ユーザ·マニュアル

RF/NFC 抽象化レイヤ（RFAL）

概要

RFAL：Radio Frequency Abstraction Layer（RF 抽象化レイヤ）は、RF/NFC 通信とこれに関連するその他の一般的動作を実行するために必要な複数の機能を提供するライブラリです。さまざまな RF IC（既存および今後の ST25R デバイス）に対する、共通の使いやすいインタフェースが含まれます。

この共通インタフェースにより、基盤となる ST25R デバイスに依存しない、上位ソフトウェア·レイヤを構築できます。すべての ST25R デバイスに対して同じ API による RFAL を使用できるため、試作段階や ST25R デバイス間の移行に際して、ソフトウェア関連の工数が軽減されます。

1 略語

表 1. 略語

略語	内容
AC	Analog Configuration（アナログ設定）
ACM	Active Communication Mode（アクティブ通信モード）
AFE	Analog Front End（アナログ・フロントエンド）
AP2P	Active P2P（アクティブ P2P）
APDU	Application Protocol Data Unit（アプリケーション・プロトコル・データ・ユニット）
API	Application Programming Interface（アプリケーション・プログラミング・インタフェース）
CE	Card Emulation（カード・エミュレーション）
CPU	Central Processing Unit（中央演算処理装置）
DEP	Data Exchange Protocol（データ交換プロトコル）
DPO	Dynamic Power Output（ダイナミック出力調整）
DS	DataSheet（データシート）
FAQ	Frequently Asked Question（よくある質問）
FDT	Frame Delay Time（フレーム遅延時間）
FWT	Frame Wait Time（フレーム待機時間）
FW	FirmWare（ファームウェア）
FIFO	First In First Out（先入れ先出し）
GPIO	General Purpose Input/Output（汎用入出力）
GT	Guard Time（ガード時間）
HAL	Hardware Abstraction Layer（ハードウェア抽象化レイヤ）
HW	HardWare（ハードウェア）
IC	Integrated Circuit（集積回路）
MCU	MicroController Unit（マイクロコントローラ）
MPU	MicroProcessor Unit（マイクロプロセッサ・ユニット）
NFCID	NFC IDentifier（NFC 識別子）
P2P	Peer To Peer（ピア・ツー・ピア）
PCD	Proximity Coupling Device（近接型結合装置）
PCM	Passive Communication Mode（パッシブ通信モード）
PICC	Proximity Inductive Coupling Card（近接誘導型結合カード）
RAM	Random Access Memory（ランダム・アクセス・メモリ）
RFAL	RF Abstraction Layer（RF 抽象化レイヤ）
RF HAL	RFAL Hardware Abstraction Layer（RFAL ハードウェア抽象化レイヤ）
RF HL	RFAL Higher Layer（RFAL 上位レイヤ）
RW	Reader/Writer（リーダライタ）
SW	SoftWare（ソフトウェア）
TC	Test Case（テスト・ケース）
TS	Test Suite（テスト・スイート）
UID	Unique IDentifier（固有識別子）

略語	内容
WTX	Waiting Time Extensions（待機時間延長）

2 RFAL ライブラリ

2.1 機能の概要

- 高性能 NFC リーダ ST25R を用いた NFC 対応アプリケーションを構築するための総合的なミドルウェア
- 主要な NFC テクノロジーとプロトコルのすべてに対応
- すべての ST25R デバイスに対応
- 複数のプラットフォーム（マイコン/RTOS/OS）およびアーキテクチャ（8 ビット～ 32 ビット）間の移植が容易
- RF/NFC 規格準拠（NFC フォーラム、EMVCo®、ISO4443、ISO15693、ISO18092）
- MISRA C:2012 準拠
- 適時更新
- 複数のサンプル・アプリケーションを提供中（製品ページのソフトウェア配付物を参照してください）
- 自由かつユーザ・フレンドリなライセンス条項

表 2. 該当製品

種類	品名
ST25R3911	ST25R3911B
	ST25R3912
	ST25R3914
	ST25R3915
ST25R3916	ST25R3916
	ST25R3917
	ST25R3920
ST25R95	ST25R95

表 3. サポート対象モード（x = サポート）

		ST25R3911B	ST25R3916	ST25R95
RW	NFC-A	x	x	x
	NFC-B	x	x	x
	NFC-F	x	x	x
	NFC-V	x	x	x
CE	NFC-A	-	x	-
	NFC-F	-	x	-
P2P		x	x	-
AP2P	イニシエータ	x	x	-
	ターゲット	x	x	-

2.2 説明

RFAL は、RF/NFC 通信や NFC デバイスを用いたその他の一般的な動作の実行に必要な複数の機能を提供します。さまざまな RF IC（既存および今後の ST25R デバイス）を、以下に対応する共通インタフェースの下にカプセル化しました。

- NFC モード:
  - リーダライタ（ポーラ）
  - P2P イニシエータ（PCM および ACM）
  - P2P ターゲット（PCM および ACM）
  - カード・エミュレーション（リスナ）

- テクノロジー:
  - NFC-A(ISO14443-A)
  - NFC-B(ISO14443-B)
  - NFC-F(FeliCa™)
  - NFC-V(ISO15693)
  - P2P(ISO18092)
  - 独自テクノロジー(例:ST25TB、CTS、Kovio™、B'、iClass®、Calypso®)
- プロトコル:
  - ISO-DEP(ISO Data Exchange Protocol(ISO データ交換プロトコル) - ISO14443-4)
  - NFC-DEP(NFC Data Exchange Protocol(ISO データ交換プロトコル) - ISO18092)
- 動作モード:
  - NFC デバイス・モード
  - 低電力モード
  - ウェイクアップ・モード

2.3 コーディングルールと規則

2.3.1 コーディング規則

ここでは、ライブラリで使用されているコーディングの規則について説明します。

- すべてのコードは、C99 規格に準拠し、C99 対応の少なくとも主要なコンパイラであれば警告を発生することなくコンパイルできます。解消できない警告は、コードに付随する文書やコメントの可能性があります。
- ライブラリはブロッキング・インタフェースと非ブロッキング・インタフェースの両方を提供します。これらは各種プロジェクトのニーズに合わせて、別々に使用することも、組み合わせて使用することもできます。
- 非ブロッキング API は最大 5 ms 継続するように設計されています。
- 非ブロッキング API は `rfalStartOperation()` と `rfalGetOperationStatus()` のペアという形で提供されます。
- ライブラリは ANSI 規格のデータ型を使用するように設計されており、それらの型は ANSI C ヘッダ・ファイル `<stdint.h>` 内で定義されています。必要に応じて、ユーザ定義の型を適用することも可能です。
- ライブラリ内では動的メモリ割り当てを使用していません。メモリを静的に割り当てるコンテキストが必要なモジュール、および次元数が極めて大きいすべてのバッファは、ユーザが提供する必要があります。例外として、オプションで提供される、利便性向上を目的とした機能 / モジュールの一部に、いくつかの大きなバッファを静的に割り当てるものがあります(例: `ST25R_COM_SINGLETXRX`、NFC モジュール)。

2.3.2 ランタイム・チェック

ライブラリは、ライブラリ関数の入力値をチェックすることで実行時の障害を検出する機能を実装します。無効なパラメータまたは矛盾する場面 / 状態で関数を呼び出すと、メソッドは対応するエラーを返します。もう一つのランタイム・チェックは `platformAssert()` マクロによって実行されます。

さらに、回復不能なエラーの発生に備え、ライブラリは `platformErrorHandle()` マクロによるエラー・ハンドラ / トラップを提供します。

2.3.3 MISRA-C:2012 準拠

車載用アプリケーションに対する適合性を高めるために、RFAL ライブラリは MISRA C:2012 規格に完全に準拠しています。

MISRA C とは C 言語によるプログラミングに対する一連のガイドラインで、車載ソフトウェアの信頼性に関する業界団体 MISRA:Motor Industry Software Reliability Association が開発、公開しています。これらのガイドラインの狙いは、曖昧さや、一般的なプログラミング・ミスの犯しやすさの観点から、C 言語において避けるべきことを指摘し、コードの安全性、セキュリティ、移植性、信頼性を向上させることです。MISRA 規格は、自動車業界の組込みソフトウェアに広く使用されています。

詳細は、RFAL の各リリース・パッケージに含まれるドキュメント・フォルダにコンプライアンス・レポートが添付されているので、そちらをご覧ください。

2.3.4 NFC 用語

NFC(NFC フォーラム)は、従来から存在していた RF 規格を集約したものです。

NFC フォーラムは、これらの規格を組み合わせた定義を、一連の簡潔な仕様文書に独自の用語を用いて提供しているため、RFAL 内では、NFC フォーラム仕様 [1]、[2]、[3] に合わせた用語を使用しています。RFAL では、完全性を保つために、適切と思われる場合は、他の規格を参照する場合もあります。

2.4 ハードウェア

2.4.1 要件

使用する ST25R に応じた、RFAL を使用できるようにするための最上位のハードウェア要件は次のとおりです。

- ST25R NFC リーダを駆動できるシリアル·インタフェース·バス（SPI、I^2C、UART）（データシート参照）
- シリアル·バスおよび IRQ バス選択やリセット·ピンなど他の制御ラインの両方の機能を有する GPIO
- 外部割込みトリガ·ソース（推奨手法、必須ではありません）
- RFAL ライブラリを格納するための十分な RAM および Flash メモリ容量を備えたマイコン / MPU/ システム
- ミリ秒のティック·カウンタによる時間計測機能（ハードウェア·タイマなど）

2.4.2 サポート対象のホスト·デバイス

RFAL は C コードで提供されるため、市販されている大部分のコントローラで使用できます。C コンパイラが提供され、上記のハードウェア要件を満たすデバイスであれば、いずれもこのライブラリおよび ST25R デバイスをサポートできる可能性があります。

STM32 マイクロコントローラは、各種アプリケーション向けの幅広いコントローラと機能に関して、RFAL が求める要件をすべて満たしています。

STM8 マイクロコントローラなど、異なるアーキテクチャによる、さほど高度な機能を備えていないデバイスも使用可能です。組込みオペレーティング·システムが必要な場合は、STM32MPU マイクロプロセッサを使用すれば、RFAL により、あらゆる NFC アプリケーションを実現できます。

2.5 ソフトウェア

2.5.1 依存関係

RFAL ライブラリは、通常の C コンパイラ·ツールチェーンで使用可能な一般的なソフトウェア抽象化と、ホスト·デバイスの HAL としてしばしば提供されるデバイス·ドライバに基づいています。

- C ツールチェーン
  - 型定義のための `stdint.h`、`stdbool.h`（ユーザ定義型も使用できます）
  - パラメータ·チェックのための `limits.h`
  - 数学演算のための `math.h`
- ユーティリティ
  - 複数の便利なマクロを提供し、Min、Max、memcpy などの複数の一般的なユーティリティへのリダイレクトを可能とするグルー·レイヤ
- `st_errno`
  - 特定のエラー / イベントの処理や報告に RFAL が使用する、事前定義済みのエラーとマクロを提供します。
- HAL
  - シリアル·インタフェース（SPI、I^2C、UART）
    シリアル·インタフェースを介したバイトの送受信に使用します。
  - GPIO
    IRQ（外部割込みソース）、チップ選択、リセット·ピンなどの固有機能の制御に使用します。
  - タイマ
    指定されたタイミング要件を満たすために必要な特定の内部メカニズムの制御に使用します。

上記のすべての依存関係は、`platform.h` という名前の一元的な場所 / ファイル内に記述する必要があります。RFAL のすべてのモジュールは、この設定ファイルを使用して、ユーザがインクルード文、機能スイッチ、マクロ定義という形で、プラットフォーム固有の設定をすべて定義 / 再マッピングできるようにします。詳細は、セクション 4.1 ペリフェラルの初期化と設定 を参照してください。

2.5.2 サポート対象の開発ツール

RFAL ライブラリはツールチェーンの標準的な機能しか必要としません。主要ツール·プロバイダのすべてに対応しています。サプライヤは、アプリケーション開発の着手から完了までの管理機能を 1 つの統合開発環境（IDE）で提供する、幅広い開発ソリューションを提供しています。

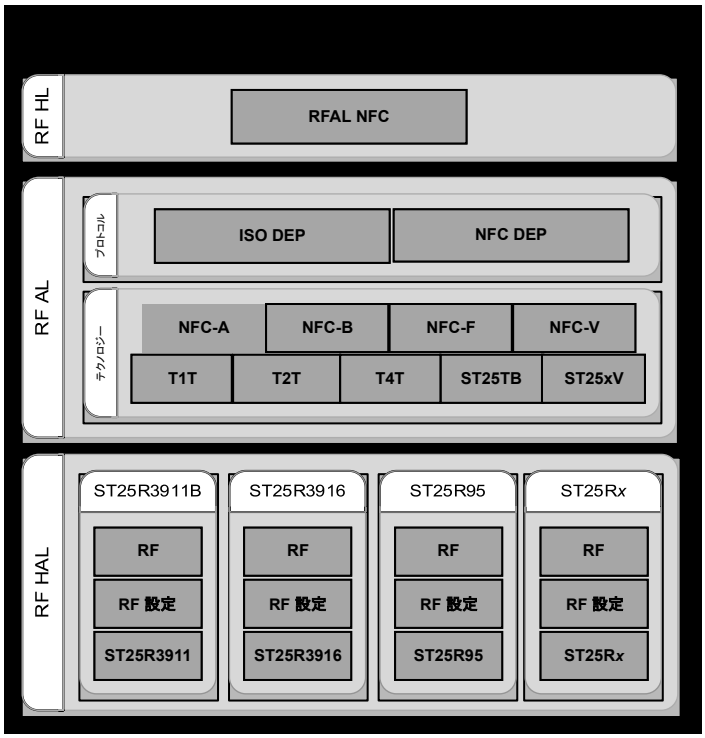
以下に、一般的に使用されている環境や、通常サンプル·プロジェクトが提供されている場所のリストを示します。

- STM32CubeIDE
  - コンパイラ：GCC の C/C++
- ST Visual Develop
  - コンパイラ：Cosmic 社の C コンパイラ
- Keil® uVision
  - コンパイラ： Arm C/C++
- IAR™ Embedded Workbench
  - コンパイラ： IAR C/C++

2.6 アーキテクチャ

RFAL ライブラリは、さまざまなアプリケーションのニーズに応えるための、各種のモジュールと抽象化から構成されたマルチレイヤ·スタックです。

図 1. RFAL スタックのアーキテクチャ



注 ST25Rx は、ST25R ファミリに将来追加される可能性があるデバイスを表しています。

RFAL は内部で次の 3 つのサブレイヤに分割されます。

- RF HL
- RF AL
- RF HAL

2.6.1 RF HAL

このレイヤは、NFC IC ごとの下位ドライバを提供します。

このレイヤにカプセル化されているすべてのモジュールはチップ固有であり、ST25R のデバイスごとに提供されます。次のモジュールがあります。

ST25R

このモジュールには、レジスタへのアクセスとコマンド実行によってハードウェアを直接制御する、NFC IC の下位サポート機能が含まれます。そうした機能がインタフェースとなって、いくつかのハードウェア動作の実行を制御します。

注 これらの API への直接アクセスは推奨できません。API がチップ固有であるため、通常のドライバのフローを通らない直接アクセスを行うと、アプリケーションの移植性が損なわれるからです。通常、アプリケーションでは、この RF モジュールを最下位のインタフェースとして使用します。

RF 設定

アナログ設定とも呼ばれる、このモジュールには、実行時の特定の場面で RFAL によって提供されるアナログ設定のテーブル / リストが含まれます。

テーブルには、ドライバ実行のさまざまな場面で、特定の設定（ビット）を読み込めるようにする、一連の RMV（レジスタ、マスク、値）が記述されています。これによって、デバイスの現在の動作モードに応じて、必要な IC 設定への変更が可能になります。

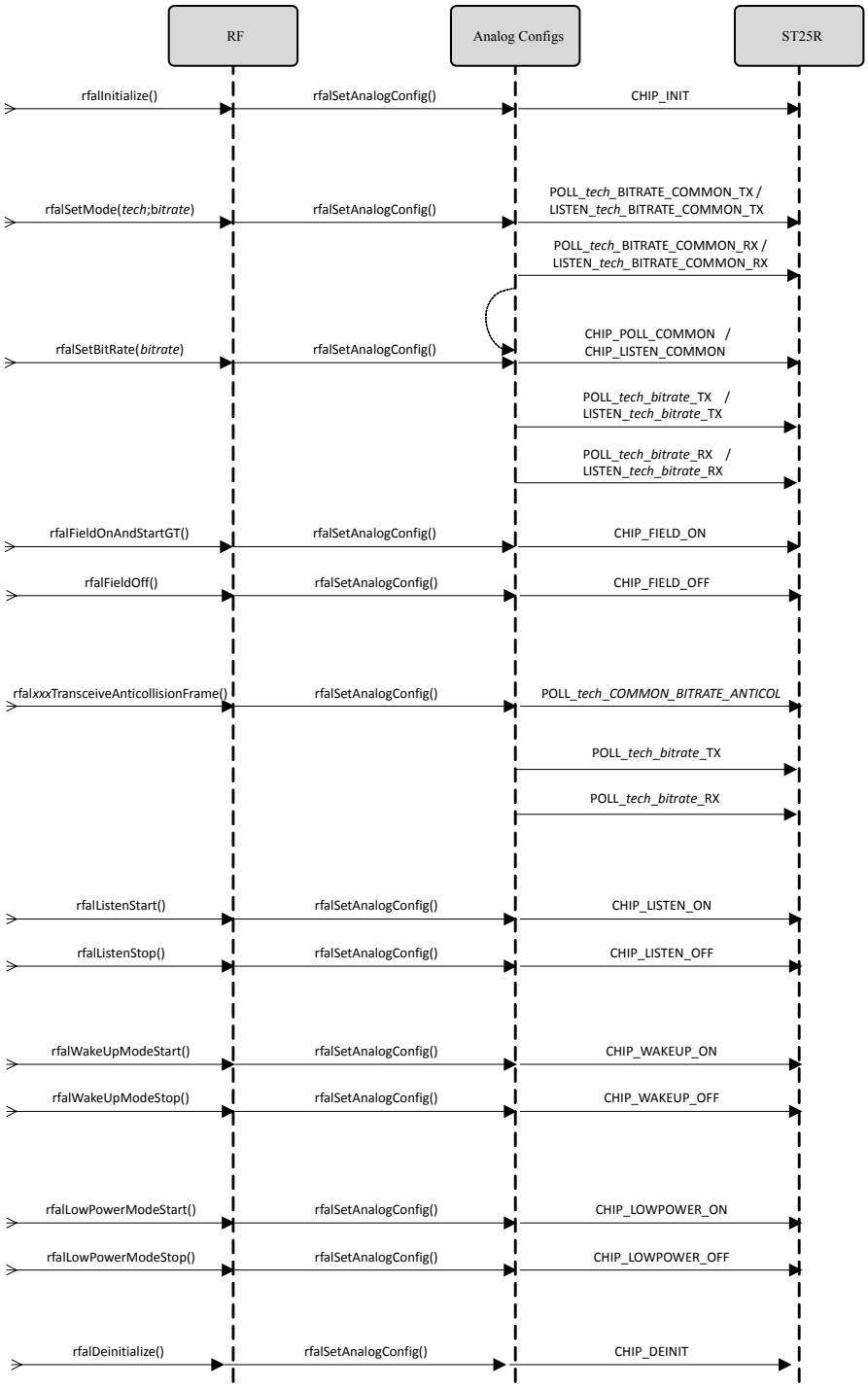
RFAL ドライバの実行中、特定のタイミングで個別に定義された一連の RMV が読み込まれます。各 RMV は 16 ビットの識別子により、内部でエンコードされています。さらに、この識別子は次のような固有イベントもエンコードします。

- `Chip Init`:IC の起動 / リセット / 初期化時に読み込まれます。
- `Chip Deinit`:IC の初期化解除時に読み込まれます。
- `Field On`:フィールド・オンへの遷移前に読み込まれます。
- `Field Off`:フィールド・オフへの遷移後に読み込まれます。
- `Wake-up On`:ウェイクアップ・モードへの移行前に読み込まれます。
- `Wake-up Off`:ウェイクアップ・モードの終了後に読み込まれます。
- `Listen On`:リッスン・モードへの移行前に読み込まれます。
- `Listen Off`:リッスン・モードの終了後に読み込まれます。
- `Low power mode On`:低電力モードへの移行前に読み込まれます。
- `Low power mode Off`:低電力モードの終了後に読み込まれます。
- `Poll Common`:ポーリング・モードにおいてビットレートに応じた設定を行う前に読み込まれます。
- `Listen Common`:リッスン・モードにおいてビットレートに応じた設定を行う前に読み込まれます。
- `Power Level n`:特定の電力レベルに移行する場合に読み込まれます（DPO との併用が可能です）。

注 ここに示した設定 / レジスタ・ビットはデバイスのアナログ的な側面に関連するものだけです。デバイスの適切な動作に必要なその他のビットは操作されず、通常のドライバ実行によって処理されます。このようなアナログ設定テーブルの計算や操作には、細心の注意と深い経験が必要です。

図 2 に、これらのアナログ設定シーケンスの詳細を示します。

図 2. アナログ設定シーケンス



注 RFAL にユーザ定義テーブルの使用を指示する必要が生じた場合は、以下のカスタム·アナログ設定の手順を参照してください。

RFAL ライブラリには、リリース検証のテスト時に使用されるデフォルトのアナログ設定(AC)テーブルが付属しています。

AC の読み込みは階層的に実行されます。複数の AC エントリで同じ設定ビットのアドレスが指定された場合、最後の設定がそれ以前の設定に関係なく優先的に適用されます。

注 しばしば誤解されるのが、AC エントリの `POLL_COMMON` または `LISTEN_COMMON` が、他のいかなるモードやビットレート固有のエントリよりも前に読み込まれるというものです。`POLL/LISTEN_COMMON` エントリが毎回確実に読み込まれるように、この AC は `rfalSetBitrate()` によって読み込まれますが、これはモード固有の AC エントリの読込み後に `rfalSetMode()` によって実行されます。

`rfalSetMode()` は、モードに依存する設定とビットレートに依存する設定の両方を処理するため、特に使い方が複雑です。このため、`rfalSetBitrate()` の実行は、`rfalSetMode()` に内部でインクルードされています。

1. `POLL_NFCA_COMMON_TX`
2. `POLL_NFCA_COMMON_RX`
3. `POLL_ COMMON`
4. `POLL_NFCA_106_TX`
5. `POLL_NFCA_106_RX`

106 kbps の NFC-A ポーラ向けに `rfalSetMode()` API を実行する場合、上記のシーケンスを読み込む必要があります。

カスタム·アナログ設定

この場合、ユーザ定義のカスタム AC テーブルが必要です。

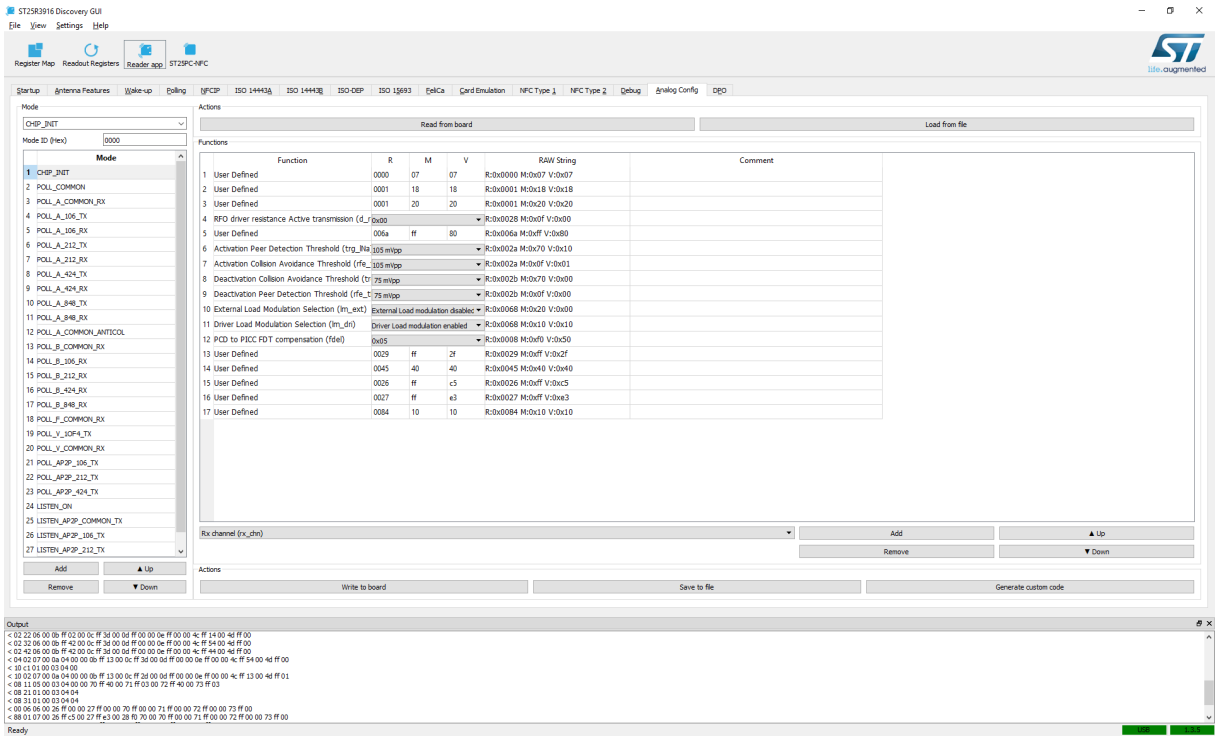
カスタム·テーブルを有効にするには、次の定義変数とともに、`RFAL_ANALOG_CONFIG_CUSTOM` プリプロセッサ定義を追加する必要があります。

- `rfalAnalogConfigCustomSettings`:同じ RFAL フォーマットに従ったカスタム設定を含むアナログ設定テーブルです。
- `rfalAnalogConfigCustomSettingsLength`:`rfalAnalogConfigCustomSettings` テーブルのサイズをバイト単位で表した変数です。

このようなカスタム AC テーブルの例は、いくつかのデモ·プロジェクトに含まれています(ST25R3916 NUCLEO など)。

AC テーブルをカスタマイズする手順は複雑であり、細心の注意と一定レベルの経験が必要です。この手順を容易にするために、DISCO GUI の使用を強く推奨します。

図 3. アナログ設定タブ



DISCO GUI には、AC 専用のタブが表示されます（デモ HW/FW が存在しない場合も）。このタブから、AC テーブルの読込み、視覚化、設定変更、プロジェクトにインクルードするカスタム AC ファイルの生成を実行できます。

RF

RF モジュールは RFAL ライブラリの要となる機能を果たします。必要な処理は、すべてこのモジュールが提供するので、NFC IC ごとに個別のサポートが可能になります。

NFC ドライバを初期化し、適切なモードに設定して必要な動作を実行する API を提供します。これに加え、RF モジュールは、NFC 関連のタスクの定期的な処理を必要とする、いくつかの重要な動作やプロセス（`rfalWorker()`：ワーカと呼ばれます）を実行する役割も担います。ワーカによる動作やプロセスには次のようなものがあります。

- トランシーバ動作
  トランシーバ動作は、ライブラリ全体の中でも最も一般的なプロシージャです。任意のデータの送受信を行ううえで、プロトコルの選択に関係なく最も簡単な方法です。トランシーバ動作中、プロセス / ワーカは AFE を制御、監視して、要求されたデータ交換を完了させます。いくつかの重要な動作（HW FIFO の再読込み、EMD 処理、その他）に求められる厳密なタイミングを守れるように、プロセス / ワーカ（`rfalWorker()`）は十分な頻度で実行する必要があります。
- リッスン・モード
  リッスン・モードは、リッスン・モード、PICC または CE モードによる NFC 通信の確立に使用します。リスナは複数のプロトコルとビットレートでアクティブ化されるため、このプロセスはサポート対象の、あるいは許容されるモード / プロトコルを設定したうえで、そのカードのアクティブ化動作全体を監視、処理します。このモードは、下位のアクティブ化の完了後、ポーラからの最初のデータ・パケットが受信された時点で終了します。RF レイヤでは、プロトコルのアクティブ化前に、次に詳細を示すパケットによりカードのアクティブ化を完了させます。
  – NFC-A：UID/NFC ID 選択後に最初に受信されるパケット（T4T エミュレーションの場合は RATS コマンド、P2P の場合は `ATR_REQ`）
  – NFC-F：`SENSF_REQ` 後に最初に受信されるパケット（T3T エミュレーションの場合はチェック / 更新コマンド、P2P の場合は `ATR_REQ`）
  – AP2P：`ATR_REQ` を受信した最初のデータ

  アクティブ化が完了し、データ・パケットを受信すると上位 / プロトコル・レイヤがプロトコルのアクティブ化が適切に実行されたことを確認し、適切なモードを設定し、トランシーバ動作を使用してデータを交換します（ポーラ・モードと同様）。

- ウェイクアップ·モード
  このモードでは AFE がウェイクアップ·モードに設定されます。このモードでは、デバイスが低電力状態を維持しながら、定期的に固有の測定を実行して、周囲に何らかの変化が発生していないかを評価します。この状況の更新のためにワーカを実行する必要があります。
  ほかにも、一部の ST25R デバイスでは RFAL が SW タグ検出を実行するオプションも使用できます。このモードでは、動作のほとんどが AFE 自体ではなくホストから直接トリガされるため、プロセス / ワーカを頻繁に実行する必要があります。
  `rfal_rf.h` に含まれる、このレイヤが提供する共通インタフェースとは別に、`rfal_chip.h` が提供するチップ個別のインタフェースを介して、よりチップ固有の動作にアクセスすることも可能です。これにより、ソフトウェア·レイヤを、実際の AFE 自体に依存せずに動作できる状態に保ちつつ、汎用のインタフェースによって、レジスタ読出しやコマンド実行などの特定の操作を実行できます。

2.6.2 RF AL

RFAL HAL の下位ドライバの上に配置される NFC プロトコル·スタックは、このレイヤで提供されます。ここで説明するモジュールは、いずれもチップに依存せず、機能、テクノロジー、プロトコルなどの個別サポートに重点が置かれています。

不要なモジュールを個別に無効化 / 除外できるので、多くのアプリケーション·ニーズに対する最適化が可能です。

注 RFAL の各機能 / モジュールを有効化 / 無効化する方法は、セクション 4.2 を参照してください。

テクノロジー

これらのモジュールは、複数の NFC テクノロジーのサポートや便利なメソッドを提供します。サポート対象のテクノロジーごとに、固有のニーズに応じた共通タスクへのインタフェースを利用できます（その大部分は[1]に詳述されています）。これらの動作の例として、次のようなものがあります。

- 初期化
- テクノロジー検出
- コリジョン解決
- アクティブ化
- スリープ / 非アクティブ化
- コマンドや応答の計算 / 解析
- テクノロジー / カード固有の動作

NFC-A

このモジュールは、NFC-A（ISO14443A）準拠デバイスのサポートに必要な機能を提供します。アナログ、デジタル、プロトコルの側面から NFC-A に対応するように、AFE を設定することを可能にします。ポーラ（ISO14443A PCD）のインタフェースに加えて、いくつかの NFC-A リスナ（ISO14443A PICC）ヘルパを提供します。テクノロジー検出、コリジョンの防止 / 解決、カード選択 / アクティブ化、スリープ / 非アクティブ化の実行をサポートします。

すべての NFC-A タグ·タイプ（T1T、T2T、T4T）の検出と選択は、ISO14443-3 レイヤまでの定義およびヘルパ·メソッドを提供する、このモジュールに実行させます。検出、選択の後は、特定のタグ·タイプに固有の動作を専用モジュールによって実装します。

NFC-B

このモジュールは、NFC-B（ISO14443B）準拠デバイスのサポートに必要な機能を提供します。アナログ、デジタル、プロトコルの側面からこのテクノロジーのポーラ·デバイス（ISO14443B PCD）に対応するように、AFE を設定することを可能にします。テクノロジー検出、コリジョンの防止 / 解決、カードのアクティブ化、スリープ / 非アクティブ化の実行をサポートします。

このモジュールが提供する定義およびヘルパ·メソッドは、ISO14443-3 レイヤまでカバーします（ISO-DEP / ISO14443-4 への移行に使用する ATTRIB コマンドを除く）。

NFC-F

このモジュールは、NFC-F（FeliCa - JIS X6319-4）準拠デバイスのサポートに必要なインタフェースを提供します。アナログ、デジタル、プロトコルの側面からこのテクノロジーのポーラ·デバイス（FeliCa PCD）に対応するように、AFE を設定することを可能にします。テクノロジー検出、コリジョンの防止 / 解決、カードのアクティブ化の実行をサポートします。

メモリ·ブロックの読出し、書込みを行う T3T コマンドも使用できます（`CHECK/UPDATE`）。

NFC-V

このモジュールは、NFC-V（ISO15693）準拠デバイスのサポートに必要な機能を提供します。アナログ、デジタル、プロトコルの側面からこのテクノロジーのポーラ·デバイス（ISO15693 VCD）に対応するように、AFE を設定することを可能にします。テクノロジー検出、コリジョンの防止 / 解決、カード選択 / アクティブ化、スリープ / 非アクティブ化の実行をサポートします。

一般的な T5T コマンドも使用できます。

注　このモジュール内の API は、ISO15693 で想定されている、次の 3 種類のモードによるタグ·アクセスを可能とします。

- Addressed モード：Inventory コマンドの後、特定のカードを選択せず、アドレス指定するタグの NFCID/UID をコマンドごとに指定します。
- Non-Addressed モード：特定のカードを選択せず、交換されるコマンドには固有の NFCID/UID が含まれません。このため、周辺にあるすべてのタグがコマンドに反応します（ブロードキャスト）。
- Select モード：Inventory コマンドの後、NFCID/UID を使用して特定のカードを 1 つ選択し、その後のコマンドでは NFCID/UID を指定しません。

これらのモードは、フラグおよび API の UID パラメータと `rfalNfcvPollerSelect()` を使用して制御されます。

T1T

このモジュールは、T1T 準拠デバイスのサポートに必要な機能を提供します。このモード向けに AFE を設定し、タグ固有のコマンド交換に対応できるようにします。メモリ·ブロックの読出し、書込みを行う T1T コマンドも、このモジュールで実装します（`RID`、`RALL`、`WRITE_E`）

T2T

このモジュールは、T2T 準拠デバイスと、それらのメモリ·ブロックの読出し、書込みに必要なコマンドをサポートする機能を提供します（`READ`、`WRITE`、`SECTOR SELECT`）。

T4T

このモジュールは、T4T APDU を NFC フォーラムおよび ISO7816-4 に従って計算、解析するインタフェースにより、T4T（ISO7816-4）向けの便利なメソッドと定義を提供します。

実際のデータ交換は一切実行せず、単に APDU の計算と解析のみを行います。その後 APDU はプロトコル層 ISO-DEP（ISO144443-4）を介して転送されます。

ST25TB

このモジュールは、ST25TB 準拠デバイス（SRTx、SRIx）のサポートに必要な機能を提供します。AFE を設定できるようにして、このテクノロジーのポーラ·デバイス（ISO14443B PCD）としての機能を、アナログ、デジタル、プロトコルの側面からサポートします。テクノロジー検出、コリジョンの防止 / 解決、カード選択 / アクティブ化、スリープ / 非アクティブ化のほかに、メモリ·アクセスなどのカード固有動作の実行をサポートします。

プロトコル

カードのアクティブ化に成功すると、通常はポーラとリスナ（PCD と PICC）の間でデータ交換が実行されます。このデータ交換は、単純なコマンドと応答のペアのシーケンスで実現される場合と、より高度なプロトコルの形を取る場合があります。

このようなプロトコルは、信頼性の高いデータ交換を保証する、より堅牢なメカニズムを提供し、長いフレームを複数のチャンクに分割したり、エラー発生時に固有の処理を定義し、通信中のリンクを切断せずにリカバリできるようにしたりする、データ·チェーンのメカニズムを実現します。これらのモジュールは、こうしたプロトコルを使用できるように、プロトコル固有の機能（フレーミング、タイミング、エラー処理など）のすべてに対応します。

ISO-DEP

このモジュールは、ISO-DEP（ISO14443-4）プロトコル·リンクの確立とデータ交換の実行をサポートします。ポーラとリスナの両方のモードでプロトコルのアクティブ化と非アクティブ化を処理します。プロトコルのアクティブ化が完了すると、トランシーバ動作によりデータ（通常は APDU）を交換します。

提供されるトランシーバ·インタフェースは非ブロッキングで、次の 2 種類の方法で使用できます。

- ブロック：APDU の全体または一部を含む、単一のデータ・ブロック（I-Block）を交換します。送信時、呼び出し元はブロックが完全なデータか部分的な（チェーン接続された）データ・チャンクかを示す必要があります（入力パラメータ `isTxChaining` 参照）。同様に、受信時にチェーン接続されたパケットを受信すると、出力パラメータ `isRxChaining` がセットされ、特別なリターン・コード（`ERR_AGAIN`）が送信されます。このインタフェースは、パケット制御、エラー処理、繰り返しなど、すべてのプロトコル固有動作に対応し、成功またはリカバリ不能なエラーが発生した場合は失敗のコードを返します。
- APDU：完全な APDU を交換します。デバイスの 1 つがサポートするアナウンスされた長さよりもデータが長い場合、APDU をより小さなチャンクつまりチェーン接続されたパケットに自動的に分割し、送受信の両方でチェーン制御を自動的に行います。このインタフェース（内部ではブロック・インタフェースを使用）によりプロトコル固有機能に対応するには、サポート対象の最大フレーム・サイズを備えた一時バッファを追加する必要があります。

NFC-DEP

このモジュールは、NFC-DEP（ISO18092、NFCIP1、P2P）プロトコル・リンクの確立とデータ交換の実行をサポートします。ポーラとリスナ（イニシエータとターゲット）の両方のモードでプロトコルのアクティブ化と非アクティブ化を処理します。

このプロトコル・レイヤは、パッシブ P2P とアクティブ P2P に使用します。パッシブ P2P ではデバイスのコリジョン防止と選択が通常の方法で実行され、アクティブ P2P の場合は、最初の RF コリジョン防止の直後に `ATR_REQ` が送信されます。

プロトコルのアクティブ化が完了すると、トランシーバ動作によりデータ（通常は LLCP パケット）を交換します。

提供されるトランシーバ・インタフェースは非ブロッキングで、次の 2 種類の方法で使用されます。

- ブロック：PDU の全体または一部を含む、単一のデータ・ブロック（I-PDU）を交換します。送信時、呼び出し元はブロックが完全なデータか部分的なデータ・チャンクか（チェーン接続されたもの）を示す必要があります（入力パラメータ `isTxChaining` 参照）。同様に、受信時にチェーン接続されたパケットを受信すると、出力パラメータ `isRxChaining` がセットされ、特別なリターン・コード（`ERR_AGAIN`）が送信されます。このインタフェースは、パケット制御、エラー処理、繰り返しなど、すべてのプロトコル固有動作に対応し、成功またはリカバリ不能なエラーが発生した場合は失敗のコードを返します。
- PDU：完全な PDU を交換します。デバイスの 1 つがサポートするアナウンスされた長さよりもデータが長い場合、PDU をより小さなチャンクつまりチェーン接続されたパケットに自動的に分割し、送受信の両方でチェーン制御を自動的に行います。このインタフェース（内部ではブロック・インタフェースを使用）によりプロトコル固有機能に対応するには、サポート対象の最大フレーム・サイズを備えた一時バッファを追加する必要があります。

注　どちらのプロトコルでも（ISO-DEP と NFC-DEP）、チェーン接続されたブロック /DEP が受信されると、エラー `ERR_AGAIN` が送信されます。この時点で、呼び出し元はただちに受信データを処理 / 保存する必要があります。`ERR_AGAIN` が返される時点では、既に他のデバイスに ACK が送信されているため、次のブロックを着信する可能性があります。`rfalWorker()` は、頻繁に呼び出されると、トランシーバ動作のパラメータで指定された同じバッファに次のブロックを配置します。

注　どちらのプロトコルでも（ISO-DEP と NFC-DEP）、リスナのデバイスがアクティブと見なされるのは、最初のデータ・パケットまたはプロトコル・パラメータ選択（PPS または PSL）の受信後です。したがって、リッスン・モードでのアクティブ化を処理するインタフェースは、最初のデータ・パケットを、アクティブ化パラメータを渡すバッファに保持します。リスナのアクティブ化が完了したら（`rfalIsoDepListenGetActivationStatus()` / `rfalNfcDepListenGetActivationStatus()` が `ERR_NONE` を返したら）、`rfalIsoDepGetTransceiveStatus()` / `rfalNfcDepGetTransceiveStatus()` メソッドを呼び出す必要があります。

注　PPS/PSL ではなくデータ・ブロックの受信によってアクティブ化が完了した場合、呼び出し元が所有し、リッスンのアクティブ化パラメータが渡されるバッファに、そのデータが残っていなければなりません。最初のデータは `rfalIsoDepGetTransceiveStatus()` によって処理され、呼び出し元に通知したうえで、`rfalIsoDepStartTransceive()` / `rfalNfcDepStartTransceive()` を先頭に、次のトランザクション・フローを通常どおり進めることができます。

2.6.3 RF HL

このレイヤは、デバイス固有のドライバ（RF HAL）および必要な NFC テクノロジーとプロトコルのスタック（RF AL）の上位に配置されます。NFC フォーラム動作（NFCC）、EMVCo、DISCO/NUCLEO デモなどで RFAL 機能を利用する、アプリケーション / 上位レイヤです。

NFC フォーラム準拠のポーラやリスナ・デバイスとしての動作など、一般的な NFC 関連タスク向けの便利な抽象化を可能とします。

このレイヤには、次のようなモジュールが含まれます。

NFC

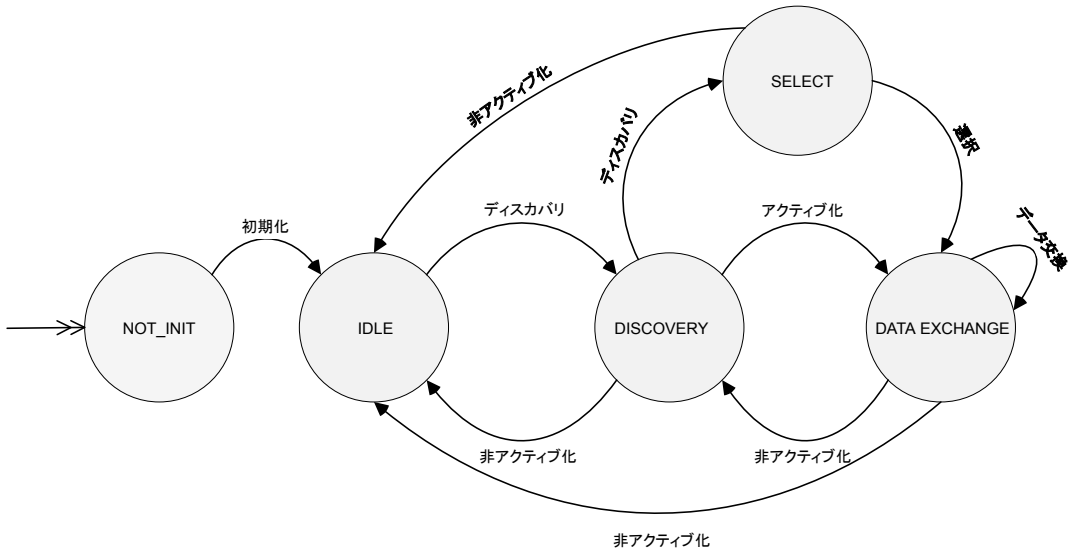
このモジュールは、NFC フォーラム準拠のポーラとリスナ·デバイスを簡単に実装する、シンプルなインタフェースを提供します。すべての NFC テクノロジーを統合して使いやすくするとともに、テクノロジー検出、コリジョン解決、アクティブ化、データ交換、非アクティブ化といった複数の NFC 動作[1]を実行するために必要な機能を実現します。

NFC モジュールは、特に動作[1]と NCI[3]に関して NFC フォーラム仕様の影響を受けますが、同仕様に完全に整合しているわけではありません。

このモジュールの狙いは、使用方法をわかりやすく保ったまま、一般的な NFC の実装をサポートすることです。RFAL で使用できるパラメータと設定のすべてが、このモジュールのインタフェースにエクスポートされるわけではありません。

下図は、NFC フォーラムの NCI 推奨事項を模した内部ステート·マシンです。矢印は、呼び出し元がこのモジュールを制御するために使用できる API を表し、円が状態を表します。

図 4. NFC モジュールの状態図



各状態と遷移は、以下を表します。

- 状態
  - NOT_INT：初期化が完了していない、起動時の状態です。
  - IDLE：モジュール /RFAL の初期化が完了し、使用できる状態です。
  - DISCOVERY：ディスカバリ·サイクルの実行中です。動作は、設定されたモードとテクノロジーによって異なります。内部では、モジュールが、指定された設定および存在するデバイスに応じて、複数の動作や状態（ウェイクアップ、テクノロジー検出、コリジョン解決、アクティブ化）の間を遷移しています。
  - DATA EXCHANGE：データ交換が実行中であること、または完了したことを表します。この状態になると、呼び出し元はデータ交換を実行するか、単にデバイスを非アクティブ化します。
  - SELECT：複数のデバイスが識別された場合、呼び出し元に通知し、実行すべきアクションを待機します。アクションとは、特定のリスナ·デバイスのアクティブ化 / 選択、または非アクティブ化のいずれかです。
- 遷移（コマンド / 通知）：
  - 初期化：RFAL の初期化を実行し、IDLE 状態に移行します。
  - ディスカバリ：モジュールを、指定された設定に従ってディスカバリ·モードに移行します。
  - アクティブ化：デバイスがアクティブ化されると（ポーラまたはリスナ）、通知が発生し、データ交換を実行できるアクティブ化された状態に移行するか、単にデバイスを非アクティブ化します。
  - データ交換：データ交換を実行中または完了。
  - 選択：複数のデバイスが識別された場合、呼び出し元に通知し、実行すべきアクションを待機します。アクションとは、特定のリスナ·デバイスのアクティブ化 / 選択、または非アクティブ化のいずれかです。

NFC モジュールは独自のステート·マシンを備えているため、独自のワーカ / プロセスである rfalNfcWorker() を使用して NFC 関連の動作を継続的に制御します。この独自ワーカは内部で RFAL の rfalNfcWorker() を呼び出すことで、呼び出し元が両方のワーカを実行する負担を軽減します。

このモジュールにより、ユーザは特定の非同期イベント(NCI 用語での通知)を監視するために、通知コールバックを登録できます。

たとえば、設定したデバイスの制限が許せば DISCOVERY 状態では複数のタグが検出されます。その場合、SELECT 状態に到達したこと、識別されたタグのいずれか 1 つを選択する必要があることが呼び出し元に通知されます。

通知コールバック(notifyCb)の使用は必須ではなく、呼び出し元には NFC タスクの状態をポーリングして適宜アクションを実行するという選択肢もあります。

デバイスのアクティブ化後、データ交換が他のレイヤと同じ方法で実行されます。トランシーバ動作とデータ交換のメソッドのペア rfalNfcDataExchangeStart() / rfalNfcDataExchangeGetStatus() を使用できます。デバイスのアクティブ化中は、特別なインタフェースが選択されます。このインタフェースが、アクティブ化と非アクティブ化のシーケンスの種類と、データ交換に使用するプロトコルを決定します。次の 3 つのインタフェースを使用できます(NCI に対応 [3])。

- ISO-DEP:ISO-DEP(ISO4443A)プロトコルが使用されます。
- NFC-DEP:NFC-DEP(ISO18092、NFCIP1、P2P)プロトコルが使用されます。
- RF:特定のプロトコル / フレーミングは使用されず、送受信時には呼び出し元に対して完全なパススルーが許可されます。

注 幅広いテクノロジーとプロトコルをサポートするために、RF インタフェースを使用するときの長さはバイト数ではなくビット数で指定します。したがって rfalNfcDataExchangeStart() / rfalNfcDataExchangeGetStatus() の入力 txDataLen と出力 rvdLen は、いずれもビット数を表します。ISO-DEP または NFC-DEP インタフェースを使用する場合、これらのパラメータはバイト数で指定します。

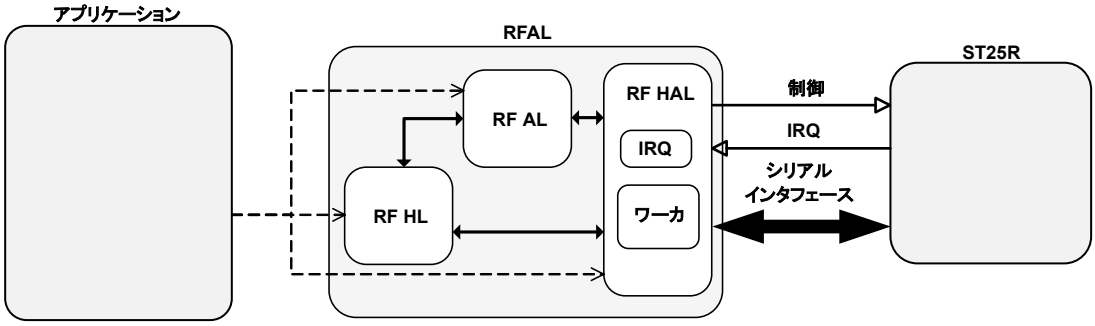
注 プロトコルのセクションで、リッスン·モードおよびアクティブ化状態に到達したら、はじめに txDataLen をゼロに設定して rfalNfcDataExchangeStart() メソッドを呼び出し、rxData の最初のデータ·パケットと rcvLen の場所を取得する必要があります。

注 セクション 2.3.1 で規定したように、ライブラリは、次元が極めて大きいバッファを一切割り当てないため、ユーザが提供する必要があります。しかし、このモジュールはその規則の例外です。ユーザの利便性のためにトランシーバのバッファは内部に静的に割り当てられるからです。それでも、この種のメモリのサイズは、セクション 4.2 に示した設定を介してユーザが定義します。

2.6.4 概要

図 5 に示すとおり、RFAL はいくつかの積層化されたレイヤとモジュールによって構成されます。これらは、提供する機能と依存関係に基づいてライブラリ内に体系化されています。

図 5. システムの概要



RFAL には次のコンポーネントが含まれます。

- ST25R
  下位ドライバが対応するインタフェース。次の 3 つのインタフェースを介して AFE の通信 / 制御 / 監視を行います。
  – 制御ライン（RESET など）：デバイスの起動または再初期化時
  – シリアル・インタフェース（SPI、I^2C、UART）：デバイスのレジスタへのアクセス、コマンドの実行など
  – IRQ ライン / 信号：新しいイベント / 割込みを通知するために ST25R デバイスが使用します。そのようなイベントが発生した場合、ホストは可能なかぎり迅速に IRQ 情報を取得し（外部割込みまたはポーリングに対する ISR によって）、その内容に従って応答する必要があります。
- RFAL ワーカ
  ワーカ / プロセスは、RFAL ライブラリの中核をなす部分です。IRQ 信号を受信したら、ISR によって IRQ ステータス・レジスタの内容を取得し（そのように設定されている場合）、制御をアプリケーションのコンテキストに戻します。いくつかの動作 / モードは、ST25R からの新しい信号の処理と、これに対応する特定の動作を実行するために、RFAL ワーカ / プロセスの実行を必要とします。RFAL を適切に動作させるには、RFAL ワーカの適切な処理を保証することが極めて重要です。

ライブラリは、複数の階層的なレイヤから構成されていますが、上位および下位のレイヤと同時に対話できます。こうした細かい制御が許されているのは、各プロジェクトのさまざまなユース・ケースと特殊なニーズに対応するためですが、同時にライブラリの内部メカニズムに対する知識も要求されます（RFAL HL NFC はインタフェースを使用する最も統合され、より簡単な手段を許可します）。

たとえば、アプリケーションに AP2P モードによるポーリングが必要だとします。これを実現するために、ユーザがアクセスできるモジュールがいくつか用意されています。初期化、タイミング、設定、RF フィールド制御のための下位 RF モジュールや、プロトコルのポーリングや処理を実際に実行する NFC-DEP モジュールなどです。

セクション 付録 B に、各種モジュール間の相互作用を詳細に示したシーケンス図を多数収録しました。

2.6.5 同時使用

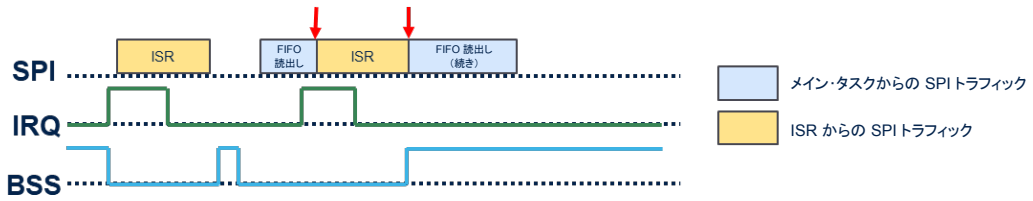
ライブラリを組み込む場合、同時使用の側面に配慮し、ハードウェア・リソースへの秩序あるアクセスを保証する必要があります。ベア・メタルであるのか、RTOS や組込み Linux などのマルチスレッド環境であるのかなど、システム自体の特性によって、取り組むべき同時使用の複雑性は異なります。

シリアル・インタフェースの不可分性

NFC AFE は、さまざまなコンテキストまたはスレッドでアドレスが指定される、シリアル・インタフェースによって制御されます。各動作の不可分性を保証し、実行中の動作（メイン / ユーザ・コンテキストなど）が、他のコンテキスト（割込み /ISR コンテキストなど）によって中断されないようにすることが重要です。

そのようなシナリオを 図 6 に示します。ここでは、SPI 制御の ST25R に 2 つの割込みが通知されています。これを受けてメイン / ユーザ・コンテキストで動作中のドライバは FIFO 読出し動作をトリガし、受信データを取得します。すると ST25R によって別の割込みが通知され、実行中の SPI 動作 / トラフィックが突然中断されたことで、SPI シーケンスの無効化という結果を招いています。

図 6. 保護なしの同時実行



上記のようなシナリオを回避するために、シリアル・インタフェース上の各動作を保護する必要があります。つまり、共有リソースが要求されたら、これをブロックすることで、複数のコンテキストからの同時アクセスを防ぎます。

図 7. 保護ありの同時実行

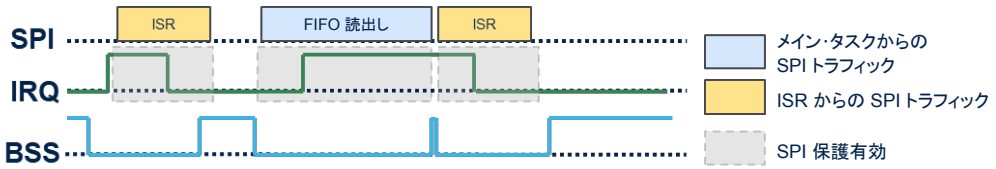


図 7 では、動作中に割込みが発生した場合（IRQ 信号が High）、そのイベントの処理は、シリアル・インタフェースが再度解放されるまで延期されています。

RFAL には、シリアル・インタフェースを保護する手段として API `platformProtectST25RComm()`、`platformUnprotectST25RComm()`、`platformProtectST25RIrqStatus()`、`platformUnprotectST25RIrqStatus()` が用意されています。詳細は、セクション 4.1 を参照してください。保護の実際の実装方法は、プラットフォームやシステム・アーキテクチャごとに異なり、さまざまな形態を取ります。ベア・メタルの組込みシステムで実行する場合は割込みイネーブルのメカニズムを使用し、マルチスレッド環境の場合はミューテックスを使用することを推奨します。こうした保護メカニズムの例は、st.com より入手可能な複数のデモを通して確認できます。具体的には、X-CUBE-NFC3、X-CUBE-NFC5、STSW-ST25R-LIB のほか、セクション 付録 A でも紹介しています。

マルチスレッド環境

RTOS、組込み Linux などのマルチスレッド環境で実行する場合は、ほかにも考慮が必要な制約があります。これらの環境向けに実装する場合は、通常、割込みステータスに対応するスレッドと、ユーザ / メインのコンテキストに対応する別のスレッドを使用します。

この場合も前述の方法と同じようにして複数のコンテキストからの同時アクセスを保護する必要があり、RFAL ではより詳細な保護メカニズム（通常はミューテックスを使用）を、次の API によって導入できます：`platformProtectST25RComm()`、`platformUnprotectST25RComm()`、`platformProtectST25RIrqStatus()`、`platformUnprotectST25RIrqStatus()`、`platformProtectWorker()`、`platformUnprotectWorker()`。詳細は、セクション 4.1 を参照してください。

こうした保護メカニズムの例は、www.st.com から入手可能な複数のデモを通して確認できます。具体的には、次のようなデモが用意されています。

- STSW-ST25R009
- STSW-ST25R013（Linux 用パッケージ）
- STSW-ST25R-LIB（FreeRTOS 用）

3 リリース

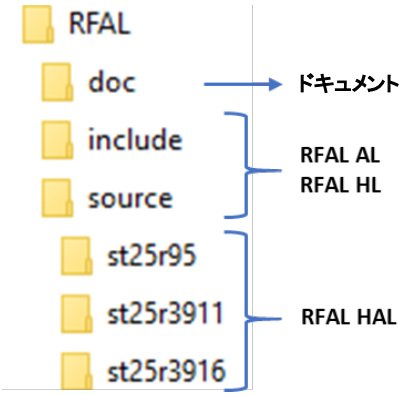
3.1 パッケージの説明

RFAL は、複数のデバイス固有パッケージ（STSW-ST25RFALxxx）の形で st.com より入手できます。各パッケージには、共通の RFAL AL および RFAL HL レイヤに加え、対応する RFAL HAL レイヤが含まれます。

個々の ST25R デバイスに対応する各種 RFAL HAL がフォルダとして同時に提供されています。たとえば異なる ST25R デバイスごとに対応するフォルダが提供されています。

パッケージの構成は次のとおりです。

図 8. RFAL パッケージの構成



- doc/：リリース・ノート、詳細な API ドキュメント（Doxygen フォーマット）、該当するデバイスではコンプライアンス・レポート（例：MISRA）など、RFAL に関するドキュメントが保存されています。
- include/：共通の RFAL AL および RFAL HL レイヤに対応する拡張 .h（ヘッダ）ファイルが保存されています。RFAL を使用するプロジェクトでは、インクルード・パスにこの場所を追加する必要があります。
- source/：共通の RFAL AL および RFAL HL レイヤに対応する .c（ソース）および内部 .h（ヘッダ）ファイルの一部が保存されています。RFAL を使用するプロジェクトでは、インクルード・パスにこの場所を追加する必要があります。
- source/st25r…/：共通の RFAL HAL に対応する .c（ソース）および内部 .h（ヘッダ）ファイルが保存されています。RFAL を使用するプロジェクトでは、インクルード・パスにこの場所を追加する必要があります。

3.2 品質基準

RFAL の各リリースの品質要件を確実に満たすために、リリース・プロセスの一環として一連の品質チェックを実施しています。数々の社内テストに加えて、各リリースの一貫性と品質を確保するために公式のテスト・スイートと静的コード解析ツールを使用しています。ST25R のデバイスごとに固有の機能が存在することから、該当するデバイス固有のテスト・グループが適用されます。次のようなテスト・スイート（TS）を使用します。

- NFC フォーラム・デジタル TS
- EMVCo PCD/PICC デジタル TS
- ISO IEC 10373-6 PCD デジタル TS
- ISO IEC 23917 NFCIP-1 プロトコル TS
- MISRA-C 2012

ツールと TS は、個々の規格の進化に追随するために、提供中の最新バージョンに定期的に更新されます。テスト・レポートは RFAL パッケージの doc/ フォルダに保存されています。さらに詳細なテスト・レポートも、ご要望に応じて提供いたします。

4 RFAL ライブラリの使用方法

RFAL の設定と構成では、いくつかの項目をユーザが定義する必要があります。

これらの設定は、すべての RFAL モジュールが使用する `platform.h` という名称のファイルで指定します。設定例は、X-CUBE-NFC3、X-CUBE-NFC5、STSW-ST25R-LIB、および セクション 付録 A を参照してください。

4.1 ペリフェラルの初期化と設定

セクション 2.4.1 で説明したように、RFAL を実行するには、いくつかのコンポーネントが必要です。ターゲット・プラットフォーム上で GPIO、外部割込みソース、シリアル・インタフェースの初期化および RFAL に対して提供 / 定義される HAL インタフェースの初期化を(ST25R のデバイス要件に従って)確実に行う必要があります。

RFAL の設定、プラットフォーム固有の HAL、インクルードするシステムおよびインタフェースの定義は、`platform.h` ファイルに集められ、ユーザはニーズに合わせて、以下のような項目を調整する必要があります。

ピン定義

- `ST25R_SS_PIN / ST25R_SS_PORT`:SPI のチップ選択(CS)/ スレーブ選択(SS)に使用するピンとポートの定義です(必須)。SPI の CS は、SPI ペリフェラルによって自動的に制御されるわけではなく、ソフトウェアで制御する必要があります。HAL ドライバが GPIO のポートとピンのアドレス指定に単一の定義を使用する場合、定義を 1 つだけ記述したうえで、それを GPIO マクロで使用します。
- `ST25R_INT_PIN / ST25R_INT_PORT`:ST25R の IRQ ラインで使用するピンとポートの定義です(必須)。ISR は、この GPIO の状態を確認します。HAL ドライバが GPIO のポートとピンのアドレス指定に単一の定義を使用する場合、定義を 1 つだけ記述したうえで、それを GPIO マクロで使用します。
- `PLATFORM_LED_RX_PIN / PLATFORM_LED_RX_PORT`:受信 LED に使用するピンとポートの定義です(任意)。定義した場合、受信中に LED がこの GPIO を High に駆動します。オプションの LED 機能を使用しない場合は、この定義を削除してください。HAL ドライバが LED のポートとピンのアドレス指定に単一の定義を使用する場合、定義を 1 つだけ記述したうえで、それを GPIO マクロで使用します。
- `PLATFORM_LED_FIELD_PIN / PLATFORM_LED_FIELD_PORT`:フィールド LED に使用するピンとポートの定義です(任意)。定義した場合、フィールドがオンになると、RFAL がこの LED を High に駆動します。このオプション機能は、この定義を削除することで無効化されます。HAL ドライバが GPIO のポートとピンのアドレス指定に単一の定義を使用する場合、定義を 1 つだけ記述したうえで、それを LED マクロで使用します。

ユーザ固有プラットフォームのハードウェア・ドライバへのインタフェース(MCU HAL)

- `platformProtectST25RComm()`
  ST25R 通信チャネルへの一意のアクセスを保護します。シングル・スレッド環境(マイクロコントローラ)では IRQ を無効化し、マルチスレッド環境ではミューテックスのロックをオンにします(必須)。
- `platformUnprotectST25RComm()`
  ST25R 通信チャネルへの一意のアクセスの保護を解除します。シングル・スレッド環境(マイクロコントローラ)では IRQ を有効化し、マルチスレッド環境ではミューテックスのロックをオフにします(必須)。
- `platformProtectST25RIrqStatus()`
  IRQ ステータス変数への一意のアクセスを保護します。シングル・スレッド環境(マイクロコントローラ)では IRQ を無効化し、マルチスレッド環境ではミューテックスのロックをオンにします(任意)。

- `platformUnprotectST25RIrqStatus()`
  IRQ ステータス変数の保護を解除します。シングル・スレッド環境（マイクロコントローラ）では IRQ を有効化し、マルチスレッド環境ではミューテックスのロックをオフにします（任意）。
- `platformProtectWorker()`
  RFAL ワーカの単独実行を保護 / ロックします（任意）。不要の場合は、このマクロは空のままにしておく必要があります。
- `platformUnprotectWorker()`
  RFAL ワーカの個別実行の保護 / ロックを解除します（任意）。不要の場合は、このマクロは空のままにしておく必要があります。
- `platformIrqST25RPinInitialize()`
  ST25R の IRQ ピンを入力 / 外部割込みとして初期化します（任意）。RFAL は初期化時に、このマクロを呼び出して IRQ ピンを設定します。GPIO がユーザによって既に設定されている場合、このマクロは空のままにしておく必要があります。
- `platformIrqST25RSetCallback( cb )`
  ST25R の ISR を設定します（任意）。RFAL は初期化時に、このマクロを呼び出して IRQ ピンのコールバック（cb）を設定します。割込みがユーザによって既に設定され、IRQ 発生により `st25r391xIsr()` が呼び出される場合、このマクロは空のままにしておく必要があります。
- `platformLedsInitialize()`
  LED を出力として設定します（任意）。RFAL は初期化時に、このマクロを呼び出して必要なピンを出力として初期化します。LED を使用しない場合、またはユーザによって既に設定されている場合、このマクロは空のままにしておく必要があります。
- `platformLedOff( port, pin )`
  LED をオフにします（任意）。LED を使用しない場合、このマクロは空のままにしておく必要があります。
- `platformLedOn( port, pin )`
  LED をオンにします（任意）。LED を使用しない場合、このマクロは空のままにしておく必要があります。
- `platformGpioSet( port, pin )`
  GPIO を論理 High の状態に設定します（必須）。
- `platformGpioClear( port, pin )`
  GPIO を論理 Low の状態に設定します（必須）。
- `platformGpioToogle( port, pin )`
  GPIO の論理状態を反転します（必須）。
- `platformGpioIsHigh( port, pin )`
  GPIO の状態が論理 High であるかどうかを確認します（必須）。
- `platformGpioIsLow( port, pin )`
  GPIO の状態が論理 Low であるかどうかを確認します（必須）。
- `platformTimerCreate( t )`
  ミリ秒単位で指定した時間（t）で満了するタイマを作成します（必須）。これによって、定義した時間が経過するタイムアウトまで、タイマ / ティック /参照値が計算されます。タイマ / ティック /参照値は符号なし 32 ビット変数として返されます。
- `platformTimerIsExpired( timer )`
  作成したタイマが満了しているかどうかを確認します（必須）。このマクロは、`platformTimerCreate()` が返すタイマ / ティック /参照値を受信し、満了している場合に True を返します。
- `platformTimerDestroy( timer )`
  タイマを作成済みの場合、指定したタイマを停止、解放します（任意）。不要の場合は、このマクロは空のままにしておく必要があります。

- `platformDelay( t )`
  ミリ秒単位で指定した時間(t)だけ、マイクロコントローラを遅延させます / ブロックします(必須)。
- `platformGetSysTick()`
  ミリ秒単位で現在のティック·カウンタの値を返します(任意)。不要の場合は、このマクロは空のままにしておく必要があります。
- `platformAssert()`
  指定した式が True であるかどうかをアサートします。ログ記録、エラー処理、またはその他の場合はトラップ(任意)。不要の場合は、このマクロは空のままにしておく必要があります。
- `platformErrorHandle()`
  グローバル·エラーを処理またはトラップします(任意)。不要の場合は、このマクロは空のままにしておく必要があります。
- `platformSpiSelect()`
  SPI の SS/CS つまりチップ選択 | スレーブ選択をオンにします(必須)。
- `platformSpiDeselect()`
  SPI の SS/CS つまりチップ選択 | スレーブ選択をオフにします(必須)。
- `platformSpiTxRx( txBuf, rxBuf, len )`
  SPI のトランシーバ動作を実行します(必須)。指定したバッファとの間で指定したバイト数(len)のデータを送受信します。SPI バスには、指定したバイト数(len)に対応するサイクル数だけクロックを供給する必要があります。送信バッファ(txBuff)が NULL の場合、受信のみを実行する必要があり、MOSI ラインにはゼロが出力され、MISO ラインのバイトが rxBuff に格納されます。受信バッファ(rxBuff)が NULL の場合、送信のみを実行する必要があり、MOSI ラインから送信され、MISO ラインからのデータは無視されて rxBuff には格納されません。
- `platformLog( ... )`
  指定したメッセージをログに記録します(任意)。

上記の定義およびインタフェースには、必須のものと任意のものがあります。RFAL 自体から呼び出される IRQ/GPIO 初期化、ST25R の LED 処理、グローバル·エラーの処理 / トラップなど、具体的な機能 / 拡張は不要です。

4.2 ライブラリの設定

RFAL では柔軟性に富んだ幅広い設定が可能です。不要なモジュールの無効化、グローバル·バッファのサイズ調整のほか、一部の固有動作も調整できます。機能の有効化 / 無効化については、次のようなユーザ定義のスイッチを使用できます。

- `RFAL_FEATURE_NFCA` - NFC-A(ISO14443A)に対する RFAL サポート
- `RFAL_FEATURE_NFCB` - NFC-B(ISO14443B)に対する RFAL サポート
- `RFAL_FEATURE_NFCF` - NFC-F(FeliCa)に対する RFAL サポート
- `RFAL_FEATURE_NFCV` - NFC-V(ISO15693)に対する RFAL サポート
- `RFAL_FEATURE_T1T` - T1T(Topaz)に対する RFAL サポート
- `RFAL_FEATURE_T2T` - T2T(MIFARE Ultralight)に対する RFAL サポート
- `RFAL_FEATURE_T4T` - T4T に対する RFAL サポート
- `RFAL_FEATURE_ST25TB` - ST25TB に対する RFAL サポート
- `RFAL_FEATURE_ST25xV` - ST25TV/ST25DV に対する RFAL サポート
- `RFAL_FEATURE_ISO_DEP` - ISO-DEP(ISO14443-4)に対する RFAL サポート
- `RFAL_FEATURE_ISO_DEP_POLL` - ポーラ·モード(PCD)ISO-DEP(ISO14443-4)に対する RFAL サポート
- `RFAL_FEATURE_ISO_DEP_LISTEN` - リッスン·モード(PICC)ISO-DEP(ISO14443-4)に対する RFAL サポート
- `RFAL_FEATURE_NFC_DEP` - NFC-DEP(NFCIP1/P2P)に対する RFAL サポート
- `RFAL_FEATURE_LISTEN_MODE` - リッスン·モードに対する RFAL サポート
- `RFAL_FEATURE_WAKEUP_MODE` - ウェイクアップ·モードに対する RFAL サポート

メモリ·リソースに関しては、以下のオプションによって最大ブロック·サイズを設定できます。

- `RFAL_FEATURE_ISO_DEP_IBLOCK_MAX_LEN` - ISO-DEP の I-Block の最大長。rfalIsoDepFSx で定義される値を使用します。
- `RFAL_FEATURE_NFC_DEP_BLOCK_MAX_LEN` - NFC-DEP のブロック / ペイロード長。許容される値:64、128、192、254
- `RFAL_FEATURE_NFC_RF_BUF_LEN` - RFAL NFC レイヤが使用する RF バッファ長。
- `RFAL_FEATURE_ISO_DEP_APDU_MAX_LEN` - ISO-DEP の APDU の最大長。APDU は、1 つまたは複数のチェーン接続された ISO-DEP ブロックから構成されます。

- RFAL_FEATURE_NFC_DEP_PDU_MAX_LEN - NFC-DEP の PDU の最大長。PDU は、1 つまたは複数のチェーン接続された NFC-DEP ブロックから構成されます。

このほかに、以下のプリプロセッサ命令を追加することで、一部の RFAL 内部動作をカスタマイズすることも可能です。

- ST25R_COM_SINGLETXRX - ST25R がシリアル·インタフェース(SPI/I²C)上で単一のトランシーバ動作を使用します(ST25R3911 と ST25R3916 で使用可能)。完全なコマンドを計算して、一意のシリアル動作に使用することで時間を短縮するバッファを追加する必要があります(許容される最大ペイロード長を格納できるサイズのバッファ)。
- RFAL_COLRES_AGC - ビット·コリジョン識別機能を縮小します(ST25R3911 のみで使用可能)。2 つのカード(一方が弱く、もう一方が強い)のコリジョンとして解釈されるような、アクティブ負荷変調におけるノイズが多いデバイス(例:Samsung Galaxy S8-US(SM-G950U))に対して有用です。
- ST25R_SELFTEST - ST25R を有効化します。初期化中にホスト通信のセルフテストを実行します。ポート設定の問題を診断するときに便利です。
- ST25R_SELFTEST_TIMER - 初期化中のタイマのセルフテストを有効化します。ポート設定の問題を診断するときに便利です。

コンパイルしたドライバがユーザの目的とするデバイスに確実に適合するように、ST25R3911B または ST25R3916 または ST25R95 の中からターゲット·デバイスを選んで定義するプリプロセッサ命令の追加も必要です。

4.3 初回のサンプル実行手順

RFAL を既存の FW プロジェクトに設定する必要があります。以下にその手順を、段階を追って説明します。

1. システム·プラットフォームに必要なペリフェラルが適切に設定され、セクション 2.5.1 と セクション 2.4.1 に記載された要件に適した HAL を提供可能であることを確認します。ホスト HAL が 1 対 1 のインタフェースを提供しない場合、アプリケーション上に適応 / グルー·レイヤを追加して、セクション 4.1 に記載されたインタフェースの要件を満たせるようにします。
2. ST25R のサンプルに添付された既存の platform.h ファイルをプロジェクト·フォルダにコピーします(ST25R のすべてのサンプル·プロジェクト、たとえば X-CUBE-NFC3、X-CUBE-NFC5、STSW-ST25R-LIB などの Doxygen で生成されたドキュメント内や、セクション 付録 A から入手できます)。
3. RFAL の設定
   a. プロジェクトの実行に必要なすべてのインタフェース、つまり GPIO、タイマ、シリアル·インタフェースを platform.h 内に定義します。
   b. プロジェクト / アプリケーションにおける必要な機能とプロトコルおよびバッファ·サイズを検討します。検討結果に応じて、機能を有効化 / 無効化します(RFAL_FEATURE_...)。
4. メモリの最適化が必要な場合、次のバッファ配列サイズを設定します(通常はデフォルト値で問題ありません):
   RFAL_FEATURE_ISO_DEP_IBLOCK_MAX_LEN、RFAL_FEATURE_NFC_DEP_BLOCK_MAX_LEN、
   RFAL_FEATURE_NFC_RF_BUF_LEN、RFAL_FEATURE_ISO_DEP_APDU_MAX_LEN、
   RFAL_FEATURE_NFC_DEP_PDU_MAX_LEN
5. プロジェクトのコンパイルに、次のフォルダ内にある必要な .c ファイルをすべて追加します。
   a. rfal/source
   b. rfal/source/st25r…
6. プロジェクトに次のインクルード·パスを追加します。
   a. rfal/include
   b. rfal/source
   c. rfal/source/st25r…
7. ST25R のターゲット·デバイスを指示するプリプロセッサ命令を追加します。
8. 必要なすべてのペリフェラルが適切に初期化され、それらのインタフェースが機能することを確認します。
9. ST25R 固有の ISR メソッドを IRQ ピン、ISR ハンドラに追加します(例:st25r3911Isr()、st25r3916Isr() など)。
10. プロジェクトをコンパイルします。
11. コンパイルに成功した場合、まず次の手順を実行します。
    RFAL が正常に初期化されない場合は、次の確認作業を行います。
    最初の SPI 動作のやり取りを観察して SPI に問題がないことを確認します。最初にリセット / セットのデフォルト·コマンドが発行され、IC の ID レジスタ読出しコマンドが続きます(デバイスのデータシートを参照してください)。IC の ID レジスタの読出し値が予想されるものと異なる場合、エラーが報告されます。
    さらに、自己診断機能(ST25R_SELFTEST、ST25R_SELFTEST_TIMER)を使用して、必要な機能のどのような側面が初期化時に動作していないのかを把握します。
    初期化に成功した場合は、初期化後に rfalFieldOnAndStartGT() を追加して、アンテナ出力から放射される RF キャリアが観測されれば、設定に問題がないことを示す良い兆候といえます。

5 一般的な質問

5.1 各 RFAL API に関する詳細な情報

RFAL のすべてのリリースに、すべての API に関する詳細情報が記載されたドキュメントが添付されています。それは、ドキュメント·フォルダ `doc/rfal.chm` にあります。ドキュメントは Doxygen によって生成されています。ソース·コードに Doxygen フォーマットでコメントが記述されています。これによって、各 API に関する詳細な情報が得られます。

5.2 RFAL ライブラリのフットプリントの計算

前述した RFAL パッケージのドキュメント·フォルダから `doc/rfal.chm` ファイルを入手できます。このファイルの中には、次のような、すべての RFAL オブジェクト / モジュールとそれらの要件が列挙されています。

テキスト	データ	bss	10 進数	16 進数	ファイル名
764	8	1024	1796	704	rfal_analogConfig.o
818	11	0	829	33d	rfal_cd.o
54	0	0	54	36	rfal_crc.o
26	0	0	26	1a	rfal_dpo.o
1004	12	0	1016	3f8	rfal_iso15693_2.o
5908	0	188	6096	17d0	rfal_isoDep.o
4150	1	2180	6331	18bb	rfal_nfc.o
1774	0	56	1830	726	rfal_nfca.o
986	2	0	988	3dc	rfal_nfcb.o
4884	0	176	5060	13c4	rfal_nfcDep.o
1150	0	322	1472	5c0	rfal_nfcf.o
2098	0	0	2098	832	rfal_nfcv.o
892	0	0	892	37c	rfal_st25tb.o
1346	0	0	1346	542	rfal_st25xv.o
302	0	0	302	12e	rfal_t1t.o
298	0	0	298	12a	rfal_t2t.o
672	0	0	672	2a0	rfal_t4t.o
27126	34	3946	31106	7982	(合計)

テキスト	データ	bss	10 進数	16 進数	ファイル名
ST25R3911B					
7648	0	1020	8668	21dc	rfal_rfst25r3911.o
1536	4	0	1540	604	st25r3911.o
1562	0	97	1659	67b	st25r3911_com.o
756	0	20	776	308	st25r3911_interrupt.o
11502	4	1137	12643	3163	(合計)

テキスト	データ	bss	10 進数	16 進数	ファイル名
ST25R3916					
9816	0	1028	10844	2a5c	rfal_rfst25r3916.o
1648	4	0	1652	674	st25r3916.o
748	0	0	748	2ec	st25r3916_aat.o
1140	2	513	1655	677	st25r3916_com.o
634	0	16	650	28a	st25r3916_irq.o
258	0	0	258	102	st25r3916_led.o
14244	6	1557	15807	3dbf	(合計)

テキスト	データ	bss	10 進数	16 進数	ファイル名
ST25R95					
3412	0	444	3856	f10	rfal_rfst25r95.o
258	0	0	258	102	st25r95.o
1082	162	0	1244	4dc	st25r95_com.o
542	21	0	563	233	st25r95_com_ce.o
1576	17	28	1621	655	st25r95_com_spi.o
6870	200	472	7542	1d76	(合計)

上記の値は、最適化を -O3 に設定して STM32L4 をコンパイルした結果に基づいています。

各モジュールは、すべての機能を含めてコンパイルしています。個々のモジュールの特定の機能を無効化すれば、ここに示した値よりもフットプリントが減少します(詳細は セクション 5.1 各 RFAL API に関する詳細な情報 を参照してください)。たとえば、リッスン・モードが不要で RFAL_FEATURE_ISO_DEP_LISTEN と RFAL_FEATURE_LISTEN_MODE を無効化すると、ISO-DEP と RF モジュールのフットプリントが小さくなります。

目的とするアプリケーションと必要なモードを把握すれば、この表を使ってフットプリントの概算値を計算できます。以下に、RFAL v2.2.0 に基づく ST25R3916 の例をいくつか示します。

表 4. NFC-A/NFC-B ポーラ(ISO144443A/B リーダ)

モジュール	Flash	RAM
AC テーブル	764	8
ISO-DEP	5908	188
NFC-A	1774	56
NFC-B	986	2
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	22670	1817
	22.14 K	1.77 K

表 5. NFC-V ポーラ(ISO15693 リーダ)

モジュール	Flash	RAM
AC テーブル	764	8
CRC	54	0
ISO15693-2	1004	12
NFC-V	2098	0
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	17158	1583
	16.76 K	1.55 K

表 6. NFC フォーラム・ユニバーサル・デバイス

モジュール	Flash	RAM
AC テーブル	764	8
CRC	54	0
ISO15693-2	1004	12
ISO-DEP	5908	188
NFC-A	1774	56
NFC-B	986	2
NFC-DEP	4884	176
NFC-F	1150	322
NFC-V	2098	0
T1T	302	0
T2T	298	0
T4T	672	0
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	33132	2327
	32.36 K	2.27 K

表 7. RFAL HL による NFC-A/NFC-B ポーラ(ISO144443A/B リーダ)

モジュール	Flash	RAM
AC テーブル	764	8
ISO-DEP	5908	188
NFC	4150	2181
NFC-A	1774	56
NFC-B	986	2
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	26820	3998
	26.19 K	3.90 K

表 8. NFC-A パッシブ・リスナ(CE)

モジュール	Flash	RAM
AC テーブル	764	8
ISO-DEP	5908	188
NFC-A	1774	56
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	21684	1815
	21.18 K	1.77 K

表 9. NFC-F パッシブ・リスナ(CE)

モジュール	Flash	RAM
AC テーブル	764	8
NFC-F	1150	322
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	15152	1893
	14.80 K	1.85 K

表 10. RFAL HL による NFC-A/NFC-B リスナ（ISO144443A/B CE）

モジュール	Flash	RAM
AC テーブル	764	8
ISO-DEP	5908	188
NFC	4150	2181
NFC-A	1774	56
NFC-F	1150	322
RFAL HAL	9816	1028
	1648	4
	1140	515
	634	16
合計	26984	4318
	26.35 K	4.22 K

表 11. RFAL の全機能

モジュール	Flash	RAM
AC テーブル	764	8
CD	818	11
CRC	54	0
DPO	26	0
ISO15693-2	1004	12
ISO-DEP	5908	188
NFC	4150	2181
NFC-A	1774	56
NFC-B	986	2
NFC-DEP	4884	176
NFC-F	1150	322
NFC-V	2098	0
ST25TB	892	0
ST25xV	1346	0
T1T	302	0
T2T	298	0
T4T	672	0
RFAL HAL	9816	1028
	1648	4
	748	0
	1140	515
	634	16
合計	41112	4519
	40.15 K	4.41 K

5.3 カスタム·アナログ設定テーブルの使用

RFAL には、サポート対象のすべてのデバイス向けにデフォルトのアナログ設定テーブルが用意されています。

カスタマイズされたユーザ定義のテーブルが必要な場合も、簡単に導入できます。

セクション 2.6.1 RF HAL を参照してください。

5.4 AM 変調指数の変更

変調タイプまたは変調指数はアナログ設定と見なされるため、アナログ設定テーブルで制御します。特定のテクノロジーまたはビットレートで変調指数を変更するには、その特定のモードで変調指数を適用 / 変更する必要があります。AC テーブルには、共通のエントリとビットレート固有のエントリがあり、セクション 2.6.1 の説明に従って適用されます。

どの設定を適用すべきかを見極めるにはデバイスのデータシートを参照してください。

5.5 AM/OOK 変調タイプの変更

セクション 5.4 と同様に、変調方式の設定もアナログ設定テーブルで制御します。特定のテクノロジーまたはビットレートで変調指数を適用 / 変更するには、その特定のモードで変調タイプ / 指数を適用 / 変更する必要があります。AC テーブルには、共通のエントリとビットレート固有のエントリがあり、セクション 2.6.1 の説明に従って適用されます。

どの設定を適用すべきかを見極めるにはデータシートを参照してください。

デバイスが、さまざまな変調方式(抵抗かレギュレータ、またはその両方による)を許容するということは、特定のアンテナ / ボードに対する正しい設定や、その設定を決定するための手法が、複数あることを意味します。

5.6 ユーザ定義データ型の使用方法

RFAL では C 言語の標準データ型を使用しますが、標準ライブラリ(`stdint`、`stdbool`、`stdlib` など)の使用を強制するものではありません。それらを直接インクルードしていないからです。外部ライブラリの依存関係は、ユーザ定義の `platform.h` にまとめて記述できるので、ユーザ固有のカスタマイズが可能です。ユーザ定義のデータ型が必要な場合は、C の標準ライブラリをインクルードせずに、これらの型をアプリケーション·レベルで定義するという方法も選べます。そのような設定の例は、STM8 のデモ(STSW-STM8-NFC5)で確認できます。

5.7 ブロッキング API と非ブロッキング API

RFAL のいくつかの API は、2 つの形式、ブロッキングと非ブロッキングで提供されます。ブロッキング·メソッドは簡単に使用でき、その結果、コードの可読性が高まり多くのアプリケーションに適しています。

実行する動作によっては、保全処理を必要とする他のペリフェラルが存在する場合など、一定期間、他の処理をブロックすることが許されません。

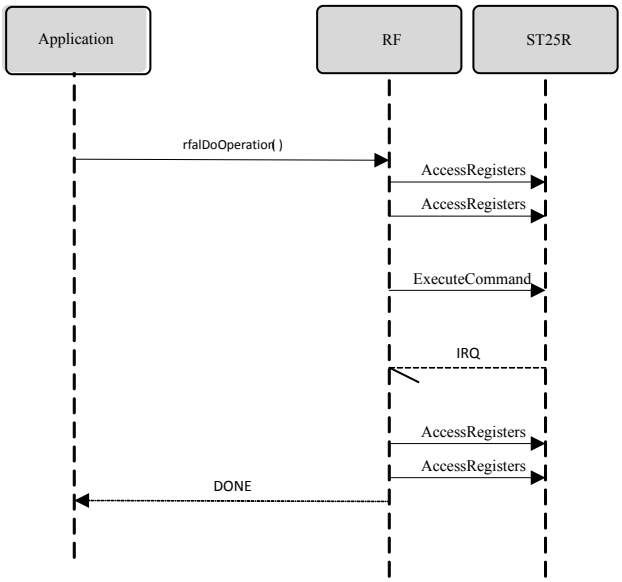
このため、多くの API は `rfalStartOperation()` や `rfalGetOperationStatus()` の形で非ブロッキング·インタフェースを提供しています。これを使用する場合、呼び出し元では、コンテキストを保持し、それらの動作のフローを制御するための、より複雑なステート·マシンを実装する必要があります。

セクション 2.6.4 で強調したように、遵守すべき 2 つの重要な側面があります。

- `rfalStartOperation()` による動作を開始したら、それが完了するまで等価の `rfalGetOperationStatus()` を呼び出し / ポーリングします。動作が完了するとは、`ERR_BUSY` 以外の値が返されることです。動作が完了して結果が返されたら、呼び出し元はただちに、その結果に応じた処理を実行する必要があります。この時点以降に `rfalGetOperationStatus()` を呼び出しても、前の結果やコンテキストの有効性が保持されるかどうかは保証されません(特定の例外を除く)。
- 非ブロッキング API は特定のプロシージャを実行し、開始と終了のイベントが定義されています。前の RFAL 動作の実行中は、別の RFAL 動作を開始することは禁止されています。起動すると、予期せぬ動作を引き起こし、状態の整合性が失われます。

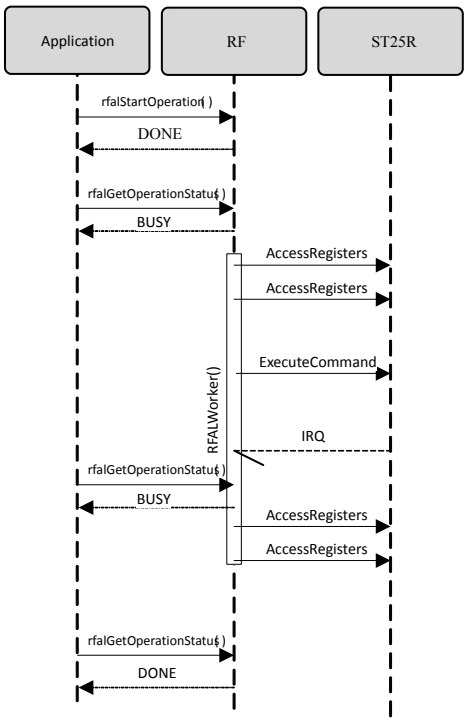
次のシーケンス図に、これら 2 つのインタフェースによる手法をわかりやすく示しました。ブロッキング·インタフェースによって API が呼び出されると、制御は呼び出された RFAL メソッドに渡され、このメソッドが決められたプロシージャを実行しますが、これには一定の時間がかかります。内部で経過するその時間は、外部のイベントや設定(シリアル·インタフェースの速度など)の影響を受けます。シーケンスが完了するまで、制御は呼び出し元 / アプリケーションに戻されません。ブロッキング API では、その動作を実行するために必要な場合は内部で `rfalWorker()` を呼び出す場合があることに注意してください。

図 9. ブロッキング API



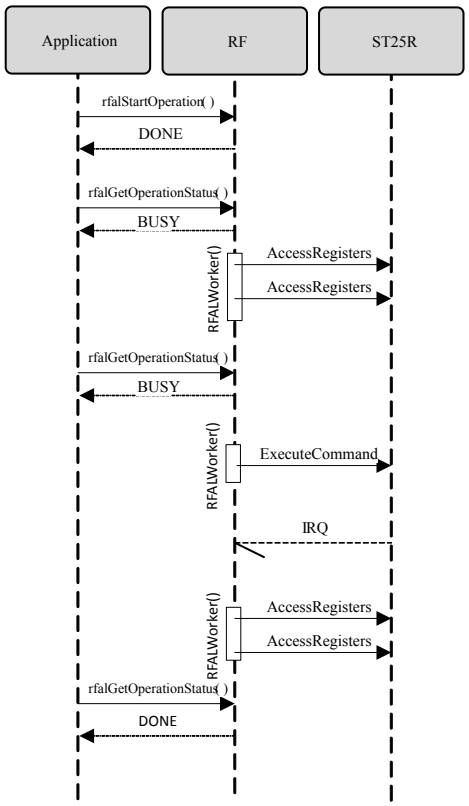
非ブロッキング API を使用する場合のフローは、まったく異なります。動作をトリガする呼び出しは、速やかに制御を呼び出し元に返し、その時点から動作が完了したかどうかのポーリングが開始されます。`rfalWorker()` は、自律的な動作により、要求されたタスクを実行できなければなりません。

図 10. 非ブロッキング API



シングル·スレッド環境では、これらは実際にはシーケンシャルに実行されます。アプリケーションが `rfalWorker()` を実行し、その動作完了を待機してポーリングするコンテキストに戻ります。下図は、実際の動作が `rfalWorker()` の実行中(当然 ISR による)にのみ、進行する様子を表しています。

図 11. 非ブロッキング API:rfalWorker() の詳細



注 RFAL を最終プロジェクト / 製品に組み込む際は、できるだけ非ブロッキング API を使用することを推奨します。ただし st.com で提供しているデモ·プロジェクトの中にはブロッキング API を使用しているものもあります。これは単純性とコードの可読性の向上、試作目的、または単一タスクのアプリケーションであることなどの要因に基づく措置です。ソフトウェア開発者は、そうした使用方法が悪影響を及ぼさないこと、目標とする製品やシステム·アーキテクチャで許容されることを評価する必要があります。

5.8 HW/SW 制御の SPI チップ選択(CS)

RFAL では、ソフトウェアとハードウェアで SPI チップ選択を制御できます。ソフトウェア制御の使用を推奨します。動作を細かく分割できることから、より高い柔軟性を確保でき、メモリ管理が容易になるとともに、初期化時に、定義済みの状態に強制的に移行できるからです。

ソフトウェアによる CS/SS(スレーブ選択)制御は、SPI HW ブロック / ペリフェラル自体による処理よりも遅い場合があり(使用するプラットフォームによる)、一部の固有アプリケーションでは最適化が必要です。ハードウェア制御の SPI CS/SS を使用するには、それに合わせて SPI ペリフェラルを設定するとともに `ST25R_COM_SINGLETXRX` を設定し、API の `platformSpiSelect()` と `platformSpiDeselect()` をクリアする必要があります。

注 ハードウェア制御のチップ選択を使用する場合、SPI HW ブロック / ペリフェラルの中には必要な動作をサポートできないものがあります。それらのペリフェラルは、定義されたビット数(16 ビットなど)を処理すると、自動的に CS/SS を選択 / 選択解除する場合があるからです。FIFO の書込み / 読出しなどでは、1 つの SPI 動作で任意のバイト数の送受信が必要です。

5.9 ST25R の ISR 内のループ

ST25R の割込みサービス・ルーチン（ISR）には、while ループが含まれます。これは、一見意外に思えるかも知れません。すべてのホストがレベルでトリガされる IRQ に対応しているわけではなく、エッジ・トリガの IRQ しか使用できないものがあるからです。

前のイベントの ISR 実行中に、新しい IRQ が発生すると問題が生じます。

図 12. エッジ・トリガの ISR：正常時

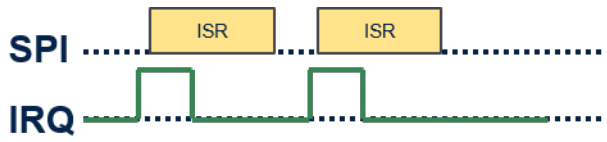


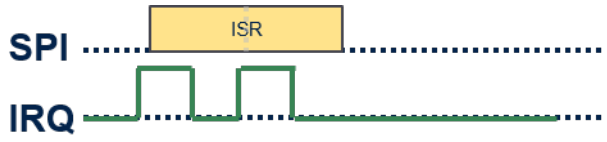
図 13. エッジ・トリガの ISR：問題あり

図 13 の場合、エッジ・トリガの ISR は、IRQ の発生間隔が十分に広く、その間に ISR の処理が完了していれば複数の IRQ を処理できます。

しかし、このような状況が保証されるわけではなく、ISR がまだ実行中で IRQ ステータス・レジスタから情報を取得している間に、次の IRQ イベントが発生する可能性があります。そのような場合、IRQ は High に戻ったままになります。エッジ / 遷移が再び検出されることがないことから、ISR が二度とトリガされなくなるからです。新しいイベントが処理されず、その後通知されるイベントも検出されないことから、ドライバは停止します。

これを避けるために、ISR プロシージャの最後に IRQ ピンを チェックし、未処理のイベントがないことを確認します。未処理のイベントが存在した場合は、再度プロシージャを実行して最新の情報を取得します。

図 14. エッジ・トリガの ISR：ループ



IRQ をレベル・トリガに設定すれば、そのようなチェック / ループは不要となり削除できます。そのような処理を行っても害はなく、相互運用性を確保するために RFAL ではレベル・トリガをデフォルトで提供しています。

5.10 デバッグ・ツール

何らかの問題が発生した場合、シリアル・インタフェース（SPI/I^2C/UART）のトレース情報を収集し、ホスト AFE のトラフィックを観測することが極めて有効です。ホスト AFE のトレース情報を収集する際は、IRQ やリセットなどの他の関連する信号も含める必要があります。このようなトレース情報が得られれば、問題の大部分を特定し対策でき、サポートも著しくシンプルになり迅速に提供できるようになります。トレース情報を得るにはロジック / デジタル・アナライザの使用を強く推奨します。ロジック・アナライザのほかに、NFC/RF スニッファによるトレースも極めて有用です。スニッファの欠点は、用途がかなり限定されることと、広く市販されているわけではない、比較的高価なツールであることです。

6 その他の参考文献

ID	参考文献
[1]	NFC Forum - Activity Technical Specification v2.1（NFC フォーラム - アクティビティ技術仕様 v2.1）
[2]	NFC Forum - Digital Technical Specification v2.2（NFC フォーラム - デジタル技術仕様 v2.2）
[3]	NFC Forum – NFC Controller Interface Technical Specification v2.1（NFC フォーラム - NFC コントローラ･インタフェース技術仕様 v2.1）
[4]	EMV Contactless Interface Specification v3.1（EMV 非接触インタフェース仕様 v3.1）

付録 A platform.h の例

```
/*Example of system platform definitions for a STM32 project*/
#ifndef PLATFORM_H
#define PLATFORM_H
/*
*********************************************************************************
* INCLUDES
*********************************************************************************
*/
#include <stdint.h>                                                    /* Include type
definitions                                */
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>
#include "stm32l4xx_hal.h"                                              /* Include the
platform HAL                                */
#include "spi.h"                                                        /* Include any
glue layer needed for interfacing with HAL */
#include "timer.h"
#include "main.h"                                                       /* Include PIN
definitions (STM32CubeMx places in main.h) */
/*
*********************************************************************************
* GLOBAL DEFINES
*********************************************************************************
*/

#define ST25R_COM_SINGLETXRX                                             /*!< Use single
Transceive                                */

#define ST25R_SS_PIN                SPI1_CS_Pin                 /*!< GPIO pin used
for ST25R     SPI SS                */
#define ST25R_SS_PORT                 SPI1_CS_GPIO_Port           /*!< GPIO port
used for ST25R     SPI SS port          */

#define ST25R_INT_PIN               IRQ_391x_Pin                /*!< GPIO pin used
for ST25R     External Interrupt    */
#define ST25R_INT_PORT                IRQ_391x_GPIO_Port          /*!< GPIO port
used for ST25R     External Interrupt   */

#ifdef LED_FIELD_Pin
#define PLATFORM_LED_FIELD_PIN      LED_FIELD_Pin               /*!< GPIO pin used
as field LED                          */
#endif

#ifdef LED_FIELD_GPIO_Port
#define  PLATFORM_LED_FIELD_PORT      LED_FIELD_GPIO_Port          /*!< GPIO port
used as field LED                       */
#endif

#ifdef LED_RX_Pin
#define PLATFORM_LED_RX_PIN         LED_RX_Pin                  /*!< GPIO pin used
as field LED                          */
#endif

#ifdef LED_RX_GPIO_Port
#define  PLATFORM_LED_RX_PORT          LED_RX_GPIO_Port             /*!< GPIO port
used as field LED                       */
#endif
```

```
/*
*********************************************************************************
* GLOBAL MACROS
*********************************************************************************
*/
#define platformProtectST25RComm()                    do{ globalCommProtectCnt++;
__DSB();NVIC_DisableIRQ(EXTI0_IRQn);__DSB();__ISB();}while(0) /*!< Protect unique
access to ST25R communication channel - IRQ disable on single thread environment
(MCU) ; Mutex lock on a multi thread environment      */
#define  platformUnprotectST25RComm()                                  do{  if  (--
globalCommProtectCnt==0)
{NVIC_EnableIRQ(EXTI0_IRQn);} }while(0)                  /*!< Unprotect unique
access to ST25R communication channel - IRQ enable on a single thread environment
(MCU) ; Mutex unlock on a multi thread environment */

#define   platformProtectST25RIrqStatus()
platformProtectST25RComm()                                    /*!< Protect unique
access to IRQ status var - IRQ disable on single thread environment (MCU) ; Mutex
lock on a multi thread environment */
#define   platformUnprotectST25RIrqStatus()
platformUnprotectST25RComm()                                 /*!< Unprotect the IRQ
status var - IRQ enable on a single thread environment (MCU) ; Mutex unlock on a
multi thread environment         */


#define platformLedOff( port, pin )                    platformGpioClear((port),
(pin))                          /*!< Turns the given LED Off                    */
#define platformLedOn( port, pin )                      platformGpioSet((port),
(pin))                            /*!< Turns the given LED On
*/
#define platformLedToogle( port, pin )                 platformGpioToogle((port),
(pin))                        /*!< Toogle the given LED                        */

#define platformGpioSet( port, pin )                    HAL_GPIO_WritePin(port,
pin, GPIO_PIN_SET)               /*!< Turns the given GPIO High
*/
#define platformGpioClear( port, pin )                  HAL_GPIO_WritePin(port,
pin, GPIO_PIN_RESET)             /*!< Turns the given GPIO Low
*/
#define platformGpioToogle( port, pin )                HAL_GPIO_TogglePin(port,
pin)                            /*!< Toogles the given GPIO
*/
#define platformGpioIsHigh( port, pin )                 (HAL_GPIO_ReadPin(port,
pin) == GPIO_PIN_SET)            /*!< Checks if the given LED is High
*/
#define platformGpioIsLow( port, pin )                (!platformGpioIsHigh(port,
pin))                         /*!< Checks if the given LED is Low             */

#define  platformTimerCreate(  t  )
timerCalculateTimer(t)                                  /*!< Create a timer with
the given time (ms)     */
#define   platformTimerIsExpired(   timer   )
timerIsExpired(timer)                                   /*!< Checks if the given
timer is expired        */
#define  platformDelay(  t  )
HAL_Delay( t )                                            /*!< Performs a delay
for the given time (ms)    */

#define  platformGetSysTick()
HAL_GetTick()                                           /*!< Get System Tick (1
tick = 1 ms)             */
```

```
#define  platformAssert(  exp  )
assert_param( exp )                                    /*!< Asserts whether the
given expression is true*/
#define platformErrorHandle()                          _Error_Handler(__FILE__,
__LINE__)                        /*!< Global error handle\trap
*/

#define  platformSpiSelect()
platformGpioClear( ST25R_SS_PORT, ST25R_SS_PIN )           /*!< SPI SS\CS: Chip|
Slave Select                */
#define  platformSpiDeselect()
platformGpioSet( ST25R_SS_PORT, ST25R_SS_PIN )             /*!< SPI SS\CS: Chip|
Slave Deselect              */
#define platformSpiTxRx( txBuf, rxBuf, len )              spiTxRx(txBuf, rxBuf,
len)                                                              /*!<  SPI
transceive                                 */


#define         platformI2CTx(         txBuf,         len,         last,
txOnly )                                                          /*!< I2C
Transmit                                   */
#define                  platformI2CRx(                  txBuf,
len )
/*!< I2C Receive                                    */
#define
platformI2CStart()
*!< I2C Start condition                           */
#define
platformI2CStop()
*!< I2C Stop condition                            */
#define
platformI2CRepeatStart()
*!< I2C Repeat Start                              */
#define
platformI2CSlaveAddrWR(add)
*!< I2C Slave address for Write operation         */
#define
platformI2CSlaveAddrRD(add)
*!< I2C Slave address for Read operation          */

#define
platformLog(...)
*!< Log  method                                   */

/*
******************************************************************************
* GLOBAL VARIABLES
******************************************************************************
*/
extern uint8_t globalCommProtectCnt;                        /* Global Protection
Counter provided per platform - instantiated in main.c    */

/*
******************************************************************************
* RFAL FEATURES CONFIGURATION
******************************************************************************
*/

#define RFAL_FEATURE_LISTEN_MODE                  true        /*!< Enable/Disable
RFAL support for Listen Mode                          */
#define RFAL_FEATURE_WAKEUP_MODE                  true        /*!< Enable/Disable
```

```
RFAL support for the Wake-Up mode                                  */
#define RFAL_FEATURE_LOWPOWER_MODE                true         /*!< Enable/Disable
RFAL support for the Low Power mode                                */
#define RFAL_FEATURE_NFCA                         true         /*!< Enable/Disable
RFAL support for NFC-A (ISO14443A)                                 */
#define RFAL_FEATURE_NFCB                         true         /*!< Enable/Disable
RFAL support for NFC-B (ISO14443B)                                 */
#define RFAL_FEATURE_NFCF                         true         /*!< Enable/Disable
RFAL support for NFC-F (FeliCa)                                    */
#define RFAL_FEATURE_NFCV                         true         /*!< Enable/Disable
RFAL support for NFC-V (ISO15693)                                  */
#define RFAL_FEATURE_T1T                          true         /*!< Enable/Disable
RFAL support for T1T (Topaz)                                       */
#define RFAL_FEATURE_T2T                          true         /*!< Enable/Disable
RFAL support for T2T (MIFARE Ultralight)                           */
#define RFAL_FEATURE_T4T                          true         /*!< Enable/Disable
RFAL support for T4T                                               */
#define RFAL_FEATURE_ST25TB                       true         /*!< Enable/Disable
RFAL support for ST25TB                                            */
#define RFAL_FEATURE_ST25xV                       true         /*!< Enable/Disable
RFAL support for ST25TV/ST25DV                                     */
#define RFAL_FEATURE_DYNAMIC_ANALOG_CONFIG       false        /*!< Enable/Disable
Analog Configs to be dynamically updated (RAM)                     */
#define RFAL_FEATURE_DPO                          false        /*!< Enable/Disable
RFAL Dynamic Power Output upport                                   */
#define RFAL_FEATURE_ISO_DEP                      true         /*!< Enable/Disable
RFAL support for ISO-DEP (ISO14443-4)                              */
#define RFAL_FEATURE_ISO_DEP_POLL                 true         /*!< Enable/Disable
RFAL support for Poller mode (PCD) ISO-DEP (ISO14443-4)            */
#define RFAL_FEATURE_ISO_DEP_LISTEN               true         /*!< Enable/Disable
RFAL support for Listen mode (PICC) ISO-DEP (ISO14443-4)           */
#define RFAL_FEATURE_NFC_DEP                      true         /*!< Enable/Disable
RFAL support for NFC-DEP (NFCIP1/P2P)                              */


#define RFAL_FEATURE_ISO_DEP_IBLOCK_MAX_LEN     256U         /*!< ISO-DEP I-Block
max length. Please use values as defined by rfalIsoDepFSx */
#define RFAL_FEATURE_NFC_DEP_BLOCK_MAX_LEN       254U          /*!< NFC-DEP Block/
Payload length. Allowed values: 64, 128, 192, 254           */
#define RFAL_FEATURE_NFC_RF_BUF_LEN              256U        /*!< RF buffer length
used by RFAL NFC layer                                     */

#define RFAL_FEATURE_ISO_DEP_APDU_MAX_LEN       512U         /*!< ISO-DEP APDU max
length.                                                    */
#define RFAL_FEATURE_NFC_DEP_PDU_MAX_LEN         512U         /*!< NFC-DEP PDU max
length.                                                     */




/*
 ******************************************************************************
 * RFAL OPTIONAL MACROS            (Do not change)
 ******************************************************************************
 */

#ifndef platformProtectST25RIrqStatus
#define platformProtectST25RIrqStatus()             /*!< Protect unique access to
IRQ status var - IRQ disable on single thread environment (MCU) ; Mutex lock on a
multi thread environment */
#endif /* platformProtectST25RIrqStatus */
```

```
#ifndef platformUnprotectST25RIrqStatus
#define platformUnprotectST25RIrqStatus()           /*!< Unprotect the IRQ status
var - IRQ enable on a single thread environment (MCU) ; Mutex unlock on a multi
thread environment         */
#endif /* platformUnprotectST25RIrqStatus */

#ifndef platformProtectWorker
#define platformProtectWorker()                          /* Protect RFAL Worker/Task/
Process from concurrent execution on multi thread platforms   */
#endif /* platformProtectWorker */

#ifndef platformUnprotectWorker
#define platformUnprotectWorker()                      /* Unprotect RFAL Worker/Task/
Process from concurrent execution on multi thread platforms */
#endif /* platformUnprotectWorker */

#ifndef platformIrqST25RPinInitialize
#define platformIrqST25RPinInitialize()                /*!< Initializes ST25R IRQ
pin                    */
#endif /* platformIrqST25RPinInitialize */

#ifndef platformIrqST25RSetCallback
#define  platformIrqST25RSetCallback( cb )                 /*!< Sets ST25R ISR
callback                        */
#endif /* platformIrqST25RSetCallback */

#ifndef platformLedsInitialize
#define platformLedsInitialize()                     /*!< Initializes the pins used
as LEDs to outputs */
#endif /* platformLedsInitialize */

#ifndef platformLedOff
#define platformLedOff( port, pin )                      /*!< Turns the given LED
Off                      */
#endif                       /*                     platformLedOff
*/

#ifndef
platformLedOn
#define platformLedOn( port, pin )                       /*!< Turns the given LED
On                       */
#endif                       /*                     platformLedOn
*/

#ifndef
platformLedToogle
#define  platformLedToogle( port, pin )                   /*!< Toggles the given
LED                        */
#endif                     /*                    platformLedToogle
*/

#ifndef
platformGetSysTick
#define platformGetSysTick()                         /*!< Get System Tick (1 tick =
1 ms)              */
#endif                     /*                    platformGetSysTick
*/

#ifndef
platformTimerDestroy
#define platformTimerDestroy( timer )                 /*!< Stops and released the
```

```
given timer           */
#endif /* platformTimerDestroy */

#ifndef
platformAssert
#define platformAssert( exp )                        /*!< Asserts whether the given
expression is true */
#endif /* platformAssert */

#ifndef
platformErrorHandle
#define platformErrorHandle()                         /*!< Global error handler or
trap                 */
#endif /* platformErrorHandle */


#endif /* PLATFORM_H */
```

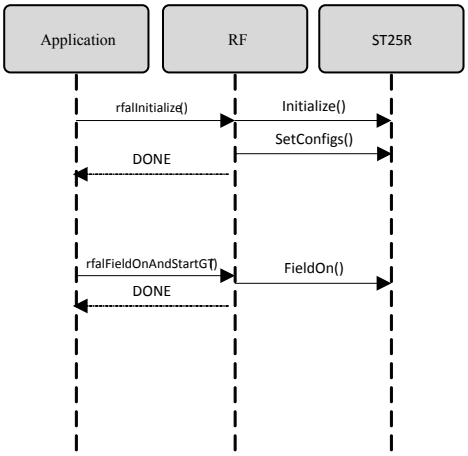
付録 B　　シーケンス図

このセクションでは、各種モジュールが特定のタスクで果たす役割を図解する、一連のシーケンス図を示します。

シンプルで見やすい図を提供するため、および一部の動作はデバイス固有であることから、具体的なイベント（個々の割込み、レジスタ·アクセスなど）のすべてが厳密に示されているわけではありませんが、実行される操作は表現されています。また、一部の API はフルネームが長すぎるため、短縮して表示している場合があります。たとえば、`rfalNfcaPollerFullCollisionResolution()` → `...NfcaFullCollisionResolution()` 、`rfalIsoDepStartApduTransceive()` → `...StartApduTransceive()` などです。

RFAL の初期化と RF キャリア·オン

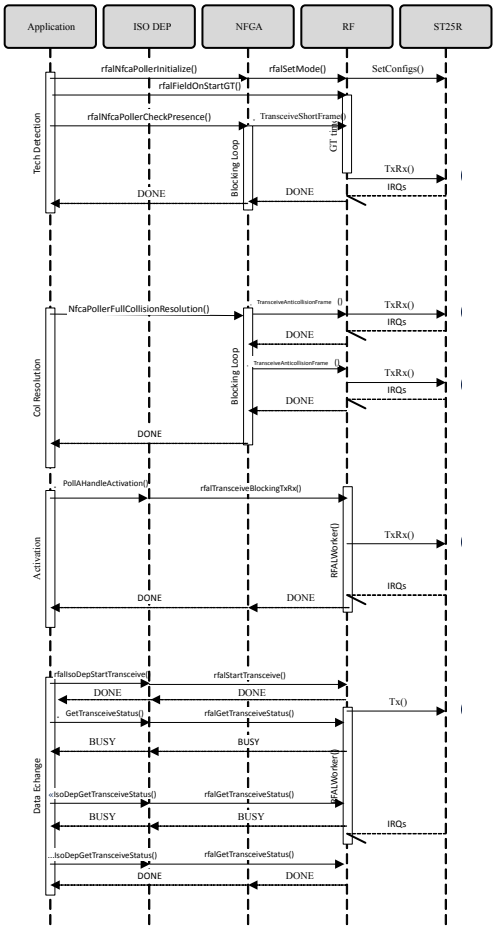
図 15. **RFAL の初期化と RF キャリア·オン**



新しいハードウェア構成の基本機能を確認するときに、しばしば使用されるこの簡単な例では、アプリケーション·レイヤが RFAL を初期化し、今度はその RFAL がリセット·シーケンスの実行、初期設定の読込み、定義された状態への設定を行うことで NFC AFE を初期化します。その後、`rfalFieldOnAndStartGT()` を呼び出して、RF キャリアのフィールドをオンにします。

NFC-A/ISO14443A ブロッキングのアクティブ化とデータ交換

図 16. NFC-A/ISO14443A ブロッキングのアクティブ化とデータ交換



ここでは、アプリケーションが NFC-A T4T デバイスをアクティブ化し、データを交換しています。アプリケーションは、NFC-A と ISO-DEP モジュールで提供される API を直接使用し、カードのアクティブ化処理の全期間をブロッキング API で実装しています。NFC-A 通信の初期化から始めて、フィールド（RF キャリア）をオンにすると、設定済みの GT タイマがカウントを開始します。プレゼンス･チェック / テクノロジー検出を実行したら、GT の要件が満たされるのを待ちます（まだタイムアウトしていない場合）。要件が満たされたらポーリング･コマンド（SENS_REQ/REQA）を送信します。カード（複数可）が検出されると、アプリケーションは `rfalNfcaPollerFullCollisionResolution()` を呼び出してコリジョン解決に移行します。これは、いくつかのコリジョン防止コマンドを交換するブロッキング API です（実際に交換されるコマンド数は、存在するカードの数やそれらの NFCID など、複数の要因で決まります）。

すべてのリスナ / カードが識別されると、アプリケーションは選択するカードを 1 つアクティブ化でき（複数アクティブ化する場合は、この図には示していません）、サポートされているプロトコルのいずれか 1 つ（この例では ISO-DEP）をアクティブ化する必要があります。ISO14443-4 でカードをアクティブ化するために、API の `rfalIsoDepPollAHandleActivation()` を使用しています。この API は全レイヤを 4 つのアクティブ化プロシージャで処理します。この方法は、リスナ･デバイスとの間で 1 つまたは複数のコマンドを交換します。強調しておくべき重要な点は、この処理の実行中はより長いタイムアウト /FWT（~ 5 ms）が適用され、アクティブ化全体にかかる時間が合計で数ミリ秒に達する場合もあるということです。アクティブ化が完了したら、ISO-DEP プロトコルで通常のデータ交換フローが実行されます。これを実現するために、アプリケーションは ISO-DEP のトランシーバ動作を実行します（ブロックまたは APDU）。この動作は内部でプロトコルを処理し、実際の送受信には下位の RF トランシーバ･プロセスをトリガします。

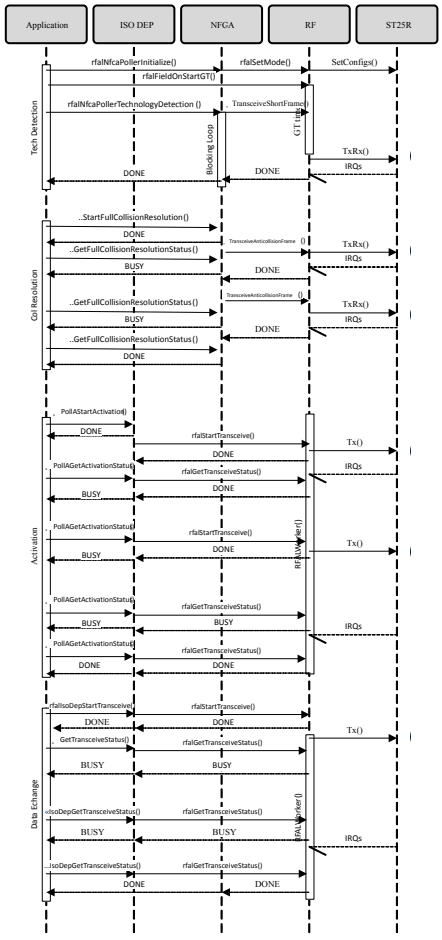
プロトコルのデータ交換(ISO-DEP と NFC-DEP)では FWT が極めて長くなる(最大 5 秒)可能性があり、待機時間の延長や繰り返しと重なれば、1 つのプロトコル·トランシーバ動作に数秒かかる場合もあります。この理由から、こうした長い実行時間が問題にならない特別なアプリケーション·フローの場合を除き、RFAL はブロッキング·プロトコルの実行機能を提供しません(ブロッキング動作は推奨されません)。上記のシーケンスの特筆すべき点は、初期の `rfalIsoDepStartTransceive()` の後、トランシーバ動作が完了する(`ERR_BUSY` 以外の戻り値)まで、`rfalIsoDepGetTransceiveStatus()` メソッドでポーリングする必要があることです。実際の送受信は、RF モジュールが `rfalWorker()` を使用して並列処理し、完了すると `rfalIsoDepGetTransceiveStatus()` が再度呼び出されるまで、結果が保存されます。

重要

- `rfalGetOperationStatus()` の呼び出し / ポーリングは、動作が完了するまで、つまり戻り値として `ERR_BUSY` 以外が返されるまで続ける必要があります。動作が完了して結果が返されたら、呼び出し元はただちに、その結果に応じた処理を実行する必要があります。この時点以降に `rfalGetOperationStatus()` を呼び出しても、前の結果やコンテキストの有効性が保持されるかどうかは保証されません(特定の例外を除く)。
- 非ブロッキング API は、特定のプロシージャを実行し、開始と終了のイベントが定義されています。前の RFAL 動作の実行中は、別の RFAL 動作を開始することは禁止されています。起動すると、予期せぬ動作を引き起こし、状態の整合性が失われる恐れがあります。

NFC-A/ISO14443A のアクティブ化とデータ交換

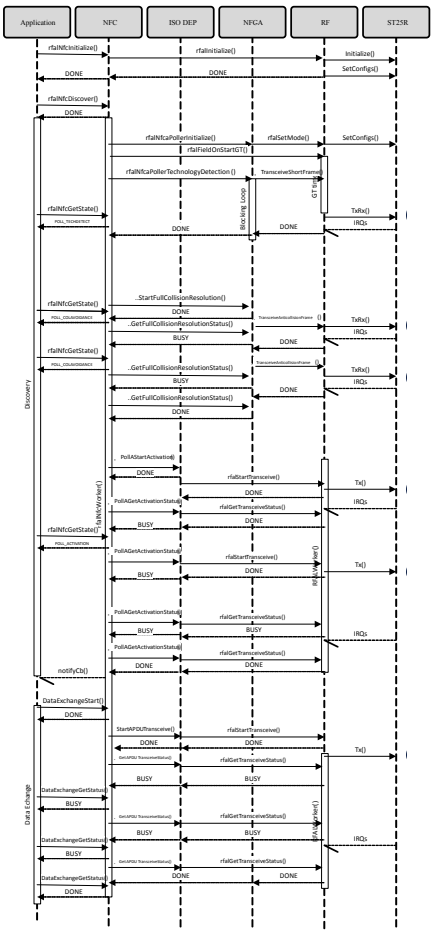
図 17. NFC-A/ISO14443A のアクティブ化とデータ交換



ここでは、前セクションと同じカードのアクティブ化およびデータ交換を実現しています（前セクションの説明を参照してください）。主な違いは、アクティブ化のプロセスで、ブロッキングではなく非ブロッキング API を使用している点です。通常どおり、非ブロッキング API が、完了するまでポーリングが必要なプロシージャを起動します。今回は、そこにコリジョン解決とプロトコルのアクティブ化のプロセスも含まれます。前セクションの例との、もう一つの軽微な違いは、プロトコルのアクティブ化において実際には 2 つのフレーム（例：RATS と PPS）が交換されていることです。これは、プロトコルのアクティブ化が、1 回のプロトコル·アクティブ化動作を実行する間に、1 回以上の RF トランシーバ動作をトリガする可能性があることを意味します。

NFC-A/ISO14443A のアクティブ化と RFAL HL による データ交換

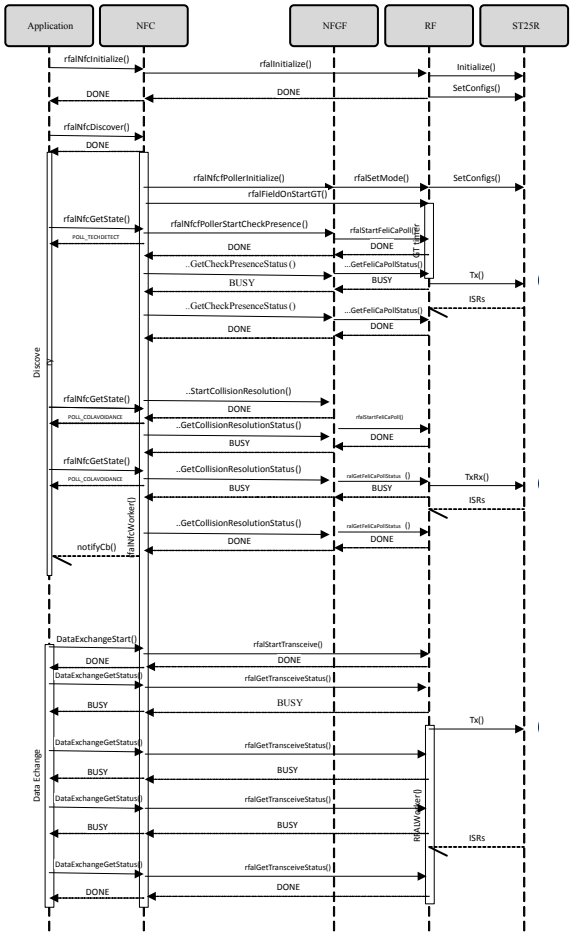
図 18. NFC-A/ISO14443A のアクティブ化と RFAL HL による データ交換



ここでも再度、前セクションと同じカードのアクティブ化およびデータ交換を実現しています（前セクションの説明を参照してください）。主な違いは、RFAL HL 上の NFC モジュールを使用している点です。このモジュールは、さまざまな動作 / プロシージャを、最小限のユーザ / アプリケーションの介入で、便利に処理してくれます。ユーザは単に、RFAL を初期化し、ディスカバリ·プロセスを開始し、デバイスがアクティブ化されたら目的のデータを交換するだけです。リスナ / カードのディスカバリ·プロセス（テクノロジー検出、コリジョン解決、アクティブ化）は、すべて内部で処理され、アプリケーションは `rfalNfcGetState()` によってポーリングするか、オプションの通知コールバック（notifyCb）を登録することで現在の状態を監視します。注目すべきイベント / 状態に至ると、NFC モジュールが通知します。前の例と同様に、リスナのアクティブ化が完了すると、トランシーバ動作によってデータ交換が実行されます。今回の例では RFAL HL を使用しており、NFC モジュールを `rfalNfcDataExchangeStart()` と `GetDataExchangeStatus()` のペアで実現できます。

NFC-V/FeliCa のアクティブ化と RFAL HL による データ交換

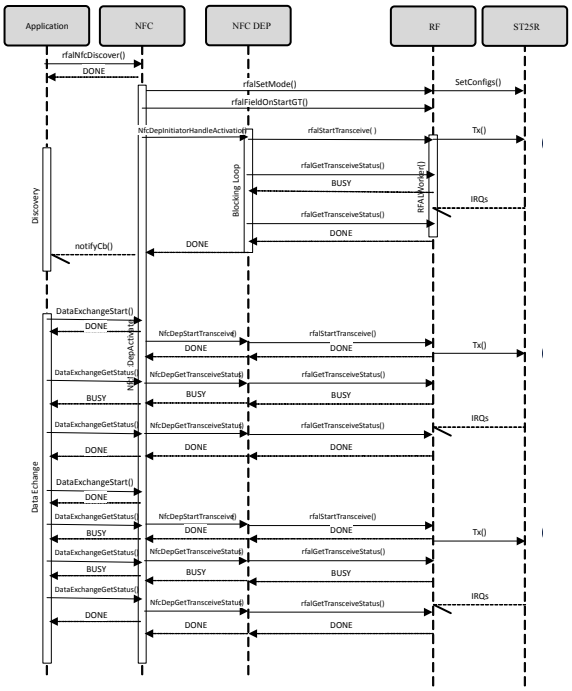
図 19. NFC-V/FeliCa のアクティブ化と RFAL HL による データ交換



この例でも、アクティブ化とデータ交換のシーケンスを示します。主な違いは、リスナが NFC-F T3T であるため、データ交換プロトコルが使われない点です。このため、プロトコルのアクティブ化は実行されず、単に RF インタフェースを使用してデータを交換します。NFC-V T5T を扱う場合も、類似のフローが実行されます。

AP2P イニシエータのアクティブ化と RFAL HL によるデータ交換

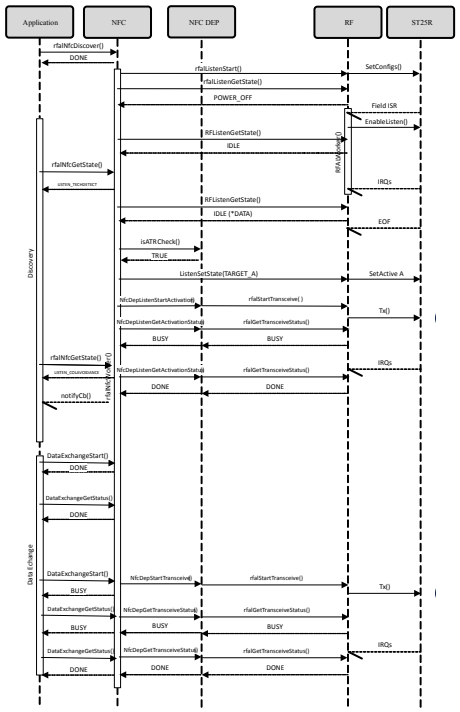
図 20. AP2P イニシエータのアクティブ化と RFAL HL によるデータ交換



AP2P ではコリジョン解決フェーズが存在しないため（RF コリジョン防止プロシージャを除く）、イニシエータがプロトコル・アクティブ化コマンド（ATR_REQ）を送信します。アクティブ化が完了すると、イニシエータはデータ交換に移行し、一連のトランシーバ動作を実行します。

AP2P ターゲットのアクティブ化と RFAL HL によるデータ交換

図 21. AP2P ターゲットのアクティブ化と RFAL HL によるデータ交換

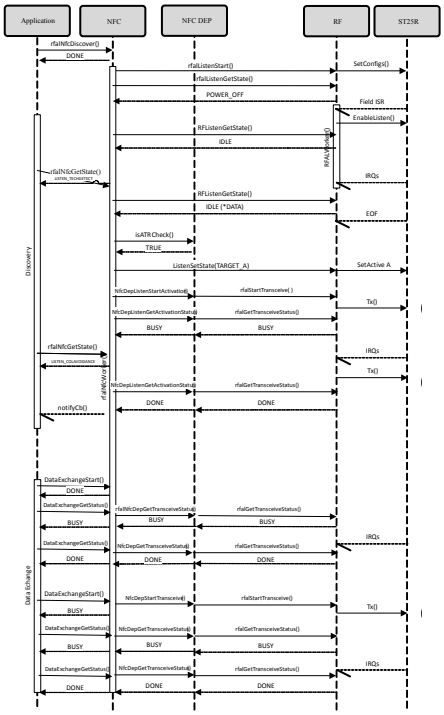


この図では、リッスン・モードにおけるアクティブ化とそれに続くデータ交換のシーケンスを確認できます。前セクションと同様に、AP2P ではコリジョン解決フェーズが存在しないため（RF コリジョン防止プロシージャを除く）、イニシエータはすぐにプロトコル・アクティブ化コマンド（ATR_REQ）の送信段階に進みます。リスナが、このフレームを受信すると、RF レイヤの汎用リッスン・モード・ハンドラが受信したペイロードを出力します。続いて、このペイロードがプロトコル・レイヤ（この例では NFC-DEP）によって検証され、予想どおりの結果が得られた場合、プロトコル・アクティブ化ハンドラが実行され、受信した ATR_REQ に応答します。受信したフレームが既にデータ・パケットであった場合（パラメータ変更 PSL_REQ がない場合）、アクティブ化は中止され、データ交換に移行します。

強調しておくべき重要な点は、リッスン・モードでアクティブ化中にデータ・パケットが受信された場合、そのデータは、データ交換 / トランシーバ動作メソッドが呼び出されて、はじめて取得されるということです。上記のシーケンスを見ると、1 回目の rfalNfcDataExchangeStart() / rfalNfcDataExchangeGetStatus() 呼び出しでは、実際の交換は行われていないことがわかります。かわりに、ポーラ / イニシエータから受信したデータは、処理のためにただちに呼び出し元に渡されます。

AP2P ターゲットのアクティブ化（PSL_REQ を含む）と RFAL HL によるデータ交換

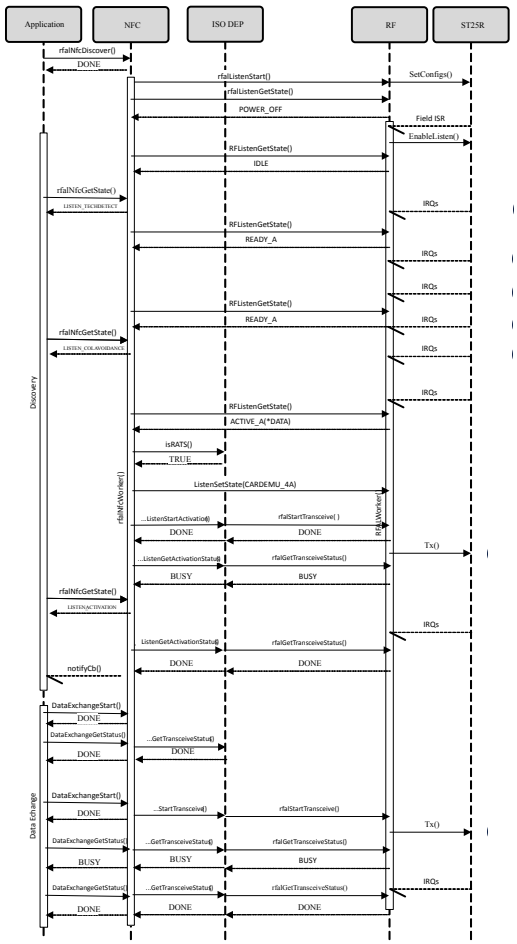
図 22. 追加フレーム（PSL_REQ）



この例は、アクティブ化中にフレーム（PSL_REQ）が 1 つ追加されている点を除き、前のセクションと同じです。PSL を使用すると、それ以降の通信のパラメータをいくつか変更でき、アクティブ化プロシージャの完了が確認された時点でデータ交換に移行します。したがって、このシーケンスでは、最初に `rfalNfcDataExchangeStart()` / `rfalNfcDataExchangeGetStatus()` が呼び出されたときにデータがすぐに取得できる状態ではないため、異なるフローになります。

NFC-A/ISO14443 リスナのアクティブ化と RFAL HL による データ交換

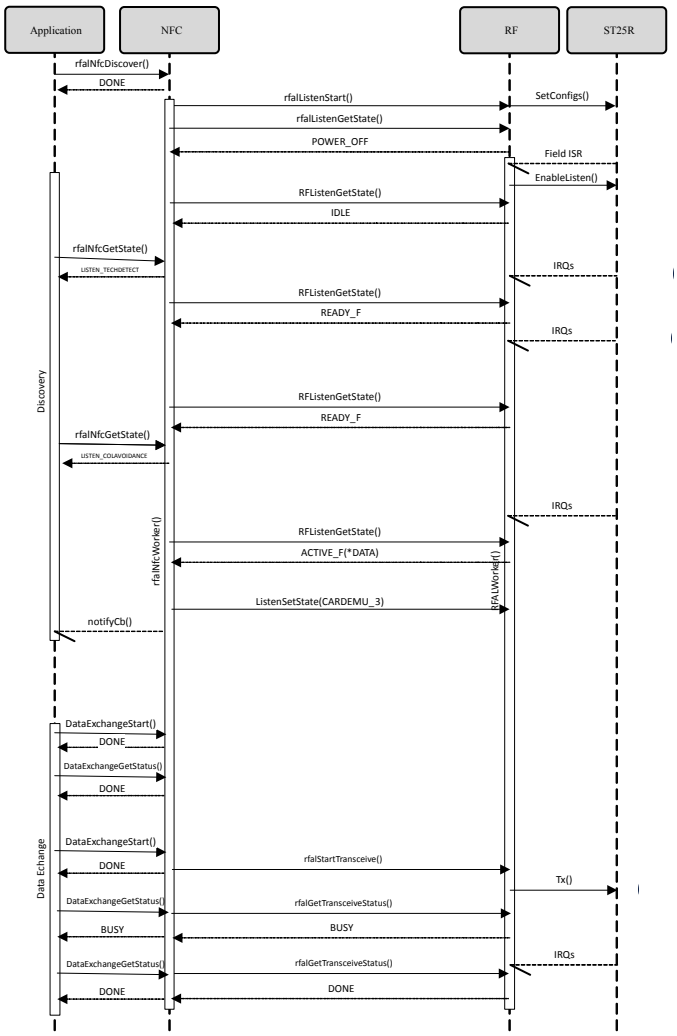
図 23. NFC-A/ISO14443 リスナのアクティブ化と RFAL HL による データ交換



この例では、ST25R が T4T CE モードのリスナ·デバイスとしてアクティブ化されます。リッスン·モードが開始されると、コリジョン防止シーケンスが実行され、デバイスが選択されると ACTIVE_A の状態に移行します。続くコマンド（RATS）を受信すると、ペイロードがプロトコル·レイヤに渡され、そこで有効性（ISO-DEP）が確認されます。プロトコルのアクティブ化は、RATS に反応して開始されます。その後、データ·パケットを受信し、プロトコルのアクティブ化が完了します（PPS なし）。前と同様に、受信データは最初のデータ交換呼び出しで渡され、アプリケーションによって処理されます。その後、他のデータ交換も通常どおり実行できるようになります。

NFC-F/FeliCa リスナのアクティブ化と RFAL HL による データ交換

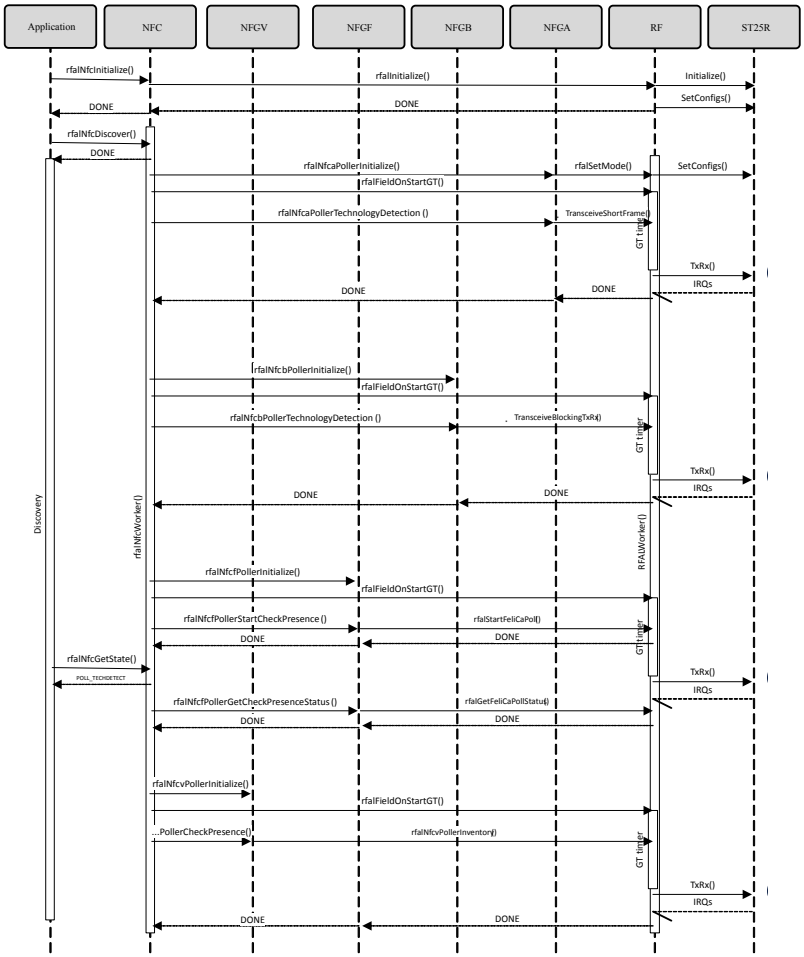
図 24. NFC-F/FeliCa リスナのアクティブ化と RFAL HL による データ交換



この例では、ST25R が T3T CE モードのリスナ·デバイスとしてアクティブ化されます。前の T4T フローと同じですが、T3T は特別なデータ交換プロトコルを必要としないため、プロトコルのアクティブ化は実行されません。

RFAL HL によるポーリング・サイクル

図 25. RFAL HL によるポーリング・サイクル



ここでは、NFC-A、NFC-B、NFC-F、NFC-V に対するポーリングを実行するように NFC モジュールを設定しています。この例ではカードが存在しないことから、さまざまなテクノロジーがポーリングされた後、タイムアウト・エラーが発生しています。すべてのテクノロジーのポーリングが実行されると、リクエストの合計時間が経過するまで待機し（まだ経過していない場合）、その後、同様のポーリングを再度実行します。

改訂履歴

表 12. 文書改版履歴

日付	版	変更内容
2021 年 10 月 21 日	1	初版発行

目次

1 略語 2
2 RFAL ライブラリ 4
2.1 機能の概要 4
2.2 説明 4
2.3 コーディングルールと規則 5
2.3.1 コーディング規則 5
2.3.2 ランタイム・チェック 5
2.3.3 MISRA-C:2012 準拠 5
2.3.4 NFC 用語 5
2.4 ハードウェア 6
2.4.1 要件 6
2.4.2 サポート対象のホスト・デバイス 6
2.5 ソフトウェア 6
2.5.1 依存関係 6
2.5.2 サポート対象の開発ツール 7
2.6 アーキテクチャ 7
2.6.1 RF HAL 7
2.6.2 RF AL 12
2.6.3 RF HL 14
2.6.4 概要 16
2.6.5 同時使用 17
3 リリース 19
3.1 パッケージの説明 19
3.2 品質基準 19
4 RFAL ライブラリの使用方法 20
4.1 ペリフェラルの初期化と設定 20
4.2 ライブラリの設定 22
4.3 初回のサンプル実行手順 23
5 一般的な質問 24
5.1 各 RFAL API に関する詳細な情報 24
5.2 RFAL ライブラリのフットプリントの計算 24
5.3 カスタム・アナログ設定テーブルの使用 29
5.4 AM 変調指数の変更 29
5.5 AM/OOK 変調タイプの変更 29
5.6 ユーザ定義データ型の使用方法 29

5.7 ブロッキング API と非ブロッキング API 29
5.8 HW/SW 制御の SPI チップ選択（CS） 31
5.9 ST25R の ISR 内のループ 32
5.10 デバッグ・ツール 32
6 その他の参考文献 33
付 録 Aplatform.h の例 34
付 録 Bシーケンス図 40
改訂履歴 51
表一覧 54
図一覧 55

表一覧

表 1. 略語 . . . 2
表 2. 該当製品 . . . 4
表 3. サポート対象モード(x = サポート) . . . 4
表 4. NFC-A/NFC-B ポーラ(ISO144443A/B リーダ) . . . 25
表 5. NFC-V ポーラ(ISO15693 リーダ) . . . 26
表 6. NFC フォーラム·ユニバーサル·デバイス . . . 26
表 7. RFAL HL による NFC-A/NFC-B ポーラ(ISO144443A/B リーダ) . . . 27
表 8. NFC-A パッシブ·リスナ(CE) . . . 27
表 9. NFC-F パッシブ·リスナ(CE) . . . 27
表 10. RFAL HL による NFC-A/NFC-B リスナ(ISO144443A/B CE) . . . 28
表 11. RFAL の全機能 . . . 28
表 12. 文書改版履歴 . . . 51

図一覧

図 1. RFAL スタックのアーキテクチャ . . . 7
図 2. アナログ設定シーケンス . . . 9
図 3. アナログ設定タブ . . . 11
図 4. NFC モジュールの状態図 . . . 15
図 5. システムの概要 . . . 16
図 6. 保護なしの同時実行 . . . 17
図 7. 保護ありの同時実行 . . . 18
図 8. RFAL パッケージの構成 . . . 19
図 9. ブロッキング API . . . 30
図 10. 非ブロッキング API . . . 30
図 11. 非ブロッキング API：rfalWorker() の詳細 . . . 31
図 12. エッジ･トリガの ISR：正常時 . . . 32
図 13. エッジ･トリガの ISR：問題あり . . . 32
図 14. エッジ･トリガの ISR：ループ . . . 32
図 15. RFAL の初期化と RF キャリア･オン . . . 40
図 16. NFC-A/ISO14443A ブロッキングのアクティブ化とデータ交換 . . . 41
図 17. NFC-A/ISO14443A のアクティブ化とデータ交換 . . . 42
図 18. NFC-A/ISO14443A のアクティブ化と RFAL HL による データ交換 . . . 43
図 19. NFC-V/FeliCa のアクティブ化と RFAL HL による データ交換 . . . 44
図 20. AP2P イニシエータのアクティブ化と RFAL HL によるデータ交換 . . . 45
図 21. AP2P ターゲットのアクティブ化と RFAL HL によるデータ交換 . . . 46
図 22. 追加フレーム（PSL_REQ） . . . 47
図 23. NFC-A/ISO14443 リスナのアクティブ化と RFAL HL による データ交換 . . . 48
図 24. NFC-F/FeliCa リスナのアクティブ化と RFAL HL による データ交換 . . . 49
図 25. RFAL HL によるポーリング･サイクル . . . 50

重要なお知らせ（よくお読み下さい）

STMicroelectronics NV およびその子会社（以下、ST）は、ST 製品および本書の内容をいつでも予告なく変更、修正、改善、改定および改良する権利を留保します。購入される方は、発注前に ST 製品に関する最新の関連情報を必ず入手してください。ST 製品は、注文請書発行時点で有効な ST の販売条件に従って販売されます。

ST 製品の選択並びに使用については購入される方が全ての責任を負うものとします。購入される方の製品上の操作や設計に関して ST は一切の責任を負いません。

明示又は黙示を問わず、ST は本書においていかなる知的財産権の実施権も許諾致しません。

本書で説明されている情報とは異なる条件で ST 製品が再販された場合、その製品について ST が与えたいかなる保証も無効となります。

ST および ST ロゴは STMicroelectronics の商標です。ST の登録商標については ST ウェブサイトをご覧ください。www.st.com/trademarks

その他の製品またはサービスの名称は、それぞれの所有者に帰属します。

本書の情報は本書の以前のバージョンで提供された全ての情報に優先し、これに代わるものです。

この資料は、STMicroelectronics NV 並びにその子会社(以下 ST)が英文で記述した資料（以下、「正規英語版資料」）を、皆様のご理解の一助として頂くために ST マイクロエレクトロニクス㈱が英文から和文へ翻訳して作成したものです。この資料は現行の正規英語版資料の近時の更新に対応していない場合があります。この資料は、あくまでも正規英語版資料をご理解頂くための補助的参考資料のみにご利用下さい。この資料で説明される製品のご検討及びご採用にあたりましては、必ず最新の正規英語版資料を事前にご確認下さい。ST 及び ST マイクロエレクトロニクス㈱は、現行の正規英語版資料の更新により製品に関する最新の情報を提供しているにも関わらず、当該英語版資料に対応した更新がなされていないこの資料の情報に基づいて発生した問題や障害などにつきましては如何なる責任も負いません。