROM デバイス選択 (?)

表 F-7：LPC コネクタのピン配置（マザーボード上から見た状態）。

ファンコネクタ

Xbox のファンコネクタは 3 ピンヘッダで、ピン 1 とピン 3 が温度調節式 PWM（パルス幅変調）ファン速度コントローラーに接続され、ピン 2 は +12V 電源に接続されている。

フロントパネルコネクタ

Xbox のフロントパネル機能——点滅する LED、電源スイッチ、イジェクトスイッチ——はフロントパネルコネクタを通じて Xbox マザーボードに接続されている。このコネクタのピン配置を表 F-8 に示す。ピン番号は Xbox マザーボード上のコネクタを上から見た状態のものである。

ピン番号	説明	ピン番号	説明
1	GND	2	電源スイッチ
3	GND	4	イジェクトスイッチ
5	緑 LED	6	赤 LED
7	赤 LED	8	緑 LED
9	未接続だが配線あり	10	ピンなし（極性合わせ）

表 F-8：フロントパネルコネクタ（マザーボード上から見た状態）。

ライセンスとクレジット

コンテンツライセンス

原著ライセンス

本サイトで翻訳・掲載している原著 Hacking the Xbox: An Introduction to Reverse Engineering は、著作権者 **Xenatera LLC**（著者 Andrew “bunnie” Huang）および出版社 No Starch Press により **Creative Commons Attribution-NonCommercial-ShareAlike 1.0 (CC BY-NC-SA 1.0)** の下で公開されています。

Copyright © 2003 Xenatera LLC. Licensed under CC BY-NC-SA 1.0 for the free edition.
Published by No Starch Press, San Francisco, CA.

- 原著無償公開の告知: <https://nostarch.com/xboxfree>
- CC BY-NC-SA 1.0 ライセンス全文: <https://creativecommons.org/licenses/by-nc-sa/1.0/>

本サイトのコンテンツ

コンテンツ種別	ライセンス
日本語翻訳テキスト	CC BY-NC-SA 1.0（原著ライセンスに準拠）
原著から抽出した図版・画像	CC BY-NC-SA 1.0（原著ライセンスに準拠）
サイト設定・スクリプト等	MIT License

CC BY-NC-SA 1.0 の要点

BY（表示）：原著著作権者 Xenatera LLC・著者 Andrew “bunnie” Huang・出版社 No Starch Press へのクレジット表示が必要です。

NC（非営利）：商業目的での使用は禁止されています。

SA（継承）：本コンテンツを改変・再配布する場合は、同一ライセンス (CC BY-NC-SA 1.0 またはそれ以降) の下で公開してください。

再利用について

本サイトのコンテンツを再利用・二次配布する場合は：

1. 著作権者 Xenatera LLC・著者 Andrew “bunnie” Huang・出版社 No Starch Press のクレジットを表示してください。
2. 非営利目的に限定してください。
3. 同一ライセンス (CC BY-NC-SA) を継承してください。

免責

本翻訳は原著の意図を忠実に再現するよう努めていますが、翻訳者は翻訳の完全な正確性を保証しません。技術的な目的には必ず原著 (PDF) を参照してください。

クレジット・謝辞

原著

項目	内容
原題	Hacking the Xbox: An Introduction to Reverse Engineering
著者	Andrew “bunnie” Huang
著作権者	Xenatera LLC
出版社	No Starch Press
出版年	2003 年
ISBN	1-59327-029-1
無償公開	2013 年 (Aaron Swartz 追悼)
原著ライセンス	Creative Commons BY-NC-SA 1.0

原著の無償版は No Starch Press 公式サイト および archive.org から入手できます。

本書の著作権は Xenatera LLC に帰属します。

Copyright © 2003 Xenatera LLC. All rights reserved.

Published by No Starch Press, San Francisco, CA.

翻訳プロジェクト

本翻訳はニコ技深圳コミュニティ / 高須正和による日本語訳コントリビューションです。翻訳者は原著の権利を主張しません。

本翻訳に対して適切なクレジット表示を行う場合は、以下のようにしてください：

Hacking the Xbox ニコ技深圳コミュニティ / 高須正和による日本語訳 — 原著: Andrew “bunnie” Huang (Xenatera LLC / No Starch Press, 2003)、翻訳・レビュー: ニコ技深圳コミュニティ / 高須正和 (@tks) — <https://takasumasakazu.net>、ライセンス: CC BY-NC-SA 1.0

免責事項

- ・ 本翻訳は著者 Andrew “bunnie” Huang、著作権者 Xenatera LLC、および出版社 No Starch Press による公式日本語版ではありません。著者 bunnie には翻訳プロジェクトへの歓迎コメントをいただいています。

- 翻訳の正確性については最善を尽くしていますが、技術的内容の最終確認は原著 (PDF) を参照してください。
- 本サイトは商業目的では運営していません。広告収入を得ていません。