シグナル・ジェネレータの SCPI プログラミング に 役立つ 10 のヒント

アプリケーションノート

プロダクツ:

- R&S®SMU200A
- R&S®SMBV100A
- R&S®SMJ200A
- R&S®SMATE200A
- R&S®AMU200A
- R&S®SMA100A
- R&S®SMB100A
- R&S®SMC100A
- R&S®SMF100A
- R&S®AFQ100A
- R&S®AFQ100B

このアプリケーションノートは、ローデ・シュワルツのシグナル・ジェネレータをリモート・コントロールするための SCPI プログラミングに関する基本ヒントと情報を簡潔に要約します。

Application Note
C. Tröster
02.2011-1GP79_0J

ROHDE&SCHWARZ

目次

1 概要

このアプリケーションノートは、ローデ・シュワルツのシグナル・ジェネレータをリモート・コントロールするための SCPI[1]プログラミングに関する基本ヒントと情報を簡潔に要約します。

次項のセクションは、SCPI プログラミング に役立つ 10 のヒントの一覧です。これらのヒントと追加の有益なヒントは、このアプリケーションノートの以降のセクションに詳細に記述されます。

リモート・コントロール・プログラミングに関しては、機器の"Operating Manual"や、R&S SMU200A ベクトル・シグナル・ジェネレータ Operating Manual の"Remote Control Basics" セクションを参照ください[1]。このマニュアルをローデ・シュワルツの ウェブサイトから無料でダウンロードできます。このマニュアル内のリモート・コントロールに関する一般的情報はすべてのローデ・シュワルツのシグナル・ジェネレータに適用されます。セクション 11 は、リモート・コントロールの基礎とプログラミングに関する文献の一覧です。

[1] SCPI は Standard Commands for Programmable Instruments の略です。イントロダクションに関しては、[8]を参照してください。

2 10 のヒント

自動テストプログラムは、速く確実に記述されるべきです。SCPI プログラミングに役立つ 10 のヒントは次の概要です:

SCPI プログラミング に役立つ 10 のヒント	
ヒント	**詳細**
定義状態でスタート プログラムの最初に、*RST、*CLS、および SYST:SERR?でのスタティックエラーのクエリ、コマンドを用いて定義状態に機器をリセットします。	セクション 3
***OPC?で待機** 固定ディレイを避けます。その代わりに、コマンド完了を待つために同期コマンド *OPC?を使用します。	セクション 4.2.1
ベースバンドプロセスをポーリング 時間を要するベースバンド演算や波形ロード動作の完了を待つために、周期的に、コマンド SOUR:BB:PROG:MCOD?を用いてベースバンドプロセスをポーリングします。	セクション 4.2.3
ベースバンド設定中はベースバンド Off 演算時間を節約するために、ベースバンドを非アクティブ状態にしている間に、ベースバンド設定(例えばリアルタイムの設定や ARB 設定)を行います。あらかじめ、必要な設定をすべて終えてから、ベースバンドをアクティブ化します。	セクション 6.1
クエリエラー 定期的にループ内でコマンド SYST:ERR?を用いてエラーキューを読み取ります。	セクション 5.1
ロジカルコマンドブロックを形成 ロジカルブロックに複数のコマンドをグループ化して、*OPC?と各ブロック後にエラークエリを送ります。	セクション 6.3
1 ラインに 1 コマンドだけを送る コマンドがある順序に実際に処理されることを確かめる場合、個別のコマンドラインで各コマンドを送ります。	セクション 4.1
機器を同期 相互依存の複数機器をコントロールするとき、機器および測定のエラーを回避するためにデバイス間を同期します。	セクション 4.3
GUI アップデートをスイッチ Off 設定スピードを速くするために、コマンド SYST:DISP:UPD OFF を用いてディスプレイ(GUI)アップデートをスイッチ Off します。	セクション 6.5
オンラインヘルプを利用 特定の設定パラメータや動作に対応する SCPI コマンドを速く容易に見つけるために、機器のオンラインヘルプを使用します。	セクション 10.1.1

3 プログラム開始

自動テストプログラムでは、最初に機器をデフォルトの設定状態に初期化するべきです。常に同一の初期条件を使用することが、成功するテスト実行のための重要な基本となります。さらに、デフォルトの機器設定は、すべての設定にとって共通の初期状態になります。

3.1 機器をプリセット

テストプログラムの初めに、プリセットコマンドを送ります。定義されたデフォルト状態に機器をリセットするために、次のプリセットコマンドのうちの 1 つを使用できます。

SCPI コマンド: `*RST`
SCPI コマンド: `SYST:PRES`

すべての機器設定(現アクティブでないものも)は、デフォルト値にリセットされます。RF 出力はオフになります。

GPIB アドレスや基準信号発生器の設定のような設定は、プリセットコマンドに影響されず変わりません。ステータスレジスタとエラーキューも影響されません。設定/機能がプリセットコマンドの影響を受けるかどうかについては、[1](キーワード"preset key"でサーチ)および[8]を参照してください。

プリセットコマンドが完了するまで、少し時間がかかることに注意してください。

3.2 ステータスレジスタのリセットとエラーキューのクリア

テストプログラムの初めに、機器のステータスレジスタをリセットして、"clear status"コマンドでエラーキューをクリアします:

SCPI コマンド: `*CLS`

このコマンドは、ステータスバイト(STB)、イベント・ステータス・レジスタ(ESR)、オペレーションイベントレジスタ、およびクエリ中のイベントレジスタをクリアします。エラーキューと出力バッファもクリアします。"clear status"コマンドの詳細は、[8]および[1](キーワード "clear status" でサーチ)を参照してください。

3.3 スタティックエラーのクエリ

スタティックエラーは重大な機器エラーを示します。例えば、接続されてない外部リファレンス信号で動作するように機器が設定されている場合、スタティックエラーが発生します。

例:
外部リファレンス信号が接続されてないと仮定します。

プログラム開始で、*CLS コマンドを用いて、エラーキューをクリアしました。外部リファレンス信号がありませんが、通常のエラークエリ(例えば SYST:ERR?)は 0 "No error"をレポートします。よって、機器がエラーの無い状態であることを確かめるために、プログラム開始時に、スタティックエラーをクエリすることは重要です。

```
<<<     SYST:ERR:ALL?
>>>    0,"No error"

<<<     SYST:SERR?
>>>    50,"Warning; External reference oscillator out of range or disconnected"
```

テストプログラムの初めに、下記コマンドでスタティックエラーの理由を機器にクエリします:

SCPI クエリ: `SYST:SERR?`

このクエリは、現在存在するすべてのスタティックエラーをリターンします。スタティックエラーはパーマネントエラーメッセージで、このクエリによって削除(デリート)されません。

スタティックエラーが存在する場合、テストプログラムは適切に対処しなければなりません(例えば、テスト実行を中止し、受信エラーメッセージを表示する)。

4 同期を使いこなす

4.1 コマンド・シーケンス

1 コマンドラインで複数の SCPI コマンドを送ることが可能です。例えば:

コマンドライン: `SOUR:POW:OFFS 20 dBm; :SOUR:POW -50 dBm`

しかし、機器は、コマンドが送られる順で処理するとは限りません[2]。コマンドがある順序で現実に処理されることを確かめるには、個別のコマンドラインで各コマンドを送ります。例えば:

コマンド・シーケンス:
```
SOUR:POW:OFFS 20 dBm
SOUR:POW -50 dBm
```

原則として、コマンドを送り、そして別々のコマンドラインで対応クエリします。例えば:

SCPI コマンド: `SOUR:FREQ 1 GHz`
SCPI クエリ: `SOUR:FREQ?`

4.2 固定ディレイの回避

機器リセットまたは波形ロード動作のようなアクションには、完了するまでにある程度の時間(数秒)がかかります。次ページの表を参照してください。ほとんどの場合、それ以降のコマンドを送る前に、アクションが完了するまで待つことが必要です。確実な動作をさせるために、コマンドの完了を待ってから、次のコマンドを送ることが理想的です。

自動テストプログラムでは、この待機期間を実装するために、ディレイまたはポーズの(言語特有)機能を使用した*固定*ディレイが用いられます。このテクニックは、以下の理由により、お勧めできません:

- コマンド完了が保証されないため、固定ディレイは安全ではありません。このために、プログラマは、多くのマージンをもつ長いディレイを実装する場合があります(かなりプログラムを遅くします)。
- テストプログラムをブロックし、処理時間を無駄に浪費する、待機ディレイを実装する場合、固定ディレイは特に不利です (詳細に関してはセクション 4.2.2.1 を参照)。
- 機器が新しいファームウェアに更新された場合、コマンドによっては、完了するまでの時間が長くなるかもしれません(例えば追加機能により)。このため、プログラムされた固定ディレイの長さは、十分ではないかもしれません。
- プログラマは、固定ディレイを実装するために"何もしない"ループを使用する場合があります。この方法は、異なるプロセッサスピードのコンピュータでテストプログラムを実行する場合に、予測不能のディレイを生じさせる可能性があります。

[2] 例外でこの事実を利用することが可能です。例えば、デジタルスタンダードの相互依存パラメータを構成するとき。

- ディレイ機能は最小ディレイ期間だけを指定します。システムの他の動作のスケジューリングによって、ディレイが要求期間より長くなるかもしれません。さらに、プロセッサタイマは、以前はシステム依存する ms レンジの分解能をもつディレイ期間を推定していました。

以上の理由により、テストプログラムで固定ディレイを使用することは、お勧めしません。次セクションに記述したように、よりよい解決方法が何種類もあります。

時間のかかるオペレーション – 概要		
オペレーション	**時間 (おおよそ)**	**注釈**
機器リセット	< 5 s	
波形ローディング	1 s to 1 min	波形サイズによる
ベースバンド演算	1 s to several minutes	シグナル構成による、例えばフレーム数
セーブ/リコール	< 10 s	
機器校正	10 s to several minutes	校正と機器による
機器のセルフテスト	> 30 s	機器による

4.2.1 オペレーション完了クエリ

実行時間のかかるコマンド完了を待つためにオペレーション完了クエリを使用します:

SCPI クエリ: `*OPC?`

*OPC?の前に送られたすべてのコマンドが実行され、ハードウェアが安定したら、このクエリは"1"をリターンします。テストプログラムがレスポンスを待っている間、さらなるコマンドは機器へ送られません。

例えば、その完了を待つために、コマンド後に*OPC?を直接送ります。

コマンドライン: `SOUR:FREQ 3 GHz; *OPC?`

*OPC?クエリは、レスポンスを待つ間にテストプログラムをブロックします。さらに、"1"が時間内に受信されない場合、タイムアウトが発生します (一般的な VISA タイムアウトは 2 ～ 5 s です。それ以上を設定しないことをお勧めします)。この理由のために、完了に時間がかかるコマンドを待つために*OPC?使用します。

とても時間のかかるコマンドの完了を待つ場合には、他の同期方法を使用することが望まれます。次の 2 セクションに、推奨する 2 つの方法を記述します。

タイムアウト
タイムアウトは、コントローラが機器からのレスポンスを待つ時間を示します。指定時間が経過したら、通信は(タイムアウト)エラーで中止します。

4.2.2 イベント・ステータス・レジスタ内の OPC ステータスをクエリ

時間のかかるオペレーションを実行し、*OPC?で完了を待つ場合、オペレーション終了前にタイムアウトが発生し、"1"を受信しません。さらに、*OPC?で待つ間、テストプログラムはブロックされます。他の(相互依存してない)コマンドを処理することや、他の機器と通信することは、不可能です。

したがって、時間のかかるオペレーションについては、オペレーション完了コマンド*OPC を送ることで通信のブロックを回避できます:

SCPI シーケンス:
```
*CLS
*OPC
```

そしてその後、下記コマンドでイベント・ステータス・レジスタ内のオペレーション完了ステータスをポーリング:

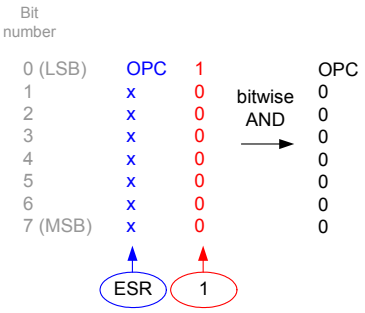
SCPI クエリ: `*ESR?`

このクエリは、イベント・ステータス・レジスタの中身をリターンし、その後に中身をクリアします。イベント・ステータス・レジスタは 8 ビットから成ります。ビット 0 がオペレーション完了ビットに関します。*OPC コマンドが送られ、かつ、*OPC 前に送られたすべてのコマンドが実行された場合、このビットは 1 にセットされます。*OPC?クエリとは対照的に、*OPC コマンドはコマンド処理をブロックしません。

イベント・ステータス・レジスタ	
ビット番号	**意味**
0 (LSB)	オペレーション完了
1	未使用
2	クエリエラー
3	デバイス依存エラー
4	実行エラー
5	コマンドエラー
6	ユーザリクエスト
7 (MSB)	電源オン

*ESR?コマンドは 8 ビット値をリターンし、その後にすべてのビットをクリアします。例えば、レスポンスが 32 であると仮定します。32 の 8 ビットバイナリ表記は、00100000 です。イベント・ステータス・レジスタの MSB はビット番号 7 です。したがって、ビット番号 5 が 1 で、他のすべてのビットが 0 です。この例で、コマンドエラーが発生しています。読み出し後に、ビット番号 5 は 0 にセットされます。

今は、イベント・ステータス・レジスタのビット番号 0 に関しています。受信 8 ビットから OPC ビットを"抽出"するために、10 進数 1 (8 ビットバイナリ表記 00000001)とのビット積"bitwise AND"演算を適用します。



Bitwise AND (ビット積)
論理積 AND 演算は各ペアの対応ビット上で実行されます。各ペアで、ビットが 1 *and* 1 で、結果が 1。そうでなければ、結果は 0 となります。

事実上、10 進数 1 でのビット積演算は、OPC ビット以外のすべてのビットを 0 にセットします。結果は、OPC ビットが 1 である場合には 00000001 であり、OPC ビットが 0 である場合には 00000000 です。10 進数表記では、結果は、1 (オペレーション完了)あるいは 0 (オペレーション未完了)です。

リターンされた OPC ビットが 1 であるまで、イベント・ステータス・レジスタを周期的にポーリングします。

4.2.2.1 ループを用いる ESR ポーリング

イベント・ステータス・レジスタをポーリングする最も容易な方法は、ディレイ関数(C での `sleep()` 関数のような)でループを使用することです。

テストプログラムや現アクティブスレッドをブロックしないように、*busy 待機*しないディレイ関数を使用することが重要です。

Busy 待機
Busy 待機は、異なるタスクを実行するので、その使用される CPU リソースを浪費します。一般的に、busy 待機は避けるべきです。ディレイは、ほとんどの時間を"スリープ状態"で費やすように、CPU 時間をほとんど費やさない busy 待機無しを用いて実装されるべきです。
Busy 待機無しは、実行停止になるアクティブスレッドと、他のスレッドに CPU リソースをパスすることをもたらします。

例:

- ESR をクリア: *CLS
- LTE ベースバンド・シグナルをアクティブ化: SOUR:BB:EUTR:STAT ON
- コマンドを送る *OPC
- ループを使用
  - クエリを送る *ESR?
  - 10 進数 1 とのビット積演算を実施することによるレスポンスを処理

    MATLAB スクリプト例:

    ```
    % ESRvalue is the response returned by query *ESR?

    OPCbit = bitand(ESRvalue,1);
    ```

    C コード例:

    ```
    // ESRvalue is the response returned by query *ESR?

    OPCbit = ESRvalue & 1;
    ```

  - リターンした OPC ビットを評価:
    - OPC ビットが 0 ならば、ポーリングを継続
    - OPC ビットが 1 ならば、ループを止め、ポーリングストップ
  - ディレイ関数を適用 (busy 待機無し)

擬似コード例:

```
send_command (err, '*CLS')
send_command (err, 'SOUR:BB:EUTR:STAT ON')
send_command (err, '*OPC')
ESRvalue = 0

while (ESRvalue & 1) == 0
{
      ESRvalue = send_query (err, '*ESR?')
      sleep (100)
}
```

スリープ中、CPU はループによって占有されません。また、プロセッサは、他のタスクを実行することができます。例えば、イベントループを処理するなど。ループ内にそれぞれのコマンドを含めることにより、イベント処理をトリガすることができます。この方法で、プログラムは規則的に GUI を更新することができ、"中止"ボタンのような GUI からの、あるいは、工業ワークフローで使用されるような外部制御信号からの、入力に反応できます。

ループ内のイベント・ステータス・レジスタのポーリングは、単純で確実であり、推奨する方法です。さらにチャレンジングな代案は、次セクションに記述するタイマ方法です。

4.2.2.2 タイマを用いる ESR ポーリング

イベント・ステータス・レジスタは繰返しタイマによってポーリングされることもできます。

例:

- ESR をクリア: `*CLS`
- LTE ベースバンド・シグナルをアクティブ化: `SOUR:BB:EUTR:STAT ON`
- コマンドを送る `*OPC`
- 繰返しタイマをスタート (busy 待機無し)
- タイマイベントが発生したらクエリを送る `*ESR?`
- 10 進数 1 とのビット積演算を実施することによるレスポンスを処理
- リターンした OPC ビットを評価:
  - OPC ビットが 0 ならば、ポーリングを継続
  - OPC ビットが 1 ならば、ループを止め、ポーリングストップ

タイマ方法を使用するコード例についてはセクション 8 を参照してください。

イベント・ステータス・レジスタをポーリングする間に、機器へ他の相互依存しないコマンドを送ることができます。しかしながら、その間に*OPC?コマンドを使用してはなりません。このクエリは STAT ON コマンドにも当てはまり、タイムアウトにつながるためです。

4.2.3 ベースバンド進捗をポーリング

ベースバンド・シグナル演算は時間のかかるプロセスです。構成されたベースバンド設定次第で、ベースバンド・シグナルの演算時間は、VISA タイムアウト設定より長くなりえます。ゆえに、ベースバンドをアクティブ化し、*OPC?で完了を待つ場合、演算が終了する前にタイムアウトが発生し、リターン"1"を受信できません。

時間のかかるベースバンド演算と ARB 波形ロードについては、下記コマンドでベースバンド進捗をポーリングします:

SCPI クエリ: `SOUR:BB:PROG:MCOD?`

このクエリは、現在の演算進捗を示す 0 ～ 100 の値をリターンします。プログレスバーに似た、“0”が演算の 0 %完了で、“100”が 100 %完了を意味し、演算終了です。

“100”がリターンされるまで、ベースバンドの進捗を*周期的に*ポーリングします。

例 – ループ方法:

- LTE ベースバンド・シグナルをアクティブ化: `SOUR:BB:EUTR:STAT ON`
- ループを使用
  - SCPI クエリを送る `SOUR:BB:PROG:MCOD?`
    最初のコールは、あまり早く起こしてはなりません! 演算が未だスタートしていない場合、リターン値は 100 もありえます。最初のコールは、STAT ON コマンド[3]から数 ms 後に起こすべきです。
  - レスポンスを評価:
    - リターン値が < 100 ならば、ポーリングを継続
    - リターン値が 100 ならば、ループを止め、ポーリングストップ
  - ディレイ関数を適用 (busy 待機無し)
- クエリを送る `*OPC?`
  演算完了後、機器は安定するための追加時間を必要としません。*OPC? クエリは実際のコマンド完了を保証します。

ループ内のベースバンドの進捗をポーリングすることは、単純で確実です。時間のかかるベースバンド演算と ARB 波形ロードを待つために使用される、推奨方法です。

ループ方法とタイマ方法の詳細な記述に関しては、セクション 4.2.2 を参照してください。

それ以上の同期方法には、参考文献[1](キーワード"preventing overlapping execution"でサーチ)と、[3]で見つけることができます。

4.2.4 要約

4.2.4.1 同期コマンド

完了に少時間ほどかかるコマンドを待つために、*OPC?クエリを使用します。大多数のオペレーションは速いので、このクエリがほとんどの場合に使用することをお勧めします。

- 利点: *OPC? 使用にとても単純です。
- 欠点: *OPC?はタイムアウトに至り、時間のかかるオペレーションを待つためには使用できません。*OPC?はテストプログラムをブロックします。

時間のかかるベースバンド演算や ARB 波形ロードを待つために、ディレイ関数を含むループ内に、SOUR:BB:PROG:MCOD? クエリを使用します。

- 利点: この方法はタイムアウトを回避し、テストプログラムをブロックしません。
- 欠点: ベースバンド処理に限定されます。

*CLS と *OPC コマンドを使用し、その後に時間のかかるオペレーションを待つために、ディレイ関数を含むループ内に、*ESR? クエリを使用します。

- 利点: この方法はタイムアウトを回避し、テストプログラムをブロックしません。
- 欠点: 他の同期方法ほど単純ではありません。

[3] 機器の最新ファームウェアリリースを使用します。

4.2.4.2 ポーリング方法

SOUR:BB:PROG:MCOD?を用いてベースバンド進捗をポーリングするためや、*ESR?を用いてイベント・ステータス・レジスタをポーリングするために、ループ方法とタイマ方法の 2 つの方法があります。

ディレイ関数を含むループ:

- 利点: この方法は実装するのにとても単純で確実です。
- 欠点: テストプログラムがループ内で"身動きとれない"ので、ポーリング中、相互依存しない他のコマンドを機器へ送ることができません。その間、他の機器との通信もできません[4]。

繰返しタイマ:

- 利点: 機器や他の機器との通信はポーリング中にブロックされません。
- 欠点: この方法は、使用プログラミング言語次第で実装するのが難しいです。

4.3 マルチ機器の同期

しばしば、自動テストプログラムは、単一のシグナル・ジェネレータだけでなく、テスト・セットアップのすべての機器もコントロールします。この場合、正確な測定結果を確保するために、ジェネレータ、スペクトラム・アナライザ、およびパワー・メータのような異なる機器を同期させることが重要です。

4.3.1 テストシグナルの安定化

例えば、テスト・セットアップは、DUT にテストシグナルを供給するシグナル・ジェネレータ、DUT の出力を解析するスペクトラム・アナライザから成ります。両機器は同じテストプログラムによって構成されます。

RF 周波数やトリガ設定のような一般的な機器設定をシーケンシャルに、あるいは両機器をパラレルに構成することができます。

スペクトラム・アナライザで測定がスタートする前に、テスシグナルが安定していることを確実にしなければなりません。テストシグナルを構成し有効化して、最終的に*OPC?クエリを送ります。レスポンスを待ちます。ジェネレータがコマンド完了を確認後、テストシグナルはレディ状態になり、リモートで測定をスタートできます。

機器を同期させなければ、スペクトラム・アナライザが既に測定スタートしているのに、ジェネレータは未だテストシグナルを演算しているかもしれません。明らかに、これは間違った結果につながります。

[4] マルチスレッディングと呼ばれる精巧なテクニックを駆使して、これらの制限を打開することは可能です。このテクニックは、実装することが容易でなく、プログラミングエキスパートだけに推奨します。

4.3.2 マスタ-スレーブ モードにおけるジェネレータ

例えば、テスト・セットアップはマスタ-スレーブ構成で複数のシグナル・ジェネレータ(例えば複数の R&S®SMBV100A)から成ります。このセットアップでは、1 台のジェネレータが、マスタ機器として機能し、スレーブ機器として機能する他のジェネレータに同期信号を与えます。マスタ-スレーブ構成は、機器の完全な同期を確実にする共通のクロックとトリガ信号によって、ベースバンドのカップリングを実現します。さらに、RF セクションは、位相コヒーレントシグナル・ジェネレータを可能にする共通のローカルオシレータ信号によってカップリングできます。

マスタ機器は、スレーブ機器より*前に*構成されなければなりません。マスタ機器を最初に構成して、*OPC?クエリを用いて機器が安定したことを確かめます。それから、スレーブ機器を構成します。

マスタ機器とスレーブ機器を同期しなければ、スレーブ機器でエラーメッセージにつながります。

5 エラークエリ

エラーのクエリは重要で、プログラム実行中に定期的に行われるべきです。特に、自動テストプログラムやテスト・セットアップの開発中に、エラーを迅速に検出するために、たびたびエラークエリを送るべきです。また、時間節約する緊急を要するアプリケーションでエラークエリを省きません。少なくとも、テスト実行の始めと終わりにエラーをクエリします。始めに、機器からスタティックエラーも読み出すべきです(セクション 3.3)。

次の表で、対応エラークエリを要約します:

エラークエリ	
クエリ	**説明**
SYST:ERR?	エラーキュー内の*最後の*エントリをクエリしてデリート
SYST:ERR:CODE?	SYST:ERR? に似ているがエラー番号だけをリターン
SYST:ERR:ALL?	エラーキュー内の *すべての*エントリをクエリしてデリート
SYST:ERR:CODE:ALL?	SYST:ERR:ALL? に似ているがエラー番号だけをリターン
SYST:ERR:COUNT?	エラーキュー内のエントリ数をクエリ
SYST:SERR?	*スタティック*エラーをクエリ

5.1 エラーキューをクエリ

テストプログラム中に、ループ内で下記コマンドを用いて、定期的にエラーをクエリします:

SCPI クエリ: `SYST:ERR?`

このコマンドは、エラーキュー内の*最後の*エントリをクエリしてデリートします。レスポンスが 0 "No error" であるまで、ループ内でこのコマンドを使用します(セクション 8 の例を参照)。

エラーが発生した場合、エラーメッセージがリターンされます。その後、テストプログラムは適切に対応するべきです。例えば、テスト実行を中止させて、受信エラーメッセージを表示します。最後のエラークエリからエラーが発生していない場合、レスポンスは 0 "No error" です。機器が単一コマンドに応じてエラーキュー内に 1 以上のエラーメッセージを書き込むことに注意してください。

ゆえに原則として、エラーをクエリする*ループ内で*SYST:ERR? コマンドを使用します。無限ループを防ぐために、指定した繰返し数後に、プログラムでループを終了することが望ましい。例えば、カウンタを用いることによって。

コマンド SYST:ERR:ALL? は、すぐにエラーキュー内の *すべて*のエントリをクエリしてデリートすることに注目してください。このクエリに対するレスポンスは長くなるかもしれません。テストプログラム内で十分な受信バッファを確保していなければ、更なるエラー("Query interrupted")を生み出します。対照的に、SYST:ERR? クエリは、より少ないバッファサイズを必要とする利点がありますが、全体のエラーキューを読み出すために、ループ内で使用しなければなりません。

5.2 スタティックエラーをクエリ

スタティックエラーは重大な機器エラーを示します。例えば、接続されていない外部リファレンス信号で動作するように機器が構成されている場合、スタティックエラーが発生します。スタティックエラーと普通の一時的なエラーには相違があることに注目してください。スタティックエラーは恒久エラーメッセージで、関連するクエリーを用いて読み出し後にデリートされません:

SCPI クエリ: `SYST:SERR?`

テストプログラムのスタートでスタティックエラーを機器にクエリすることは重要です。プログラム実行中に SYST:SERR? でスタティックエラーを機器にクエリする場合、このクエリは、現在存在するスタティックエラーだけをリターンすることに注意してください。一時的エラーメッセージはレポートされません。エラーキュー内でリスト化されるので、SYST:ERR? クエリを用いて読み出されなければなりません。

例:
外部リファレンス信号の未接続によるスタティックエラーと、6 GHz 機器に 9 GHz の RF 周波数を設定することによる一時的エラー、を起こします。2 つの異なるエラークエリのレスポンスは下記のとおりです:

```
<<<    SYST:SERR?
>>>    50,"Warning; External reference oscillator
       out of range or disconnected"
<<<    SYST:SERR?
>>>    50,"Warning; External reference oscillator
       out of range or disconnected"
```

whereas

```
<<<    SYST:ERR:ALL?
>>>    -222,"Data out of range;(A) (IdPDbFreq)"
<<<    SYST:ERR:ALL?
>>>    0,"No error"
```

6 スピード最適化

下記では、自動テストプログラムのスピードをどのように最適化でき、何秒まで実行時間を節約できるかを説明します。

6.1 設定の構成

ベースバンドをアクティブ化すると、シグナル・ジェネレータはベースバンド・シグナルを演算し始めます。構成された設定次第で、演算は数秒までかかります。すぐに、シグナルを構成するいくつかのコマンドを送ると、機器はシグナルを再演算します。実は、機器は、変更された設定に適応するために新たなコマンドを受信するごとに、*再演算*します。結果として、すべてのコマンドを実行するのにかかる時間が、再演算によって不必要に長くなります。

この理由で、ベースバンドが非アクティブ化されている間に、ベースバンド設定を構成します。これは演算時間を節約します。同じ理由で、例えば、フェージングが非アクティブ化されている間に、フェーダ設定を構成します。

シグナル・ジェネレータでは、"State"が"Off"で未だアクティブ化されなくても、対応オプションが機器にインストールされているとすれば、デジタル規格用のすべてのコマンドが利用可能であることに注目してください。規格関連のコマンドを送る前にアクティブ化する必要はありません。これは欠点でもあります。スペクトラム・アナライザは、この点で異なって作用することに注目してください。スペクトラム・アナライザでは、規格関連のコマンドが利用可能になる前に、先ず規格をアクティブ化しなければなりません。

原則として、ベースバンド(あるいは ARB、フェーダなど)をアクティブ化する前に、すべての必要な設定を構築します。

いくつかのコマンドを送りたければ、次の例に似たコマンド・シーケンスを使用します:

コマンド・シーケンス:
```
SOUR:BB:EUTR:STAT OFF
SOUR:BB:EUTR:DUPL TDD
...
...
SOUR:BB:EUTR:DL:SUBF5:ALL2:MOD QPSK
SOUR:BB:EUTR:STAT ON
*OPC?
```

その後、特定の設定を変更するために単一コマンドだけを送りたければ、必ずしも非アクティブ化状態に戻る必要がありません。"State"が"On"のまま、コマンドを送れます。

コマンドライン:
```
SOUR:BB:EUTR:DL:SUBF5:ALL2:MOD QAM16; *OPC?
```

6.2 同期

オペレーション/コマンドごとに最適な同期方法を選んでください(セクション 4.2.4 参照)。固定ディレイとは対照的に、待ち時間が最小になることを確実にします。

さらに、コマンド完了を待っている間、相互依存しない他のアクションを実行できます。例えば、シグナル・ジェネレータがベースバンド・シグナルを演算している間、同じテストプログラムは、テスト・セットアップ内の別の機器(例えばスペクトラム・アナライザ)を構成できます:

- シグナル・ジェネレータのベースバンドをアクティブ化
- 何か他のことをするために待機時間を使用、例えばスペクトラム・アナライザを構成
- シグナル・ジェネレータがオペレーション Ready であることを確かめるために、"100" がリターンされるまで(セクション 4.2.3 参照)、ループ内でベースバンド進捗をポーリング

6.3 コマンドブロック

実行時間を短くするためには、コマンドごとの後にオペレーション完了とエラークエリを送らないでください。その代わりに、ロジカルブロック内にコマンドをグループ化して、ブロック後に一度だけクエリを送ります。

コマンド・シーケンス:
```
SOURce1:FREQ 1 GHz
SOURce2:FREQ 3 GHz
SOURce1:POW -10 dBm
SOURce2:POW -30 dBm
*OPC?
SYST:ERR?
```
(レスポンスが 0 “No error” になるまでループ)

6.4 波形

6.4.1 転送時間を節約

不必要な波形転送を避けてください。同一波形が複数プログラム実行で必要な場合、最初の実行だけで、波形生成し、機器へ転送します。デフォルトのディレクトリ内にユニークな名前で保存します。すべてのその後の実行について、波形が存在するかどうかを機器にクエリする、下記コマンドを使用します:

SCPI クエリ: `SOUR:BB:ARB:WAV:CAT?`

このコマンドは、デフォルトのディレクトリ内のすべての波形ファイルの名前をリターンします。波形が存在する場合、直接 ARB にロードします。波形が存在する場合、直接 ARB にロードします。

同一波形が複数回必要とされるならば、この方法は転送時間を節約します。

同じ方法は、それぞれの SCPI クエリを用いる([1]参照 - キーワード"catalog?"でサーチ)、コントロールリスト、データリスト、RF リストにも適用できます。

6.4.2 ロード時間を節約

不必要な波形ロードのオペレーションを避けてください。同一波形が複数プログラム実行で必要な場合、波形生成し、機器へ転送して、*オフライン*でマルチセグメント波形を生成します。参考文献[1]は、機器にマルチセグメント波形を生成する方法を詳細に説明します(キーワード "multi segment"でサーチ)。マルチセグメント波形は、テスト実行中に必要とされる異なる波形を含みます。各波形は、マルチセグメント波形の 1 セグメントを表わします。デフォルトディレクトリ内にユニークな名前で保存します。すべての実行について、マルチセグメント波形が存在するかどうかを機器にクエリし、ARB にロードします。必要となるときに、テスト実行中に、個々のセグメントを再生できます。マルチセグメント波形の 1 波形から別の波形に変わることで、ロードのオペレーションを必要としないことに注目してください。とても急速に波形切替が可能となり、ロードのオペレーションに通常起因するディレイが省略されます。

同一波形セットが複数回必要とされるならば、この方法はロード時間を節約します。

詳細に関しては、参考文献[5]および[2]を参照し、この方法がアプリケーションに適切かどうかを評価してください。

6.5 GUI アップデート

機器の GUI の更新は、計算するリソースを消費します。しかしながら、自動テストシステムでは、グラフィックディスプレイを通常必要としません。
ゆえに、設定スピードを向上させるには、下記コマンドで GUI アップデートをスイッチオフします。

SCPI コマンド: `SYST:DISP:UPD OFF`

6.6 GPIB と LAN

機器との通信は、GPIB か LAN (VXI-11)接続経由で確立することができます。

GPIB 接続は、LAN 接続より少ない待ち時間をもっており、コントロール・コマンドを送るために、より速いインタフェースです。例えば、ID クエリ(*IDN?)は、レスポンス受信を含んで、LAN (VXI-11)より GPIB の方が速いです [6]。LAN 上でコマンドを送る時間は、ネットワークインフラおよび LAN インタフェーススピードのようないくつかの要因に依存します。しかしながら、大量のデータを機器に転送するとき、LAN 接続は有益です。例えば、MMEM コマンドを用いる大きな波形を転送するとき、転送レートは、GPIB より LAN の方が高速です [6]。詳細に調べるには、アプリケーションノート “Connectivity of Rohde & Schwarz Signal Generators” [6] を参照してください。

大量のデータ(例えば大きな波形)を機器へ送りたいならば、高データ転送レートの恩恵を受ける LAN 接続を使用してください。そうでなければ、GPIB 接続はスピードに関して望ましい。

7 波形転送 & ロード

7.1 波形転送

最初に、R&S®SMU のような Windows ベースの機器には、下記コマンドを用いてデフォルトパスを "D:\"に設定します:

SCPI シーケンス: `MMEM:MSIS 'D:'`
`MMEM:CDIR '\'`

ファームウェアアップデートやリカバリを実行するとき、データが消去されないことを確実にします。

R&S®SMBV のような Linux ベースの機器には、下記コマンドを用いてデフォルト・パスを "/var/user"あるいは"/hdd"に設定します:

SCPI コマンド: `MMEM:CDIR '/hdd/'`

ファームウェアアップデートやリカバリを実行するとき、"/var/user"あるいは"/hdd"ディレクトリに保管されたデータは、消去されません。

波形ファイルは、ASCII テキスト(波形ヘッダを形成する必須/オプション情報タグ)と、バイナリデータ(I/Q データ)を含んでいます。機器に波形を転送するには、下記コマンドを用いてバイナリデータブロックとして波形ファイルの全内容を送ります:

SCPI コマンド: `MMEM:DATA '<filename>',`
`#<number><length><binary data bytes>`

`<length>` バイナリデータブロックのバイト長
`<number>` `<length>`の桁数

例: `MMEM:DATA 'test.wv', #267<binary data bytes>`
(コマンド構造の詳細な記述に関しては、参考文献[1]を参照してください。キーワード “mmem:data”でサーチ)

このコマンドは、新しい "test.wv"ファイルを生成するか、既存の "test.wv"ファイルを上書きします。

コマンド SOUR:BB:ARB:WAV:DATA は、機器にバイナリデータを転送することにも使用できることに注目してください。しかしながら、このコマンドはファイルを上書きしないが、ファイルに転送したデータを付加します。

IEC/IEEE バス(GPIB)上のバイナリデータの円滑な送信を確実にするために、標準のデリミタ "line feed" (LF)の代わりに IEC/IEEE バスデリミタ "end or identify" (EOI)を使用し機器を構成します。LF キャラクタをバイナリデータストリーム中にランダムに発生させることができ、このキャラクタの最初の発生で、機器はデータ受信をターミネートするので、これが必要となります。ゆえに、データ転送を始める前に下記コマンドを送ります。

SCPI コマンド: `SYST:COMM:GPIB:LTER EOI`

すぐに、MMEM:DATA コマンドを用いて、機器へバイナリデータを送ります。大量のバイナリデータを転送したければ、参考文献[4]に詳細に記述されるようなブロックごとにデータを送ることが望ましい。ブロックごとにバイナリデータを転送し、EOI で最終ブロックだけをターミネートします。EOI は、IEC/IEEE バス上のデータ転送の終了を示します。転送後、標準の IEC/IEEE バスデリミタに機器を戻すために、下記コマンドを使用します:

SCPI コマンド: `SYST:COMM:GPIB:LTER STAN`

波形生成と転送の詳細な記述に関しては、参照文献[4]を参照してください。このアプリケーションノートは、外部 PC 上で波形ファイルを生成し、機器に転送する方法を示す、役立つコード例も含んでいます。

7.2 波形ロード

波形ファイルが機器の HDD に転送された後、下記コマンドを用い、任意波形ジェネレータ(ARB)に波形をロードします:

SCPI コマンド: `SOUR:BB:ARB:WAV:SEL '<filename>'`

*OPC? でコマンド完了を待ちます。

波形を再生するために ARB をアクティブ化します。

SCPI コマンド: `SOUR:BB:ARB:STAT ON`

コマンド完了を待ちます。使用する同期方法は波形サイズに依存します(セクション 4.2.4 参照)。

波形ファイルがもう(将来にも)必要ない場合、下記コマンドを用いて HDD 上の容量を解放するために、ファイルをデリートするべきです:

SCPI コマンド: `MMEM:DEL '<filename>'`

8 スクリプト例

下記に、シンプルなスクリプト例を示します。このベーシックなスクリプトは MATLAB コードで書かれており、R&S MATLAB Toolkit (アプリケーションノート 1GP60, [7])の関数を使用します。
関数 `rs_send_command` 機器に SCPI コマンドを送ります。
関数 `rs_send_query` 機器に SCPI クエリを送ります。

スクリプト例は推奨プログラム構造を実例します。グレーのコメントと青色の実際の SCPI コマンドに注目してください。

スクリプト

```
% use VISA interface from National Instruments to connect via
TCP/IP
[status, InstrObject] = rs_connect ('visa', 'ni',
'TCPIP::smbv100a100018::INSTR');
if (status<1)
    disp (['*** Return status from rs_connect() is : '
    num2str(status)]);
    clear;
    return;
end

% query instrument info
[status, Result] = rs_send_query (InstrObject, '*IDN?');
if (status<0), return; end
disp (Result);

[status, Result] = rs_send_query (InstrObject, '*OPT?');
if (status<0), return; end
disp (Result);

% create defined conditions
status = rs_send_command (InstrObject,'*RST; *CLS');
if (status<0), return; end

[status, Result] = rs_send_query (InstrObject, '*OPC?');
if (status<0 || Result(1)~='1'), return; end

% query for static errors
[status, Result] = rs_send_query (InstrObject, 'SYST:SERR?');
if (status<0 || Result(1)~='0')
    disp (['*** Instrument error : ' Result]);
    return;
end

% configure and turn on RF signal
status = rs_send_command (InstrObject,'FREQ 2 GHz');
if (status<0), return; end
```

```
status = rs_send_command (InstrObject,'POW -10 dBm');
if (status<0), return; end

[status, Result]= rs_send_query (InstrObject, 'OUTP ON; *OPC?');
if (status<0 || Result(1)~='1'), return; end

% query for errors in a loop
% see 'Query Error Function' below
Err = query_error (InstrObject);
if (Err == 1), return; end  % error occurred

% prepare baseband
status = rs_send_command (InstrObject,'BB:EUTR:STAT OFF');
if (status<0), return; end

status = rs_send_command (InstrObject,'BB:EUTR:PRES');
if (status<0), return; end

% configure LTE baseband signal
status = rs_send_command (InstrObject,'BB:EUTR:SLEN 40');
if (status<0), return; end

[status, Result] = rs_send_query (InstrObject, '*OPC?');
if (status<0 || Result(1)~='1'), return; end

% query for errors in a loop
% see 'Query Error Function' below
Err = query_error (InstrObject);
if (Err == 1), return; end  % error occurred

% activate baseband
status = rs_send_command (InstrObject,'BB:EUTR:STAT ON');
if (status<0), return; end

% create a repetitive timer with 5 s interval
% and a start delay of 1 s
t = timer('TimerFcn',{@poll_progress, InstrObject}, 'Period',
5.0, 'ExecutionMode', 'fixedRate', 'StartDelay', 1.0);

% start timer, i.e. begin polling of baseband progress
% see 'Timer Function' below
start (t);

% we run the following commands in an auxiliary loop (dummy) to
ease error handling
while (true)

      % configure and activate AWGN settings while waiting
      % don't send an *OPC? while waiting, because this would
      % also apply to the BB:EUTR:STAT ON command and cause a
      % timeout
      status = rs_send_command (InstrObject,'AWGN:MODE ADD');
      if (status<0), break; end
```

```
        status = rs_send_command (InstrObject,'AWGN:BWID 10 MHz');
        if (status<0), break; end

        status = rs_send_command (InstrObject,'AWGN:BWID:RAT 2');
        if (status<0), break; end

        status = rs_send_command (InstrObject,'AWGN:STAT ON');
        if (status<0), break; end

        % query for errors - see 'Query Error Function' below
        Err = query_error (InstrObject);
        if (Err == 1), break; end  % error occurred

        % at this point in the code you can continue to send
        % (not interdependent) commands while waiting for the
        % baseband calculation to complete, for example, you can
        % configure another instrument if required/desired

        % wait until baseband calculation is finished
        % (if the calculation is already finished, this loop won't
        % cause a delay)
        while (isvalid(t) == logical (1))
              pause(5);
        end

        % check status
        [status, Result] = rs_send_query (InstrObject, '*OPC?');
        if (status<0 || Result(1)~='1')
              Err = 1;
              break;
        end

        % query for errors - see 'Query Error Function' below
        Err = query_error (InstrObject);
        if (Err == 1), break; end  % error occurred


        break
end   % end auxiliary while loop

% error evaluation
if (Err == 1)
      % check if timer exists and clear timer
      if (isvalid(t) == logical (1))
            stop (t);
            delete (t);
      end
end

% clean up and end
if (Err == 0), disp ('Done.'); end
clear;
return;
```

タイマ関数

```
function poll_progress(t, event, InstrObject)

% poll the baseband progress
[status, Result] = rs_send_query (InstrObject,
'SOUR:BB:PROG:MCOD?');

disp (['baseband progress: ', Result]);

if ( str2num(Result) == 100)
    stop (t);  % stop timer
    delete (t);
end

return;
```

クエリエラー関数

```
function [Err] = query_error (InstrObject)

Result = '1';
Counter = 0;
Err = 0;

% query for errors in a loop until "0, No error" is returned
% and limit the number of iterations to 100
while (Result(1) ~= '0' && Counter < 100)

    [status, Result] = rs_send_query (InstrObject, 'SYST:ERR?');
    if (status<0)
        disp( '*** Error occurred' );
        Err = 1;
        break;
    end
    if (Result(1)~='0')
        disp (['*** Instrument Error: ' Result]);
        Err = 1;
    end

    Counter = Counter + 1;

end

return;
```

9 リストおよび掃引をプログラミング

9.1 RF リスト・モード

RF リスト・モードでは、周波数とレベルのペア値を含んだプリディファインリストに基づいて、RF シグナルが発生されます(詳細に関しては[1]を参照してください。キーワード"list mode"でサーチ)。リストエントリはステップごとに処理されます。RF リスト・モードは、高速周波数やレベルのホッピングを可能にします。

RF リスト(ファイル例 'testlist')を生成あるいは修正し、かつリスト・モードをアクティブ化するために、下記のようなコマンド・シーケンスを使用します:

コマンド・シーケンス:

```
SOUR:LIST:SEL '/hdd/testlist'
```
for Linux-based instruments.
```
'D:\testlist'
```
for Windows-based instruments.

```
SOUR:LIST:FREQ 2 GHz, 4 GHz, 6 GHz, ...
SOUR:LIST:POW 0 dBm, -10 dBm, 10 dBm, ...
SOUR:LIST:DWEL 3 ms
SOUR:LIST:MODE AUTO
SOUR:LIST:TRIG:SOUR AUTO
OUTP ON
*OPC?
SOUR:LIST:LEAR
*OPC?
SOUR:FREQ:MODE LIST
```

リスト・モードでは、高速周波数/レベル切替を達成するために、機器は予め確定したハードウェア設定で作動します。リスト・モードを使用できる前に、RF リスト内の指定周波数/レベルを発生するために必要な内部ハードウェア設定(ステップ・アッテネータ設定など)を、選択リストとともに確定し保存する必要があります。特定リストに必要なハードウェア設定を確定し保存する手順を"list learning"と呼びます。選択リストの list learning を始めるためにコマンド SOUR:LIST を使用します。その後、リスト・モード中に、保存されたハードウェア設定(周波数-レベルのペアで 1 ハードウェア設定)が呼び出されます。

list learning 中に、変調と RF 状態を含む *すべての*ハードウェア設定が保存されることに注目してください。ゆえに、list learning を始める*前に*、OUTP ON を用いて RF 出力と、ベースバンド STAT ON を用いてデジタル変調を ON します。

原則として、すべての所望の機器設定を最初に構築し、それから、list learning して、最終的にリスト・モードをアクティブ化します。

list learning 前にハードウェア安定していることを確実にするために、同期方法(例えば *OPC?)を使用します。特に RF リストが多くのエントリを持っている場合、list learning 自体ある程度の時間がかかります。ゆえに、リスト・モードをアクティブ化する前に、適切な同期方法(セクション 4.2.4 参照)を用いることで、list learning が終了するまで待ってください。

既にリストが存在し list learning されていた(例えば前のプログラム実行で)としても、リスト・モードをアクティブ化する前に、いつも list learning してください。使用されるハードウェア設定がいつも最新であることを確実にするために、SOUR:LIST コマンドは重要です。温度変動は出力周波数とレベルに影響します。機器がウォームアップされるまで、list learning で待ちます。List learning は、現コンディションに順応することや、周波数とレベル確度を保証することに重要です。

下記コマンドを用いてリスト・モードを非アクティブ化できます:

SCPI コマンド: `SOUR:FREQ:MODE CW`

9.2 掃引モード

ほとんどのシグナル・ジェネレータは、いづれか一つをアクティブ化できる異なる 3 つの掃引タイプ(周波数掃引、レベル掃引、LF 掃引)を提供します。掃引タイプごとに、異なる掃引モード(連続、単一、ステップごと)、およびトリガ・モード(オート、内部、外部)を選択できます。

周波数掃引をセットアップしアクティブ化するために、下記のようなコマンド・シーケンスを使用します:

コマンド・シーケンス:

```
SOUR:FREQ:CENTER 200 MHz
SOUR:FREQ:SPAN 300 MHz
SOUR:SWEEP:SPACING LIN
SOUR:SWEEP:STEP:LIN 20 MHz
SOUR:SWEEP:DWELL 12 ms
TRIG:FSWEEP:SOUR SINGLE
SOUR:SWEEP:MODE AUTO
SOUR:FREQ:MODE SWEEP
*OPC?
SOUR:SWEEP:EXECUTE
```

上記例では、実行コマンドでトリガされ、シングル掃引が実行されます。掃引が終了されるまで待ちたければ、掃引完了ではなく、実行コマンド完了(つまり掃引スタートのアクション)を参照するので、この目的のために、*OPC?クエリや*OPCコマンドを使用できません。実行コマンド後に*OPC?クエリを送ると、掃引がスタートしたらすぐに機器は "1"でレスポンスすることを意味します。一般に、シグナル・ジェネレータでは、オペレーション完了コマンドは、コマンド処理の完了だけで、掃引完了自体の完了を示しません。掃引完了を待ちたければ、次のクエリを用いて現在のRF周波数を繰返し読み取ります:

SCPI クエリ: `SOUR:FREQ?`

指定ストップ周波数がリターンされたとき、掃引が完了します。

特に短い滞留時間を使用するとき、最適な掃引パフォーマンスのために、SYST:DISP OFF で GUI アップデート(セクション 6.5)を OFF することが望ましい。

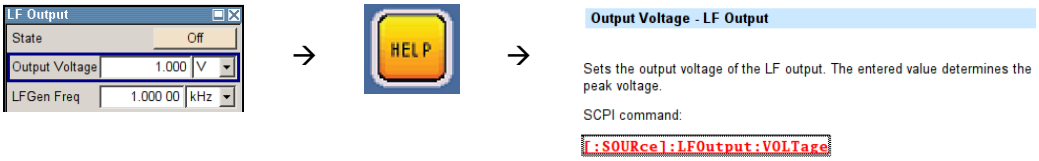
10 その他のヒント

10.1 SCPI コマンドを見つけてテスト

10.1.1 機器ヘルプ

特定の設定パラメータやアクションに対応する SCPI コマンドを見つけるとても容易で便利な方法は、機器の総合的なオンラインヘルプを使用することです。

マニュアル操作で、特定の設定パラメータを選択して、それから、機器のフロントパネルの黄色"Help"ボタンを押します。このボタンは、選択されたパラメータに応じた記述を含むブラウザウィンドウをオープンします。このページの下に、対応する SCPI コマンドが見つかります。



リモートデスクトップや VNC ビューア経由のマニュアル操作では、特定の設定パラメーターを選択して、それから、機器のオンラインヘルプをオープンするために、キーボードの"F1"キーを押します。

コマンド表記法
オンラインヘルプでは、コマンドは特有の構文で記されます: 大/小文字表記法は、ロング/ショート表記形式を区別するのに役立ちます。大文字はショート形式を示し、小文字はロング形式を示します。オプションキーワードは括弧[]で示されます。

例: [SOURce]:LFOutput:VOLTage

ショート表記形式を使用し、オプションキーワードをスキップできます; 例えば、“SOURCE:LFOUTPUT:VOLTAGE” の代わりに、“LFO:VOLT” と記述できます。詳細に関しては参考文献[1]を参照してください (キーワード“syntax”でサーチ)。

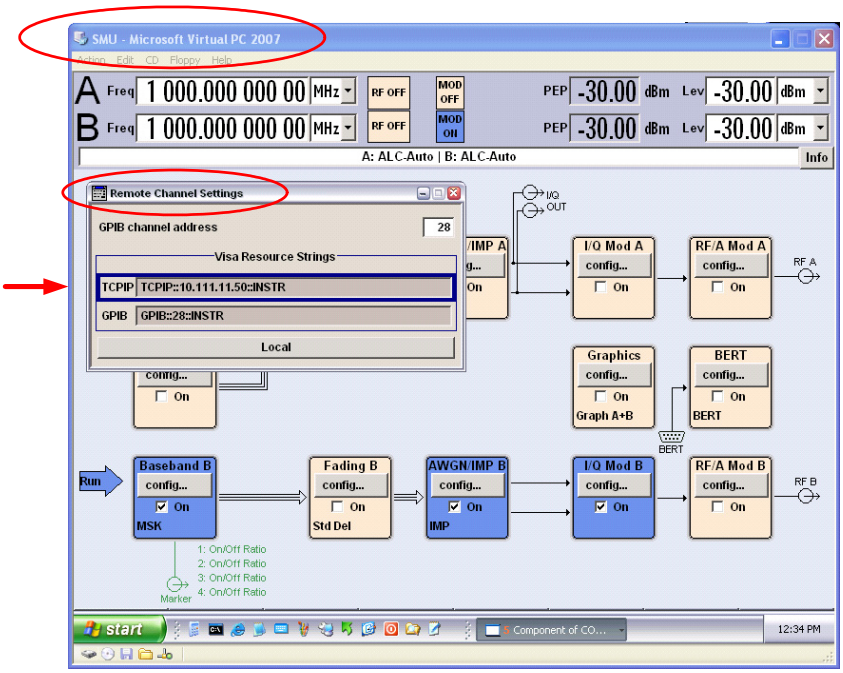
10.1.2 機器シミュレーション

機器(つまり実機)がなく自動テストプログラムを開発しテストする工程がありえます。機器ファームウェアをコンピュータか仮想マシンにインストールでき、機器シミュレータとして役立ちます。

Windows ベースの機器(例えば、R&S®SMU200A, R&S®AMU200A, R&S®SMATE200A, R&S®SMJ100A, R&S®AFQ100A)のファームウェアを Windows マシンにインストールできます。ファームウェアを PC かマイクロソフトバーチャル PC のような仮想マシンかのいずれかにインストールできます。ファームウェアは、ローデ・シュワルツのウェブサイトからダウンロード可能です。
Linux ベースの機器(例えば、R&S®SMBV100A, R&S®SMA100A, R&S®SMB100A, R&S®SMC100A, R&S®SMF100A)のファームウェアを Windows や Linux マシンにダイレクトにインストール *できません*。Windows マシンにインストールするために、リクエストに応じて、この機器ファームウェアの特別ビルドを提供します。(ローデ・シュワルツ本社のカスタマサポートに連絡してください。)

専用 PC(実コンピュータ)か仮想コンピュータにファームウェアをインストールします。インストール手順は機器へのインストールと同じです。コンピュータ 1 台に 1 つの機器ファームウェアをインストールしてください。自動テストプログラムをテストするために、コンピュータの IP アドレスを経て、この仮想機器にリモート接続できます。実機に接続する代わりに、このコンピュータに接続することを意味します。言いかえると、プログラムコード内で機器の IP アドレスを使用する代わりに、コンピュータの IP アドレスを使用します。シミュレートされた機器の GUI (Setup → Remote Settings)でも、この IP アドレスを調べることができます。

コンピュータの Windows ファイアウォールが無効であることを確認してください。そうでなければ、仮想機器に接続できないかもしれません。



シミュレーションは、すべての機器オプションがテスト用に利用可能であるという長所を持っています。シグナル・ジェネレータに未だアクセスしていなくても、すぐにプログラムし始めることができます。ローデ・シュワルツのシグナル・ジェネレータを所有していないユーザにとって、シミュレーションは、機器の全般的印象を得るためのよい方法です。

仮想機器で自動テストプログラム(あるいはその一部)をテストした後、次ステップは、実ハードウェア(つまり実機)で記述コードをテストすることです。シミュレーションは完全に機器のふるまいをエミュレートできないので、これは重要です。

10.2 機器をクエリ

設定コマンドに疑問符を加えることで、パラメータを設定するコマンドをクエリに変えることができます[5]。

例:

SCPI コマンド: `SOUR:FREQ 1.2 GHz`
SCPI クエリ: `SOUR:FREQ?`
レスポンス: 1.2E9

SCPI コマンド: `SOUR:LIST:FREQ 1.2 GHz, 2.0 GHz, 1.5 GHz`
SCPI クエリ: `SOUR:LIST:FREQ? MAX`
レスポンス: 2.0E9

数値は単位なくリターンされます。物理量は、SI 単位か UNIT コマンドを用いた単位になります。真理値(ブール値)は、OFF が 0 で、ON が 1 としてリターンされます。文字データ(テキスト)はショート表記形式でリターンされます。

例:

SCPI コマンド: `HCOP:DEV:COL ON`
SCPI クエリ: `HCOP:DEV:COL?`
レスポンス: 1

SCPI コマンド: `HCOP:PAGE:ORI LANDscape`
SCPI クエリ: `HCOP:PAGE:ORI?`
レスポンス: LAND

10.3 コードのデバッグ

プログラムコードをデバッグする必要がある場合、エラー原因(例えば誤ってスペルしたコマンド)をすぐに見つけるために、コード内の多数の位置に、エラークエリを挿入することが助けになります。とても時間がかかるオペレーションを除き、各コマンドの後に、*OPC? も送り、“1”がリターンされるかどうかをチェックしてください (例えば、観察用にスクリーンに"1"を表示します)。

[5] オペレーティングマニュアルで、他の方法が明らかに指定されないかぎり。

デバッグのための専用ツールも使用できます。例えば、プログラミング環境の特別のデバッグ機能が、個々のコードラインをステップ実行することを可能にします。これらのデバッグツールは、機器のエラーを引き起こすコード内の SCPI コマンドを識別するために使用できます(例えば高すぎるレベル値が機器に送られたから)。別のデバッグツールは、National Instruments の NI Spy ツールです; 何の情報が VISA インタフェース上に実際送られたかを見ることを可能にします。例えば、このツールは、テスト・セットアップの正しい機器か誤った別の機器かに、SCPI コマンドが確かに送られたかどうかをチェックするために使用できます。さらに、予め定義されないが、実行中に計算される値を含む SCPI コマンドを、しばしばコードが含んでいます。これらの値は、例えば、C コードの%f や MATLAB コードの変数などを用いて、SCPI コマンドに含まれています。NI Spy を使用すると、機器に送られた結果の SCPI コマンドが、本来するべきことかどうか、スペースが欠落しているかどうか、計算値が間違っているかどうか(例えば、誤った単位で: Hz ↔GHz)、をチェックできます。

11 詳細文献

このアプリケーションノートは、概して、機器リモートコントロールに関する詳細情報を提供することを意図していません。しかしながら、リモートコントロールの基礎とプログラミングに対応するいくつかの利用可能なアプリケーションノートがあります。例えば、次のアプリケーションノートはとても役立ちます:

- 1GP72 (参考文献[6])
- 1GP62 (参考文献[4])
- 1GP60 (参考文献[7])
- 1EF62 (参考文献[3])

SCPI プログラミングのチュートリアルは、ローデ・シュワルツ発行の書籍 “Automatic Measurement Control” by John M. Pieper で提供されます。

とても役立つ総合的な情報は、参考文献[1]の“Remote Control Basics”セクションで見つけることができます。

12 参考文献

[1] Rohde & Schwarz, R&S®SMU200A Vector Signal Generator Operating Manual.
The manual can be downloaded from the Rohde & Schwarz website:
www.rohde-schwarz.com/product/SMU200A → Downloads → Manuals

[2] Rohde & Schwarz Application Note: “Speeding up production test with the R&S®SMATE200A” (1GP63)

[3] Rohde & Schwarz Application Note: “Hints and Tricks for Remote Control of Spectrum and Network Analyzers” (1EF62)

[4] Rohde & Schwarz Application Note: “Importing Data in ARB, Custom Digital Modulation and RF List Mode” (1GP62)

[5] Rohde & Schwarz Application Note: “Arbitrary Waveform Sequencing with Rohde & Schwarz Vector Signal Generators” (1GP53)

[6] Rohde & Schwarz Application Note: “Connectivity of Rohde & Schwarz Signal Generators” (1GP72)

[7] Rohde & Schwarz Application Note: “R&S MATLAB Toolkit for Signal Generators and Power Sensors” (1GP60)

[8] Rohde & Schwarz Book: “Automatic Measurement Control” by John M. Pieper (ISBN: 978-3-939837-02-2)

ローデ・シュワルツについて

ローデ・シュワルツ・グループ（本社：ドイツ・ミュンヘン）は、エレクトロニクス分野に特化し、電子計測、放送、無線通信の監視・探知および高品質な通信システムなどで世界をリードしています。
75 年以上前に創業し、世界 70 カ国以上で販売と保守・修理を展開している会社です。

ローデ・シュワルツ・ジャパン株式会社

本社／東京オフィス
〒160-0023 東京都新宿区西新宿 7-20-1
住友不動産西新宿ビル
TEL:03-5925-1288/1287 FAX:03-5925-1290/1285

神奈川オフィス
〒222-0033 神奈川県横浜市港北区新横浜 2-8-12
Attend on Tower 16 階
TEL:045-477-3570（代） FAX:045-471-7678

大阪オフィス
〒564-0063 大阪府吹田市江坂町 1-23-20
TEK 第 2 ビル 8 階
TEL:06-6310-9651（代） FAX:06-6330-9651

サービスセンター
〒330-0075 埼玉県さいたま市浦和区針ヶ谷 4-2-20
浦和テクノシティビル 3 階
TEL:048-829-8061 FAX:048-822-3156

E-mail: info.rsjp@rohde-schwarz.com
http://www.rohde-schwarz.co.jp

Certified Quality System
ISO 9001

ローデ・シュワルツ・ジャパン株式会社
〒160-0023 東京都新宿区西新宿 7-20-1　住友不動産西新宿ビル 27 階
TEL:03-5925-1288/1287　FAX:03-5925-1290/1285

www.rohde-schwarz.co.jp