NEON シリーズ

プログラム実行フロー解説

起動からデモループ、終了までをソースコードから追う

対象: 初代 NEON / NEON2 / NEON3
通常版・256 色版・80286 版の共通骨格と差分

参照方針 現在の VSYNC IRQ 駆動化後ソースを主対象とし、初代 NEON は NEON_2_3 系、NEON2 は NEON2_1_1 系、NEON3 は現行 16 色/256 色/286 系を参照しています。説明は共通骨格を中心にし、世代・描画版固有の違いは該当箇所で注記します。

作成日: 2026-08-08

目次

1. 全体をフラットに見渡す

NEON シリーズの各版は、描画方式やシーン内容が異なっても、プログラム全体の寿命はほぼ同じ形です。COM プログラムとして開始し、実行条件を確定し、画面・文字・音源・時間管理を初期化してからデモループへ入り、終了要求を検出したら逆順にハードウェア状態を戻して DOS へ帰ります。

重要なのは、プログラムを「描画プログラム」「音源ドライバ」「VSYNC 処理」のどれか一つとして見るのではなく、複数のサブシステムが一つの論理フレーム番号を共有して進むデモプログラムとして見ることです。第 1 章では各処理を同じ粒度で並べ、詳細な内部事情は後の章へ回します。

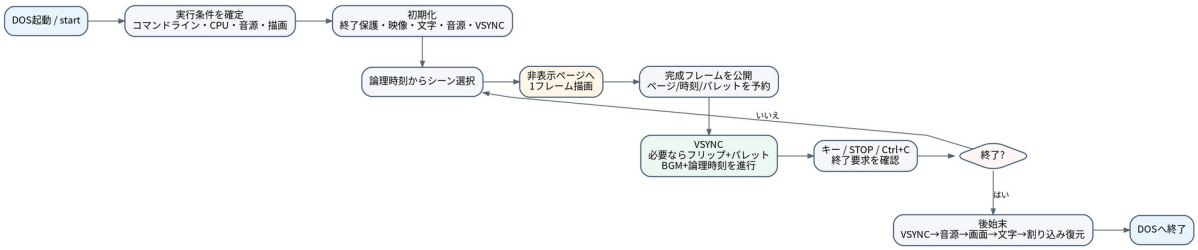

図 1-1　NEON シリーズの起動から終了までの全体フロー

段階	主な仕事	代表的な状態/データ
起動	DS/ES をプログラムセグメントへ揃え、タイトルを表示	PSP のコマンドライン、各オプション
条件確定	コマンドライン、CPU 機能、音源、描画モードを決定	sound_mode, video_backend, frame_skip, speed_percent
初期化	終了保護、グラフィック、文字 VRAM、音源、VSYNC 時間管理を準備	draw_page, sound_present, scheduler flags
フレーム生成	現在の frame_counter からシーンを決め、非表示ページへ描画	scene_index, scene_frame, draw_page
表示/時間進行	完成ページがあれば VSYNC で表示し、BGM と論理時刻を進める	vsync_frame_ready, frame_counter, music_index
入力確認	通常キーまたは STOP/Ctrl+C の終了要求を確認	break_requested
終了	IRQ、音源、画面、カーソル、割り込みベクタを復元	元 INT 0Ah/06h/23h、PIC 状態

SOURCE　NEON: NEONRUN.ASM:58-255 / NEON2: neon2.asm:33-203 / NEON3: NEON3256.ASM:28-214

現在のソースでは、デモ本体には二つの実行主体があります。メイン処理は重い描画を担当し、VSYNC 割り込みは表示タイミングと実時間側の進行を担当します。ただしこれは全体の一部であり、起動前の検出・初期化や、終了後の復元まで含めて一つのライフサイクルになっています。

用語 本書で「論理時刻」と呼ぶのは frame_counter を中心としたデモ内部の時間です。「実時間」はディスプレイの VSYNC 周期です。/SPEED は前者の進み方を変え、/AFS や /FS は主として映像フレーム生成のタイミングを調整します。

2. COM 起動と初期設定

2.1 start: COM プログラムの入口

各版は `org 100h` の COM 形式で、先頭の `jmp start` から共通した初期化列へ入ります。最初に CS を DS/ES へコピーし、文字列・テーブル・変数を同一セグメントで扱える状態にします。`cld` でストリング命令の方向も前進に固定します。

```
start:
    push cs
    pop  ds
    push cs
    pop  es
    cld
    ...
    call parse_command_line
```

SOURCE NEONRUN.ASM:58-69 / neon2.asm:33-44 / NEON3256.ASM:28-39

2.2 コマンドラインを読み、実行条件を確定する

`parse_command_line` は PSP の 80h 番地にある長さと 81h 以降の文字列を読み、空白単位でトークン化します。小文字は大文字へ変換され、`parse_option_token` が /AUTO、音源選択、/AFS、/FS、/SPEED、256 色版の PEGC バックエンド指定などを判定します。ヘルプ指定なら画面モードへ入る前に使用法を表示して終了します。

SOURCE NEON3 COMMAND256.INC:5-54, 56-150 以降。NEON/NEON2 も同系統の COMMAND/本体内パーサを使用。

2.3 CPU・音源を検出する

256 色版など CPU 依存の高速化を持つ版では `cpu_detect_features` が先に実行されます。続いて `sound_detect` が OPNA/YM2608、YM2203/26K、OPL3 を検査し、選択された構成を `sound_present` などへ確定します。検出結果に応じた既定音量を適用し、ユーザー向け音量をチップの TL 補正値へ変換します。音源が見つからなくてもプログラムはグラフィックのみで続行できます。

SOURCE NEON2 neon2.asm:53-64 / NEON3 NEON3256.ASM:48-59 / OPNA.INC:14-63, 65-166

2.4 終了保護、画面、文字、音源、VSYNC を順に立ち上げる

1. `break_guard_install` で PC-98 STOP (INT 06h) と DOS Ctrl+C (INT 23h) を「終了要求フラグを立てるだけ」のハンドラへ差し替える。
2. `video_enter` / `video16_enter` で対象版のグラフィックモードとページ構成を設定する。
3. テキストカーソルを隠し、`text_initialize` でテキスト VRAM 側の表示を準備する。
4. 音源が存在すれば `sound_initialize` で OPNA/OPL3 を初期化し、曲の先頭状態を作る。
5. `adaptive_clock_initialize` で VSYNC IRQ2 / INT 0Ah のスケジューラを導入し、実際に IRQ が動くか確認する。

SOURCE NEONRUN.ASM:130-145 / neon2.asm:105-118 / NEON3256.ASM:100-112; BREAKGUARD.INC:14-49; OPNA.INC:168-184

3. デモ本体のメインループ

初期化が終わると `.main_loop` へ入り、終了要求が来るまで同じ処理を繰り返します。通常は VSYNC スケジューラが有効な経路を通り、IRQ が使えなかった場合だけ `.legacy_main_loop` へ移ります。

3.1 通常の IRQ 動ループ

- 固定フレームスキップ時は `frame_skip_count` が 0 になるまで待つ。/AFS ではこの待ちを行わない。
- `vsync_frame_ready` が 0 になるまで待ち、前の完成フレームが VSYNC 側で表示されたことを確認する。
- 現在の `frame_counter` を基準に、非表示ページへ次のフレームを完成させる。
- `vsync_queue_completed_frame` で表示予定ページ、シーン時刻、パレットを公開する。
- VSYNC 側が公開済みフレームを受け取って `vsync_frame_ready` を 0 に戻すまで待つ。
- キー/終了要求を確認し、問題なければ `.main_loop` に戻る。

SOURCE NEONRUN.ASM:146-170 / neon2.asm:120-136 / NEON3256.ASM:114-138

図 3-1　メイン描画と VSYNC IRQ の役割分担

3.2 なぜ「完成したページ」を渡すのか

メイン処理は表示中のページとは別のページを描きます。線・ポリゴン・テキスト・シーン固有の演出を途中まで描いた状態で表示ページにしてしまうとティアリングや中間状態が見えるため、ページが完成してから `vsync_frame_ready=1` とします。VSYNC IRQ はこのフラグを見て初めてページを表へ出します。

この構造により、描画が 1 VSYNC で終わらない重い場面でも「描画途中の絵は出さない」という原則を保てます。時間の進め方はモードで異なり、/AFS では実 VSYNC ごとに音楽と映像時刻を進めて描画できなかったフレームを飛ばし、/FS:n では固定の論理フレーム進行に音楽も合わせるため、描画が遅い場合は音楽も同じだけ遅くなります。

3.3 終了チェックはメイン側で行う

STOP/Ctrl+C の割り込みハンドラ自身は後始末を行わず `break_requested` を立てるだけです。通常キーは DOS `INT 21h / AH=06h, DL=FFh` で直接ポーリングします。どちらも `.exit_demo` へ合流し、同じ後始末経路を通ります。

SOURCE NEONRUN.ASM:224-255 / neon2.asm:172-203 / NEON3256.ASM:185-214; BREAKGUARD.INC:88-92

4. シン選と 1 フレームの描画

4.1 frame_counter を scene_index / scene_frame に分解する

NEON シリーズでは、デモ全体の時刻を `frame_counter` 一つで持ちます。`select_scene` はこの全体フレーム番号を各シーンの長さで順に減算し、現在のシーン番号 `scene_index` と、そのシーン内の相対時刻 `scene_frame` に分解します。描画側はこの二つを使って形状やカメラ、演出の位相を決めます。

作品	総論理フレーム	シーン数	選択方法	代表ソース
NEON	3072	5	比較チェーン（384 + 640×4）	NEONRUN.ASM:777-821
NEON2	6144	12	scene_length_table をループ	SCENE256.INC:7-54
NEON3	6144	9	scene_length_table をループ	SCENE3_256.INC:5-62

シーン描画は `scene_routines` という near ポインタ表を `scene_index` で引き、`call word [scene_routines+bx]` する形が共通です。これにより、メインループは個々のシーン名を知る必要がありません。

```
SOURCE  NEONRUN.ASM:806-821 / NEON2 SCENE256.INC:27-54 / NEON3 SCENE3_256.INC:39-62
```

4.2 1 フレムのレンダリング工程

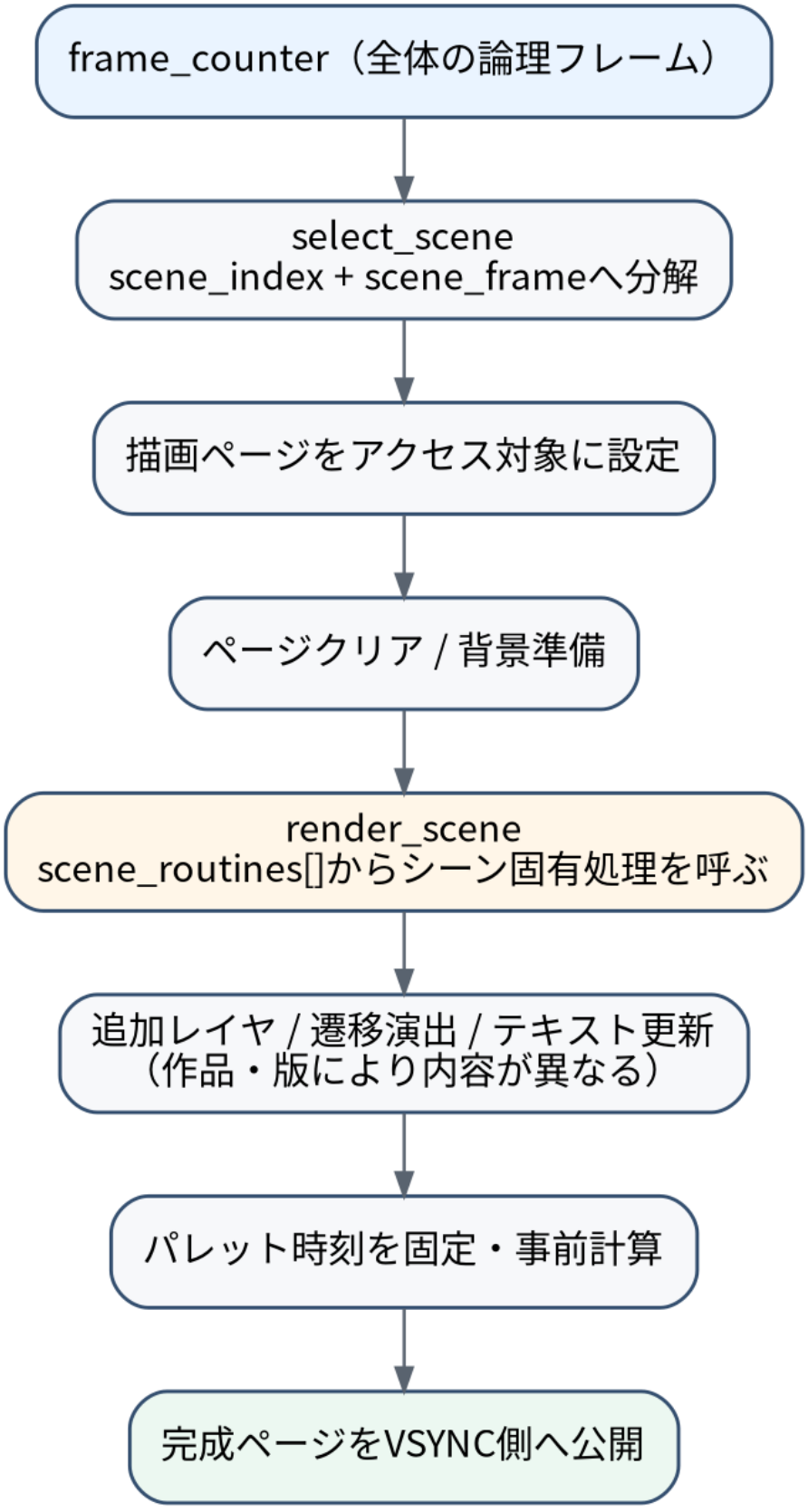

図 4-1　論理フレームから完成ページまでの代表的な流れ

NEON2 256 色版の `FRAME_RENDER256.INC` では、シーン確定 → パレット位相同期 → 描画ページ選択 → クリア → `render_scene` → Plane 固有レイヤ → 遷移 → 粒子 → テキスト、という順です。NEON3 は最終ロトズームでページを再利用する最適化があるため分岐が増えていますが、通常区間では scene 選択 → draw page 設定 → clear → render_scene → text_update という骨格です。

SOURCE　NEON2 FRAME_RENDER256.INC:1-16 / NEON3 FRAME_RENDER3_256.INC:9-54

4.3 世代が進むほど「シーンの中身」が厚くなる

初代 NEON は 5 つの大きな章を直接ディスパッチします。NEON2 は 12 章を音楽の 64 小節構成へ合わせたテーブルで管理し、NEON3 は 9 章の連続都市飛行としてカメラ状態の持続性が強くなっています。たとえば NEON3 では地下道路→鉄道の境界で、AFS により scene_frame=0 を描かない場合でも確実にカメラをリベースするため、シーン番号が 6 へ変化した瞬間を検出しています。

SOURCE　NEON3 SCENE3_256.INC:21-37（地下道路→鉄道のシーン変化検出）

5. 表示ページ・VRAM・パレット

5.1 描画ページと表示ページを分ける

各版の描画ルーチンは、まず `draw_page` をアクセスページとして選び、そのページをクリアして描画します。完成後に VSYNC 側がそのページを表示ページへ切り替え、`draw_page` を反転して次の裏ページを作ります。256 色版ではバックエンドごとに VRAM アクセス方法が異なっても、このページ交換という上位の考え方は共通です。

5.2 256 色版のバックエンド

PEGC 256 色版は Plane / Packed / PackedFull など複数の描画経路を選べます。`video_enter` は BIOS と PEGC レジスタを設定し、実際に使う `video_backend` を決定します。その後のシーン・時間管理はバックエンドから独立しており、最終的な VRAM 読み書き方法だけが切り替わります。

SOURCE　NEON3 VIDEO3_256.INC:50-65 以降（backend 定義と video_enter）、326 以降（video_leave）

5.3 パレットは「完成フレームの時刻」に固定する

現在の IRQ 駆動版では、描画中にも `frame_counter` が進む可能性があります。そのため、完成した絵と後から参照した現在時刻を混ぜると色だけが先へ進んでしまいます。`vsync_queue_completed_frame` は完成時の `scene_index` / `scene_frame` を `vsync_present_*` へコピーし、その時刻に対応するパレットを RAM へ事前計算してから `vsync_frame_ready=1` にします。

SOURCE　NEON3 AFS256.INC:110-143 / NEON1 AFS256.INC:68-92 / NEON2 AFS256.INC:68 付近以降

VSYNC IRQ 内では重い補間や除算を避け、準備済みの値を DAC へ書く処理だけにしています。NEON2 のスペクトラムや白化/フェード、NEON3 の都市シーンパレットなど、内容は作品ごとに違っても「計算はメイン側、反映は VSYNC 側」という境界を保っています。

6. VSYNC、論理時間、BGM の進行

6.1 IRQ2 / INT 0Ah が実時間の基準

現行ソースでは PC-98 の master GDC VSYNC 割り込み（IRQ2 / INT 0Ah）を、表示を切り替える共通のスケジューリング境界として使います。/AFS ではこの実 VSYNC をそのまま BGM と映像の時間基準にし、描画所要時間から音楽テンポを切り離します。一方 /FS:n では、VSYNC は公開タイミングとして使いつつ、BGM と映像の論理時刻は「完成フレーム＋明示された固定 skip」のときだけ一緒に進め、処理が重い場合は両方を同じだけ遅らせます。

```
adaptive_vsync_irq:
    ...
    if (fixed_fs_skip_slot)
        vsync_visual_tick_due = 1
    if (vsync_frame_ready) {
        set_display_page(vsync_pending_page)
        commit_prepared_palette()
        draw_page ^= 1
        vsync_frame_ready = 0
        if (/FS:n) vsync_visual_tick_due = 1
    }
    if (/AFS || vsync_visual_tick_due) {
        advance_audio_tick()
        advance_visual_tick()
    }
    adaptive_vsync_count++
```

SOURCE NEON3 AFS256.INC:31-108 / NEON1・NEON2 AFS256.INC:10-66 付近

6.2 VSYNC 割り込みの処理順

6. GDC へ書き込み、次回の VSYNC IRQ を再アームする。
7. /FS:n 時は固定 skip countdown を進め、1 つ消費したら `vsync_visual_tick_due=1` とする。
8. 完成フレームがあれば表示ページを切り替え、予約済みパレットを反映する。
9. `advance_logical_tick` を必ず呼び、BGM とデモの論理時間を進める。
10. `adaptive_vsync_count` を進める。
11. 自分で IRQ2 を有効化した場合は PIC へ EOI を送り IRET、既存利用者がいた場合は旧 INT 0Ah へチェインする。

この順序により、/AFS では完成ページが無い VSYNC でも音楽と映像時刻を進め、次回の描画がその時点の frame_counter に追いつくことで自動フレームスキップになります。/FS:n では完成フレームを公開したとき、または明示された固定 skip を 1 つ消費したときだけ音楽と映像時刻を一緒に進めます。したがって固定モードでは描画遅延による追加の自動 skip や BGM 先行が発生しません。

6.3 /SPEED は論理時刻の倍率

`advance_logical_tick` は `speed_percent` を 100 分率の位相蓄積として扱います。100 なら 1 回の VSYNC で 1 論理 tick、100 未満なら時々据え置き、100 超なら時々 2 回以上 `advance_one_logical_tick` を呼びます。1

論理 tick では `sound_tick` を先に実行し、その後 `frame_counter` をインクリメントし、総フレーム数に達したら 0 へ戻します。

SOURCE NEON: NEONRUN.ASM:735-768 / NEON2: neon2.asm:207-242 / NEON3: NEON3256.ASM:218-249

6.4 /AFS と /FS の意味

モード	描画開始条件	VSYNC 中の BGM/時刻	映像が重い場合
/AFS	前フレームが表示済みならすぐ次を描く	毎 VSYNC、音楽と映像時刻を同時に進める	未描画の論理フレームを飛ばし、BGM テンポは実時間を維持
/FS:n	完成表示後、指定された固定 skip を消費してから次を描く	完成フレーム/固定 skip の論理 slot だけ音楽と映像時刻を一緒に進める	追加の自動 skip はせず、処理が遅ければ BGM も映像と同じだけ遅れる
IRQ 不可時	従来ポーリング経路	`advance_logical_tick` で音楽＋映像を同期更新	AFS は環境に応じて従来測定または固定相当へ縮退

7. 音源検出・初期化・シーケンサ

7.1 sound_detect

音源処理は OPNA 側の共通入口 `sound_detect` から始まります。OPNA/YM2608 または YM2203/26K を先に調べ、必要に応じて OPL3 を調べ、/AUTO・/OPNA・/26K・/OPL・/BOTH などの指定と組み合わせて最終的な `opna_present` / `opl_present` / `sound_present` を決めます。

SOURCE OPNA.INC:14-63

7.2 sound_initialize

`sound_initialize` は存在するチップだけを初期化し、ミキサ設定を反映した後、曲のインデックス・セクション・divider を初期値へ戻します。最後に `music_force_step` を呼ぶため、メインループへ入った時点で先頭ステップのレジスタ状態が作られています。

SOURCE OPNA.INC:168-184

7.3 sound_tick は論理 tick の一部

外側から見た `sound_tick` は `music_tick` への入口です。曲データは `MUSIC_DIVIDER` ごとに 1 ステップ進み、各チップへ必要なレジスタを書き込みます。NEON2/NEON3 では 6144 論理フレームと音楽構成が対応するよう設計されており、シーン境界も音楽側の節と合わせやすい構成です。

SOURCE OPNA.INC:186-210 以降。NEON3 版ではコメント上、1024 score steps × 6 logical frames = 6144 frames。

7.4 sound_stop

終了時は OPL3 の全音停止とミキサ復元、OPNA/26K 側の shutdown を行います。これは画面復元より前に呼ばれ、音源が鳴りっぱなしのまま DOS へ戻ることを防ぎます。

SOURCE OPNA.INC:189-201 / 各メイン ASM の .exit_demo

> 描画と音楽の同期 /AFS は「実 VSYNC が来たら音楽と映像時刻を進める」ため、重い 3D でも BGM テンポを維持し、映像側が追いつくためにフレームを飛ばします。/FS:n は逆に同期優先で、描画が間に合わない余分な VSYNC では音楽も映像時刻も進めません。

8. 入力、STOP/Ctrl+C、終了処理

8.1 非同期割り込みの中では後始末しない

PC-98 の STOP は INT 06h、DOS の Ctrl+C は INT 23h です。`break_guard_install` は元ベクタを保存して両方を共通 `break_guard_handler` へ差し替えます。ハンドラは `break_requested=1` として IRET するだけで、DOS、BIOS、VRAM、音源を一切操作しません。

SOURCE BREAKGUARD.INC:14-49, 88-99

8.2 通常キーも同じ終了経路へ合流

メインループではまず `break_requested` を確認し、その後 DOS AH=06h による直接コンソール入力をポーリングします。キーがあれば `.exit_demo` へ進みます。これにより、STOP/Ctrl+C と通常キーのどちらも必ず同じクリーンアップ順序を通ります。

8.3 終了処理の順序

12. `adaptive_clock_shutdown` で VSYNC IRQ 側のスケジューラ処理を停止し、INT 0Ah/PIC の状態を復元する。
13. 音源が存在すれば `sound_stop` で発音停止・ミキサ/音源状態を復元する。
14. `video_leave` / `video16_leave` でグラフィックモード、ページ、PEGC/GRCG/EGC などを戻す。
15. テキストカーソルを再表示する。
16. `break_guard_restore` で INT 06h / INT 23h の元ベクタを戻す。
17. 終了メッセージ/クレジットを表示し、DOS INT 21h AH=4Ch で終了する。

SOURCE NEONRUN.ASM:235-255 / neon2.asm:183-203 / NEON3256.ASM:194-214; AFS256.INC:184-225; BREAKGUARD.INC:51-86

終了保護の復元を画面・音源の復元より後にしている点が重要です。グラフィックモード中のクリーンアップ途中に再度 STOP/Ctrl+C が入っても、割り込みコンテキストで中途半端な終了処理が走らないようにしています。

9. IRQ が使えない場合のフォールバック

VSYNC IRQ2 は PC-98 環境や常駐ソフトの状態によって既に利用されている可能性があります。そのため、スケジューラは導入しただけで成功扱いにせず、実際にカウンタが進むかを確認します。

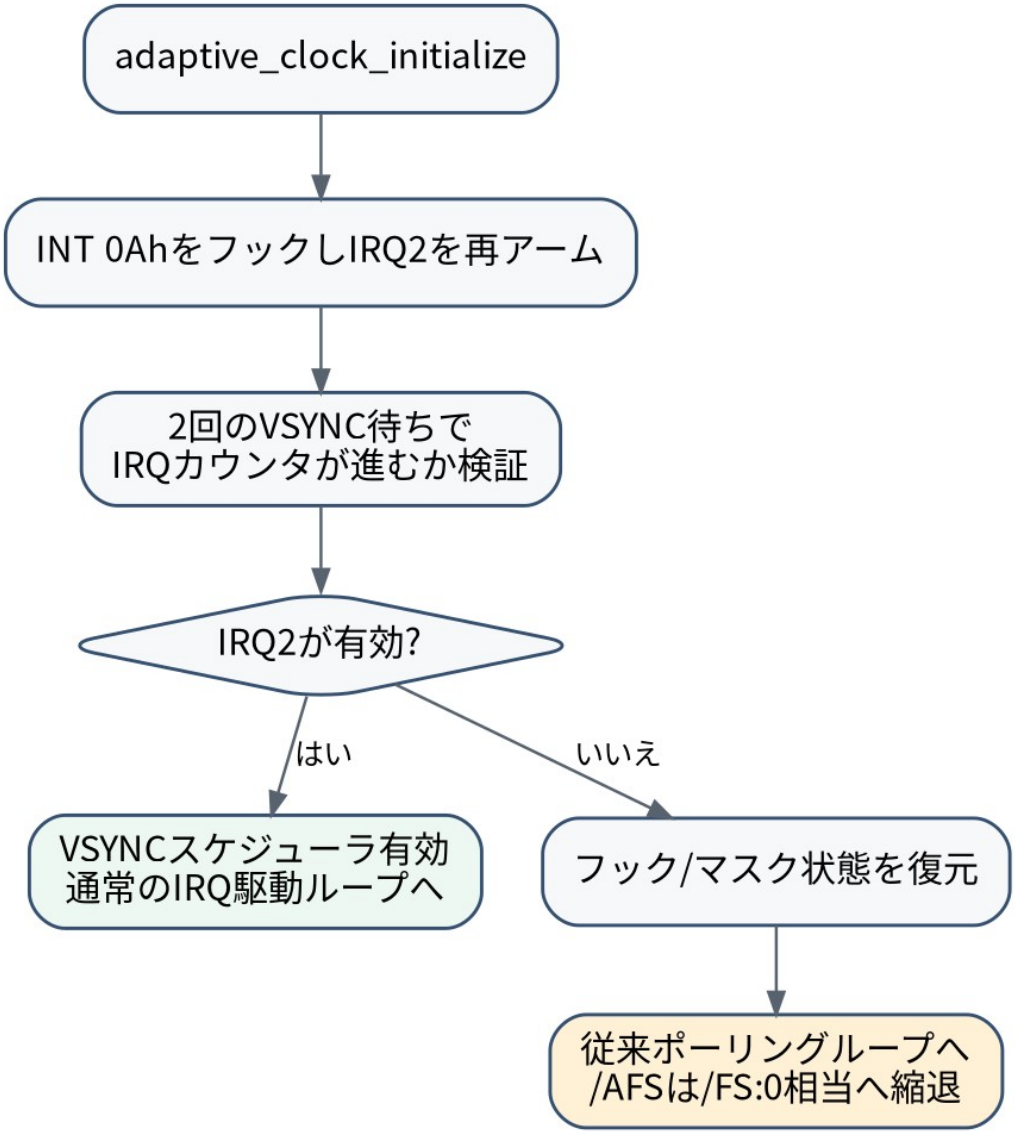

図 9-1　VSYNC IRQ スケジューラ初期化とフォールバック

9.1 install と所有権

`adaptive_vsync_install` は PIC マスク状態を保存し、INT 0Ah の旧ベクタを取得して自前ハンドラを設定します。IRQ2 が元々マスクされていた場合だけ自分でアンマスクし `adaptive_vsync_owned=1` とします。元々利用されていた場合は既存ハンドラを尊重し、終了時にはチェイン/復元の責任を分けます。

```
SOURCE  NEON3 AFS256.INC:145-182
```

9.2 実動作確認

`adaptive_clock_initialize` は割り込みを入れた後、VSYNC カウンタを読み、`wait_vsync` を 2 回行って再度カウンタを読みます。差が 0 なら IRQ が実際には動いていないとみなし、`adaptive_clock_shutdown` で元へ戻して legacy 経路を使用します。検証中はスケジューラ本体をまだ running にしないため、起動検査の 2 VSYNC が BGM 時間として消費されないようになっています。

SOURCE NEON3 AFS256.INC:235-290（他世代も同じ骨格）

9.3 legacy loop

legacy 経路では、メイン側が `wait_vsync` を呼び、ページフリップ、パレット反映、`advance_logical_tick` を同期的に実行します。/AFS 時は旧来の経過 VSYNC 測定ヘルパを使うか、IRQ 無効判定時には /FS:0 相当へ縮退します。つまり IRQ スケジューラが利用できなくてもデモ自体を停止させず、従来方式で続行する設計です。

SOURCE NEON3256.ASM:140-183 / NEONRUN.ASM:172-223 / neon2.asm:138-170

10. NEON / NEON2 / NEON3 と各版の差分

共通フローは強く維持されていますが、世代が進むにつれて「シーン選択・描画内容・ビデオバックエンド」が拡張されています。下表は、実行フローを読むうえで重要な差だけをまとめたものです。

項目	初代 NEON	NEON2	NEON3
全体時間	3072 frames / 5 scenes	6144 frames / 12 scenes	6144 frames / 9 continuous scenes
シーン選択	比較チェーン	長さテーブル	長さテーブル + 一部境界状態管理
描画の性格	2D/3D + text、比較的直接的	多数の演出レイヤ・パターン・粒子	連続 3D 都市、持続カメラ、ロトズーム
256 色	別 NEON256、PEGC 系	Plane/Packed 系 + CPU 機能活用	Plane/Packed/PackedFull、都市パレット
16 色/標準	NEONRUN は 8 色、286 版あり	16 色 GRCG/EGC + 286 版	16 色版 + 286 版（200/400）
VSYNC/BGM	現行版は IRQ 駆動へ統一	現行版は IRQ 駆動へ統一	現行版は IRQ 駆動へ統一

10.1 版が違っても変わらない骨格

- `start` で環境・オプション・検出を確定する。
- 画面、文字、音源、VSYNC 時間管理を初期化する。
- `frame_counter` をシーン時刻へ分解し、裏ページへ描画する。
- 完成フレームを VSYNC 側へ渡し、表示とパレットを同期する。
- 音楽と映像時刻を、/AFS または/FS のモード規則に従って同期して進める。
- キー/STOP/Ctrl+C をメイン側で検出して一つの終了経路へ入る。
- IRQ→音源→画面→文字→break guard の順に安全に復元して DOS へ戻る。

10.2 読むときは「版固有の描画」と「共通制御」を分ける

ソース量の大半はシーン描画・VRAM アクセス・音楽データですが、プログラムの制御フローを理解するためには、まずメイン ASM、AFS*.INC、SCENE*.INC、FRAME_RENDER*.INC、VIDEO*.INC、OPNA.INC、BREAKGUARD.INC の順に役割を分けて見ると整理しやすくなります。特に 256 色の Plane/Packed 差や 286 用投影コードは「フレームをどう描くか」の差であり、「いつ描いていつ表示し、いつ時間を進め、どう終了するか」という外枠は共通です。

11. BGM データ形式

NEON シリーズの実行時 BGM は、S98 や MIDI ファイルを読み込む方式ではありません。COM 内にアセンブルされた短い 1 バイトイベント列を、`sound_tick` / OPNA / OPL3 側のシーケンサが一定の論理間隔で読み、音源レジスタへ変換して演奏します。音色、音量、セクションごとの強弱は別のテーブルや初期化コードで与えるため、譜面データそのものは非常にコンパクトです。

実行時データと配布用データの違い NEON3 に同梱した S98 は OPNA+OPL3 のレジスタログ、MIDI は鑑賞・DAW 用の編曲データです。どちらも実行時プレイヤーが読む入力ファイルではなく、COM が読むのは以下の内部スコア表です。

11.1 1 バイトの音符トークン

3 作品で基本トークンは共通です。`REST=FFh` は休符/キーオフ、`HOLD=FEh` は直前の発音を保持し、通常の音符は `NOTE(block, semi) = (block << 4) | semi` で 1 バイト化します。上位 4bit が音域（block）、下位 4bit が半音番号です。たとえば `54h` は block=5、semi=4 を表し、実際の F-number や OPL block/F-number への変換は音源ドライバ側で行います。

図 11-1　内部 BGM スコアの格納イメージ

```
music_data:
        db 54h,0FEh,0FFh,0FFh,5Bh,0FEh,...
        ; 1チャンネル分の全 step を並べ、その後に次チャンネルが続く
```

`music_data` は基本的に channel-major です。つまり「step 0 の全チャンネル」を交互に並べるのではなく、チャンネル 0 の全 step、チャンネル 1 の全 step……という順で格納し、読み出し時に `channel * MUSIC_STEPS + music_score_index` の形で現在位置を求めます。

SOURCE　共通トークン: NEON CONFIG256.INC:14-17,32-34 / NEON2 CONFIG256.INC:14-20,35-37 / NEON3 CONFIG3_256.INC:18-28,47-49

11.2 作品ごとのスコア長とチャンネル構成

作品 / 曲名	内部スコア	主なチャンネル構成	論理時間
NEON / Pulse Cartography	512 step × 12ch（32 bars）	OPNA: FM 0..5 + SSG 6..8 / OPL3 側も共通スコアを利用	6 logical frames / step = 3072 frames
NEON2 / Signal Convergence	1024 step × 12ch（64 bars）	OPNA: 0..5 FM + 6..8 SSG / OPL3: 0..9、10..11 予約	6 logical frames / step = 6144 frames

作品 / 曲名	内部スコア	主なチャンネル構成	論理時間
NEON3 / METROPOLITAN IGNITION	1024 step × 9ch（64 bars）+ rhythm 1024 bytes	9 tonal roles を OPNA/OPL3 で共有 + 独立 rhythm mask	6 logical frames / step = 6144 frames

初代 NEON では `Pulse Cartography` が 12ch×512step、NEON2 では 64 小節/1024step へ拡張され、OPL3 専用の第 10 音声を追加できる余地を持ちます。NEON3 では 9 つの共通 tonal role へ整理し、リズムだけを別配列に分けました。

SOURCE NEON DATA.INC:485-487 / NEON2 MUSIC_R11_OPNA_OPL3.INC:1-25 / NEON3 MUSIC_URBAN_D8.INC:1-8,603-604

11.3 6 フレームごとのシーケンサと /AFS・/FS

標準速度では `MUSIC_DIVIDER=6` なので、6 論理フレームごとに `music_score_index` が 1 進みます。NEON2/NEON3 の 1024step なら 1024×6=6144 論理フレームで 1 周です。重要なのは「6 実 VSYNC」ではなく「6 論理フレーム」だという点です。/AFS では論理フレーム自体が実 VSYNC へ追従するため BGM テンポを維持し、/FS:n では映像の固定論理進行に音楽も合わせるため、描画が遅ければ BGM も同じ比率で遅くなります。

`/SPEED` は音楽用 `audio_speed_phase` と映像用 `speed_phase` の双方へ同じ倍率を掛けます。このため、AFS/FS のどちらでも「BGM だけが別の倍率で進む」ことを避けています。

SOURCE NEON3 NEON3256.ASM: advance_audio_tick / advance_visual_tick / AFS256.INC: adaptive_vsync_irq .tick_time

11.4 NEON3 の独立リズムマスク

NEON3 では旋律 9ch とは別に `rhythm_steps` を 1024byte 持ち、各 step を 1byte の bit mask として扱います。bit は BD=01h、SD=02h、CYM=04h、HH=08h、TOM=10h、RIM=20h です。同じマスクを OPNA Rhythm 側と OPL3 percussion 側が読み、それぞれの音源に適したレジスタ書き込みへ変換します。これにより、旋律スコアを増やさずに打楽器の同時打点を表現できます。

SOURCE NEON3 MUSIC_URBAN_D8.INC:603-604 / OPNA.INC:913-914 / OPL3.INC:1208-1209

12. グラフィックデータ形式

NEON シリーズには、一般的なゲームのような「画面 1 枚を格納したビットマップ」や「スプライト画像集」を中心とする共通グラフィックファイル形式はありません。画面の大部分は、少数の座標・辺・配置パラメータ・lookup table を入力として、そのフレームの形状を CPU が生成して VRAM へ描く手続き型です。したがって、ここでいう「グラフィックデータ形式」は、描画ルーチンへ渡すコンパクトな ASM 内テーブル群を指します。

固定画像ではなく生成規則 同じモデル/シーン情報から 16 色、256 色、286 版の異なる VRAM 方式へ出力できます。作品の見た目を規定する中心は「ピクセル列」より「幾何データ＋時間関数＋描画バックエンド」です。

12.1 シーン時刻のテーブル

NEON2 と NEON3 では、各シーンの長さを `dw` のフレーム数配列、処理先を `dw` の near pointer 配列として持ちます。`select_scene` は `frame_counter` から長さを順に引いて `scene_index` と `scene_frame` を求め、同じ index で `scene_routines` を参照します。これは画像データではありませんが、どの生成規則をいつ使うかを決める最上位のグラフィックメタデータです。

```
scene_length_table:
        dw 640,640,512,896,...
scene_routines:
        dw scene_city_descent, scene_tower_canyon, ...
```

SOURCE NEON2 SCENE256.INC / NEON3 SCENE3_256.INC

12.2 頂点・面・辺 - 初代 NEON の八面体

初代 NEON の八面体は、6 頂点を「signed word の x,y,z 3 要素」、8 面を「3 つの頂点 index + 面色の 1byte 属性」、12 辺を「始点 index, 終点 index」の byte pair として保持します。毎フレーム回転して `oct_projected` へ投影座標を作り、面は back-face 判定後に三角形塗り、辺は線分として描きます。モデルそのものは数十 byte 規模で、姿勢やスクリーン座標はデータに保存せず毎フレーム計算します。

データ	格納単位	意味	例
頂点	dw x, y, z	3D モデルのローカル座標	oct_vertices
面	db v0, v1, v2, color	三角面の頂点参照＋面色	oct_faces
辺	db v0, v1	wireframe の接続関係	oct_edges / city_box_edges
投影 scratch	dw screen_x, screen_y, ...	そのフレームだけ使う計算結果	oct_projected

SOURCE NEON DATA.INC:105-124 / VIDEO.INC: draw_octahedron

12.3 NEON3 の都市は「箱の 8 頂点」ではなく箱記述から生成

NEON3 のビル等は、個々の 8 頂点を静的に並べるのではなく、`city_obj_x/y/z` と `w/h/d` をセットした box descriptor から 8 頂点をその場で組み立てます。`city_obj_skew_q8` で道路方向へせん断でき、flags bit0 で中間水平 band、bit1 で屋上 antenna を追加します。辺の接続だけは共通の `city_box_edges` 12 組を再利用します。大量の都市要素を「同じ形状トポロジ + 異なる寸法/位置」で表現するための方式です。

フィールド/表	型	役割
city_obj_x / y / z	dw	箱のワールド座標
city_obj_w / h / d	dw	幅・高さ・奥行き

フィールド/表	型	役割
city_obj_skew_q8	dw (Q8)	Z 方向に応じた X せん断
city_obj_flags	db bit flags	中間 band / roof antenna
city_box_edges	12 × (db v0,v1)	全 box 共通の辺接続
height/jitter/park/rail 配列	小さな dw table	建物変化、木、線路等の安定配置

SOURCE NEON3 DATA3.INC:906-927,1005-1027 / CITY3D256.INC:927-1057

12.4 データから VRAM までの変換パイプライン

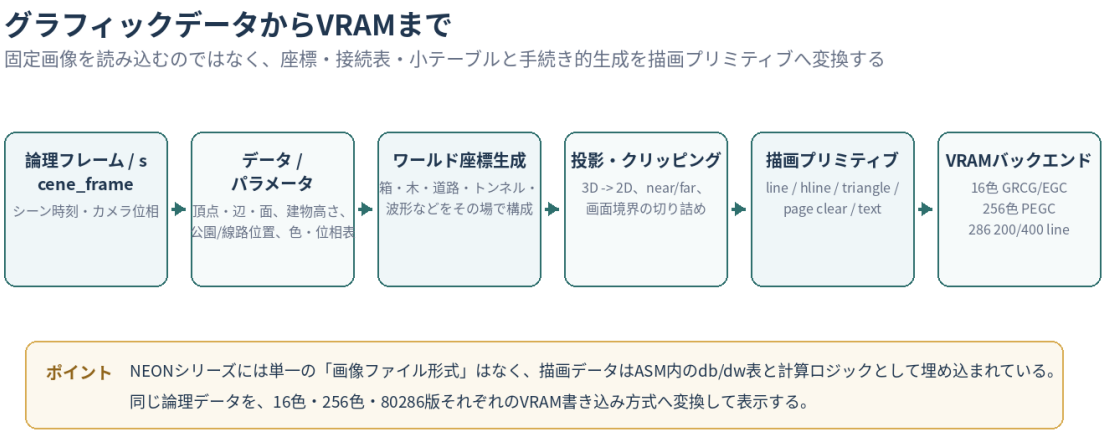

図 12-1　幾何データから各版の VRAM へ到達するまで

3D 要素は「論理フレーム → scene/camera parameter → world 座標 → projection/clip → line/triangle 等の 2D primitive → VRAM」という段階を通ります。16 色版は GRCG/EGC 系の plane 操作、256 色版は PEGC Plane/Packed/PackedFull、286 版は 16 色 GRCG 系へ落とします。上位の幾何データと下位のラスタライズを分けているため、同じシーン構成を複数のハードウェア条件へ移植できます。

286 版 NEON3 では論理座標系を 640×400 に保ち、標準の 640×200 モードだけ最終 primitive 側で Y を 1/2 へ変換します。/400 では同じ論理データをそのまま 400line へ出すため、シーン側に 200line 専用データを二重に持つ必要がありません。

SOURCE NEON3 CONFIG3_286.INC: 200/400 line policy / CITY3D286_CORE.INC: logical 640x400 comment / VIDEO3_286.INC

12.5 lookup table と作業領域も「描画データ」の一部

`sin_table`、perspective reciprocal、tunnel scale などはモデルそのものではなく、フレーム中の乗除算を減らすための lookup table です。また `tunnel_current`、`oct_projected`、`city_box_projected` のような配列は毎フレーム上書きされる scratch buffer で、永続的なシーン資産ではありません。ソースを読むときは「固定モデル/配置」「時間で変わる parameter」「計算結果の scratch」を区別すると、何が作品データで何がレンダラ内部状態か分かりやすくなります。

付録 A. 主要ソースファイルとラベル一覧

役割	初代 NEON	NEON2	NEON3
プログラム入口/主ループ	NEONRUN.ASM / NEON256.ASM / NEON286.ASM	neon2_16.asm / neon2.asm / NEON2286.ASM	NEON3_16.ASM / NEON3256.ASM / NEON3286.ASM
コマンドライン	本体内 or COMMAND256.INC	COMMAND16/256/286.INC	COMMAND3_16/COMMAND256/ COMMAND3_286.INC
VSYNC/AFS	AFS*.INC	AFS16/256/286.INC	AFS3_16/AFS256/AFS3_286.INC
シーン選択	本体内 / SCENE256.INC	SCENE16/SCENE256.INC	SCENE3_256.INC + 3D scene source
フレーム生成	本体内 / FRAME_RENDER256.INC	FRAME_RENDER256.INC / 16 色本体	FRAME_RENDER3_256/16/286.INC
映像	VIDEO*.INC	VIDEO16_* / VIDEO256*.INC	VIDEO3_16/256/286.INC + Packed 系
文字	TEXT*.INC	TEXT16/256/286.INC	TEXT3_16/TEXT256/ TEXT3_286.INC
音源	OPNA.INC + OPL3.INC	OPNA.INC + OPL3.INC	OPNA.INC + OPL3.INC
安全終了	BREAKGUARD.INC	BREAKGUARD.INC	BREAKGUARD.INC
BGM データ	DATA.INC: music_data	MUSIC_R11_OPNA_OPL3.INC	MUSIC_URBAN_D8.INC + rhythm_steps
グラフィックデータ	DATA.INC: oct_vertices/faces/edges	SCENE*.INC + DATA*.INC の演出/lookup 表	SCENE3*.INC + DATA3.INC + CITY3D*.INC の box/配置表

代表ラベルを追う場合は、次の順で検索すると全体制御がつながります。

```
start
  -> parse_command_line
  -> sound_detect / video_enter / text_initialize / sound_initialize
  -> adaptive_clock_initialize
  -> .main_loop
       -> snapshot render_frame_counter
       -> select_scene / render_scene / text_update
       -> vsync_queue_completed_frame
  <-> adaptive_vsync_irq
       -> present completed page
       -> /AFS: audio+visual every VSYNC
          /FS : audio+visual only fixed logical slots
  -> .exit_demo
       -> adaptive_clock_shutdown -> sound_stop -> video_leave -> break_guard_restore
```

照した代表的なソス範

SOURCE 初代 NEON: NEONRUN.ASM:58-255, 735-821 / AFS256.INC:10-290 / BREAKGUARD.INC:14-99

SOURCE NEON2: neon2.asm:33-203, 207-242 / SCENE256.INC:7-54 / FRAME_RENDER256.INC:1-16 / AFS256.INC

SOURCE NEON3: NEON3256.ASM:28-214, 218-249 / SCENE3_256.INC:5-62 / FRAME_RENDER3_256.INC:1-54 / AFS256.INC:31-290

付録 B. 読み進めるときのおすすめ順序

ソースを実際に追跡する場合は、個々の描画ルーチンから入るより、以下の順序で「外側から内側へ」読むと迷いにくくなります。

1. メイン ASM の `start` から `.exit_demo` までを通読し、初期化順とループ骨格を把握する。
2. AFS*.INC の `adaptive_vsync_irq` と `vsync_queue_completed_frame` を読み、描画側と実時間側の境界を理解する。
3. `advance_logical_tick` / `advance_one_logical_tick` と OPNA.INC の `sound_tick` をつなげ、音楽と frame_counter の関係を見る。
4. SCENE*.INC の `select_scene` と `scene_routines` を確認し、全体フレームがシーン固有処理へ届く経路を見る。
5. FRAME_RENDER*.INC を見て、ページクリア、scene 描画、追加レイヤ、text_update の順を確認する。
6. 最後に VIDEO*.INC や各シーン実装へ入り、VRAM 方式・3D・パレットなど版固有の詳細を見る。
7. BGM を追う場合は CONFIG*.INC の `MUSIC_STEPS` / `MUSIC_DIVIDER` と `music_data` を確認し、OPNA.INC / OPL3.INC の `music_score_index` 計算へつなげる。
8. グラフィックデータを追う場合は SCENE*.INC → DATA*.INC の固定表 → 投影/clip → VIDEO*.INC の primitive 描画という順に見て、固定データと毎フレーム scratch を区別する。

まとめ NEON シリーズは「論理時刻からシーンと内部 BGM スコアを選び、幾何データ/パラメータから裏ページへ画面を生成し、VSYNC で完成結果だけを公開する」という構造で読むと全体が整理できます。BGM は 1byte token 列、グラフィックは座標・辺・配置表を中心とした手続き型で、どちらも小さなデータから時間変化を生成する設計です。