製造までの4つのステップ：
モデル・ベース設計で実現するソフトウェア無線

Part 4：Zynq SoC向けSDRキットとSimulinkによるコード生成のワークフローに基づくラピッド・プロトタイピング

著者：Mike Donovan/Andrei Cozma/Di Pu

はじめに

本シリーズのPart 1～3では、まず「Zynq®-7000 All Programmable SoC（以下、Zynq SoC）」をベースとしたソフトウェア無線（SDR：Software-defined Radio）向けのラピッド・プロトタイピング・プラットフォームを紹介しました[1]。それに続いて、「MATLAB」と「Simulink」を使用し、ADS-B（Automatic Dependent Surveillance-Broadcast：放送型自動従属監視）信号を正しく処理してデコード（復号化）するためのアルゴリズムの開発手順を説明しました[2]。さらに、シミュレーションを使用してそのアルゴリズムを検証する方法と、SDRプラットフォームから取得したライブ・データを使用して検証する方法について解説を行いました[3]。このようにして一連の作業を行ってきたわけですが、最終的な目標は、C言語のコードとHDLのコードに変換可能で、SDRプラットフォームのソフトウェア/ハードウェア・インフラにそのまま統合できる検証済みのモデルを作成することです。

Part 2（「MATLABとSimulinkによるモードS信号の検出とデコード」）で説明したSimulinkモデルは、各種ハードウェアに対する十分な忠実度を備えたシミュレーション・モデルです。そのような忠実度を備えることから、対象となるシステムの設計によって、ADS-Bのメッセージが正しくデコードされることを保証することが可能になります。本稿では、このSimulinkモデルを出発点として本シリーズの最後の作業について説明します。それを通して、Zynq SoCをベースとしたSDR向けのラピッド・プロトタイピング・プラットフォーム上で動作するレシーバ回路を構築します。Part 1～3と同様に、レシーバ回路の開発には、MATLABとSimulinkを使いこなせるだけのスキル、Zynq SoCベースの無線用ハードウェアに関する知識、ソフトウェアとハードウェアを統合するためのスキルが必要です。

本稿で説明する作業の内容は、以下のようにまとめることができます。

- Simulinkモデルを、Zynq SoCのFPGAファブリックとプロセッシング・システム（ARM® Cortex-A9）のそれぞれをターゲットとする機能に分割する
- より高い性能が得られるHDLコードが生成されるようにするために、Simulinkモデルに設計変更を加える
- ADS-B用レシーバのアルゴリズムに対応するHDLのソース・コードとC言語のソース・コードを生成する
- 生成されたソース・コードを、Zynq SoCをベースとする無線プラットフォームの設計に適用する
- ターゲットとするハードウェアで航空機からのライブ信号を取得し、完成した設計のテストを行う

これらの作業を終えれば、完全に検証されたSDRシステムを構築できたことになります。このシステムには、SimulinkベースのADS-Bモデルから自動生成されたC言語とHDLのコードが適用されています。民間航空機からのライブ信号をリアルタイムに受信し、デコードを実施することができます。

ハードウェア、ソフトウェア向けにモデルを分割

まずは、実装用のコードを生成するための最初の作業を行います。この作業とは、設計した回路を、Zynq SoCが備えるプログラマブル・ロジックとプロセッシング・システムのそれぞれで実行する機能に分割することです。

一般に、分割作業は、回路における各コンポーネントの処理の要件と、必要な実行レートや実行時間を洗い出すことから始めます。演算を多用し、サンプル・レートでリアルタイムに実行する必要のある処理（データの変復調アルゴリズムなど）を担うコンポーネントは、プログラマブル・ロジックに実装すべきです。それよりも処理負荷の軽いタスク（データのデコードやレンダリング、システムの監視/診断など）は、ソフトウェアによる実装が向いています。その他の検討事項としては、演算で使用するデータ型や、処理の複雑さ、入出力データの精度などがあります。プログラマブル・ロジックをターゲットとする演算は、いずれも固定小数点、整数、またはBooleanのデータ型を使用します。三角関数や平方根といったより複雑な演算には、利用可能なハードウェア・リソースによって効率的な実装を行うために、近似が適用されます。このような制約は、いずれも精度の低下につながるほか、システムの機能に悪影響を及ぼす恐れもあるため、正しい評価を行ったうえで実装に進む必要があります。一方、プロセッシング・システムをターゲットとするコンポーネントは浮動小数点を扱うことができます。どのような複雑な演算でも高い忠実度で実装できますが、一般に、実行速度は遅くなります。

こうした制約をガイドラインとしてとらえると、ADS-B信号のデコードに使用するアルゴリズムの分割方法が明確になります。I/Qサンプル（サンプリングによって得られたデータ）のフロント・エンドにおける処理からチェックサムの計算まで、すべてはModeS_Simulink_Decode.slxというモデルに含まれています。同モデルのDetectorブロックの機能は、Zynq SoCのプログラマブル・ロジックに実装するのが適切です（図1）。一方、Modified BufferとDecode & Displayの両ブロックで実現するメッセージ・ビットのデコードは、プロセッシング・システムに容易に実装できます。

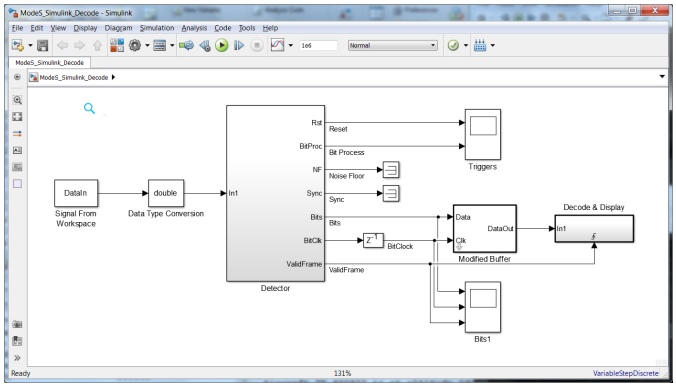


図1. ModeS_Simulink_Decode.slxにおける
FPGAとCortex-A9への処理の分割

アナログ・デバイセズ（ADI）は、以下に示す内容を実装したSimulinkモデルのファイルをGitHubリポジトリで提供しています[4]。

SimulinkモデルからHDLコードを生成する

Mode S Decoderモデルの Detectorブロックは、CalcSyncCorr、CalcNF、SyncAndControl、Bit-Process、CalcCRC、FameDetectの各サブシステムから構成されています（図2）。この回路を基に、MathWorks社の「HDL Coder」[5]を使用してHDLのソース・コードを生成します。

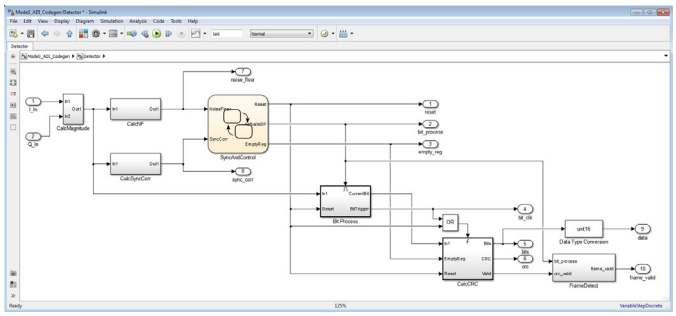


図2. HDLコードの生成に使用するDetectorブロック

HDL CoderでHDLコードを正しく生成するには、Simulinkモデルがいくつかの条件を満たしている必要があります。最も重要な条件を以下に挙げます。

- HDLコードの生成をサポートするブロックを使用する：HDL Coderは、約200のSimulinkブロックに対するコードの生成をサポートしている[6]。Detector回路の場合、Stateflowダイアグラムや Digital Filterブロックを含むすべてのブロックがHDLコードの生成をサポートしている
- 固定小数点のデータ型を使用する：Detector回路では、信号のデータ型として12ビット、24ビット、Booleanを使用する。 12ビットのデータ型は、ADIのRFアジャイル・トランシーバ（RF Agile Transceiver™）IC「AD9361」が備えるA/Dコンバータの分解能と同じである
- スカラー信号またはベクトル信号を使用する：ベクトル信号は、マルチチャンネルの信号またはリソースの共有に使用できる
- モデル内で代数ループは使用しない：HDL Coderは、代数ループの条件を含むモデルについてはHDLコードの生成をサポートしていない

当初、ModeS_Simulink_Decode.slxモデルは上記のすべての条件を満たしているわけではありませんでした。問題があったのは、CalcCRCブロックの中で、受信したビット・データと算出されたチェックサムとを比較する部分です。そこで、このブロックをDetectorブロックの外部に移動し、最終的にはC言語で実装することにしました。このような変更を加えたModeS_ADI_CodeGen.slxモデルを使用して、HDLのコードを生成しました（図3）。手作業でコーディングを行うのとは大きく異なり、わずか数分で数千行のHDLコードを生成できます。HDL Coderによって生成されたソース・コードは、Simulinkモデルのビット精度とサイクル精度をそのまま引き継いでいます。これは、モデル・ベース設計が生産性に対してもたらす主要なメリットの1つです。つまり、生成されたコードは、Simulinkモデルを正確に変換したものになるということです。

また、HDLのコードは読みやすくトレースが可能な状態で生成されるため、技術者は設計したモデルと生成されたコードを容易に対応づけることができます。以下に示す方法によって、このようなことが実現されています。

- モデルの階層が、HDLコードのファイルでも維持される。この例では、トップ・レベルのブロックの名前がDetector.vhdで、次の階層にあるサブシステムの名前がCalcNF.vhd、Bit_Process.vhdといった具合になっている
- モデルで使用したブロック名、ポート名、信号名、データ型、複雑さが、生成後のコードにも踏襲される

モデルとソース・コードはリンクしており、Simulinkモデル内のブロック上でクリックすると、生成されたHDLコードに自動的に移動できます。同様に、生成されたコード内のハイパーリンクをクリックすると、Simulinkモデルが開き、コードのその部分に対応するブロックが強調表示されます。

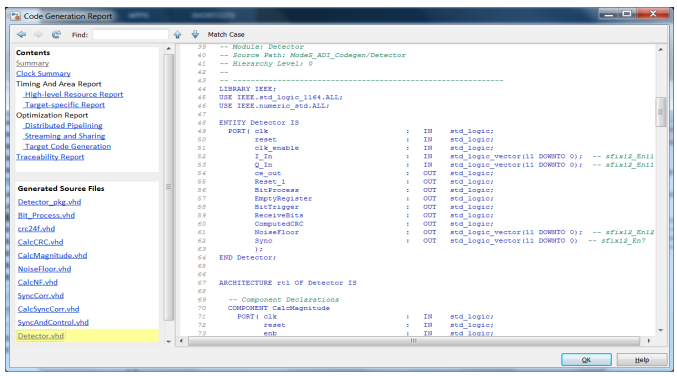


図3. ModeS_ADI_CodeGen.slxから
生成されたHDLのソース・コード

クロックの高速化が可能なHDLコードの生成に向け、ADS-Bモデルを最適化

Simulinkモデルからは正しく機能するHDLコードが生成されますが、最初の生成結果で満足する設計者はほとんどいません。速度や面積の制約を満たせていないことが多いからです。そのため、望ましい結果を得るためにSimulinkモデルの最適化が行われることになります。モデルの最適化を行ったら、シミュレーションを実行し、加えた変更によってアルゴリズムに悪影響が及んでいないことを確認します。Simulinkとコードの自動生成がもたらす大きなメリットは、このようにモデルに対する変

更を行ってHDLコードを再生成できる点です。この方法の方が、HDLのソース・コードを変更するよりもずっと簡単で、ミスも生じにくくなります。HDLのソース・コードに直接手を加えると、アルゴリズムに悪影響が及ぶ恐れがあります。

本稿の回路の場合、モデルから生成されたHDLコードは、比較的低いクロック・レートで実行するため、使用するFPGAファブリックに簡単に適合しました。こういうケースは初期設計の段階ではよくあります。HDL Coderが備えるツールで解析したところ、このモデルのクリティカル・パスは、I/Qサンプルの入力からサブシステムであるCalcCRCの最初のレジスタまででした。このような回路では、パイプライン・レジスタを挿入する方法によってクロック速度を高めるということがよく行われます（図4）。パイプラインを挿入すると、処理全体の遅延が大きくなる代わりに、信号処理の間のパスが短くなります。多くの場合、このトレードオフは許容されるはずです。クロック・レートを高められるのであれば、遅延が多少増加しても大きな代償にはならないからです。

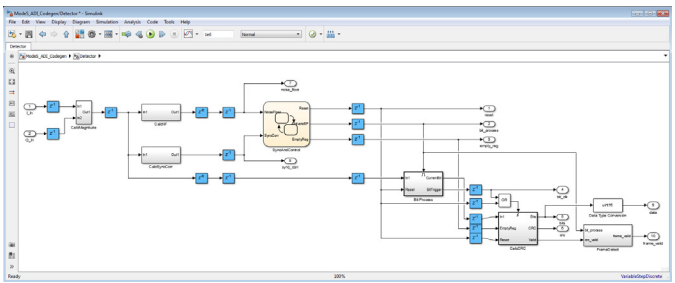


図4. パイプライン・レジスタを挿入したDetector回路

このように、サブシステム間のパイプライン・レジスタによって、クロック・レートを向上することができました。ただ、Digital Filterブロックにおいて適切なアーキテクチャを選択すれば、クロック・レートをさらに高めることができます。多くのSimulinkブロックでは、アーキテクチャについて選択肢が用意されているため、それにより、回路の速度や面積を最適化できるようになっています。ノイズフロアとプリアンブルのコリレーションの計算に使用されるデジタル・フィルタの場合、出力乗算器をパイプライン化することでデジタル・フィルタ内のクリティカル・パスが短くなり、回路のクロック・レートを高めることができます（図5）。

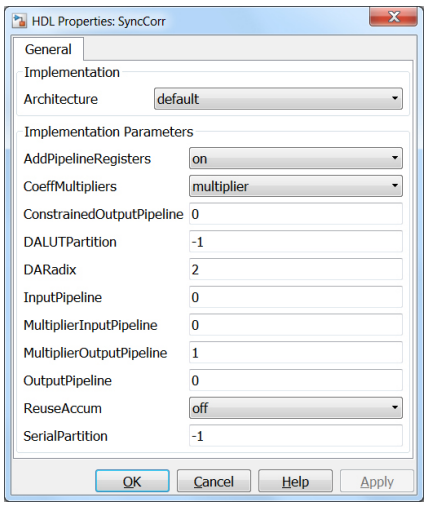


図5. Digital FilterブロックではHDLブロックに関する選択が行える

パイプラインに関する2つの簡単な変更を加えることで、生成されたHDLコードは140MHzのクロック・レートに対応できるものになりました。これは、コード生成ツールを利用する技術者にとっては有益な教訓になるでしょう。その教訓とは、ハードウェアの設計原則に関するちょっとした知識をモデルに適用するだけで、生成されるコードに対して大きな改善をもたらすことができるということです。この例の回路をさらに最適化することも可能ですが、そこまでは行いませんでした。この例ではタイミングとリソースに関する目標が比較的シンプルだったため、この時点のHDLコードによって十分にそれらを満たすことができていたからです。

従来、無線システムの設計では、開発時間の大部分をHDLコードのテストとデバッグが占めていました。実は、ここまでに示したモデル・ベース設計では、シミュレーションとコード生成の対象となるモデルの開発により多くの時間を費やしています。ただし、開発工程の全体を見れば時間は大幅に短縮されています。生成されたソース・コードの動作が、シミュレーションで検証済みの動作と正確に一致するからです。組み込みハードウェア上では、ごくわずかのデバッグ作業しか必要ありませんでした。

MATLAB CoderでCコードを生成する

HDLのコードを生成する場合と同様に、「MATLAB Coder」[7]によって、この回路のデコード機能を実現するC言語のコードを生成するには、いくつかの条件を満たす必要があります。最も重要な条件は次の2つです。

- MATLAB Coderがサポートしている関数を使用する：MATLAB Coderは、MATLAB言語のほぼすべてに加え、さまざまなツールボックス[8]をサポートしている。それでも、モデルの開発時にMATLAB Coderがサポートしていない関数を使用してしまうことはあり得る。そのため、MATLAB Coderがサポートしていない関数を検出するための「MATLAB Code Generation Readiness Tool（コード生成の準備状態ツール）」[9]などのツールが提供されている
- MATLAB変数を宣言した後にそのサイズや型は変更しない：これは、生成されたコードの中でメモリが正しく割り当てられるようにするために必須の条件である

MATLABのコードからC言語のコードを生成するための最も簡単な方法は、新しいMATLAB Coderプロジェクトを開くことです。これはMATLABツールストリップの「アプリケーション」タブで行うことができます。MATLAB Coderプロジェクトの最終的な出力結果を図6に示しました。

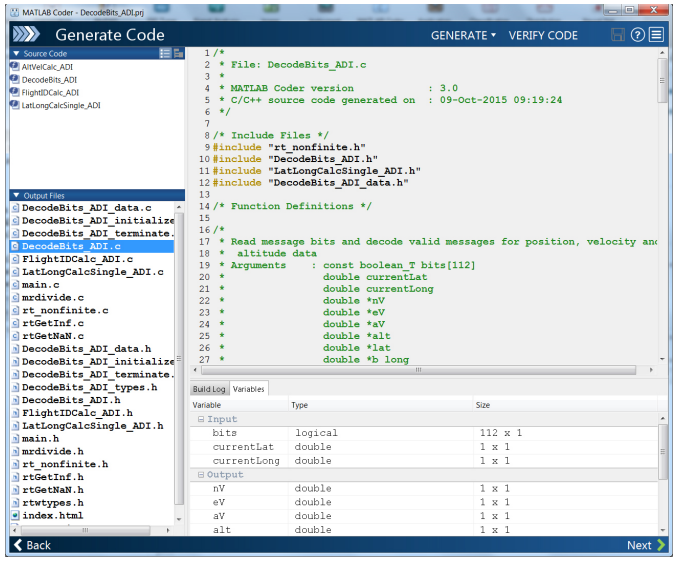


図6. DecodeBits_ADI.mに対応する
MATLAB Coderプロジェクト

このプロジェクトにおいて、トップ・レベルのMATLAB関数はDecodeBits_ADI.mです。ユーザーはこの関数に必要なデータ型とサイズを入力引数として指定する必要があります。図6を見ると、この関数の入力引数は、112ビットのBoolean型データと2つの倍精度の値（ユーザーの現在の緯度と経度用）であることがわかります。DecodeBits_ADI.mの出力のサイズとデータ型（北方向の速度用の*nV、東方向の速度用の*eV、高度用の*altなど）は、MATLAB Coderによって自動的に定められます。MATLAB Coderは、トップ・レベルのエントリ・ポイント・ファイルであるDecodeBits_ADI.mから呼び出されるその他すべての関数（AltVelCalc_ADI.m、LatLongCalc_ADI.mなど）を検索し、デコード用のアルゴリズム全体に対応するC言語のソース・コードを生成します。

MATLAB Coderによって生成されるのは、MATLABで記述した機能をそのままC言語に変換したコードです。HDLコードを生成する場合と同様に、MATLAB Coderによって生成されるソース・コードは読みやすくトレースが可能です。そのため、技術者は元のMATLABコードと生成されたCコードの間の関係を容易に把握することができます。この例のCコードは、MATLABコマンド・プロンプトから生成可能です。また、任意のANSI Cコンパイラでコンパイルすることができます。

HDLコードをプラットフォームに配備する

ここまでの作業では、最初に、設計した回路をZynq SoCのプログラマブル・ロジックとプロセッシング・システムで実行する機能に分割しました。続いて、HDL/C言語のコード生成に向けて回路の最適化を行いました。最適化した回路が正しく動作し、性能面の条件を満たすことをシミュレーションで確認したら、この設計をSDR向けのハードウェア・プラットフォームに配備（デプロイ）し、システムの機能を実際の条件下で検証します。

検証を行うために、ADIのSDRプラットフォーム「AD-FMCOMMS3-EBZ」[10]を使用します。このボードには、ADIのLinuxディストリビューションを搭載するXilinx社のボード「ZC706」[11]を接続しています。

AD-FMCOMMS3-EBZには、ADIが提供するオープンソースのHDLリファレンス設計「Vivado」が付属しています[12]。このリファレンス設計には、構成（コンフィギ

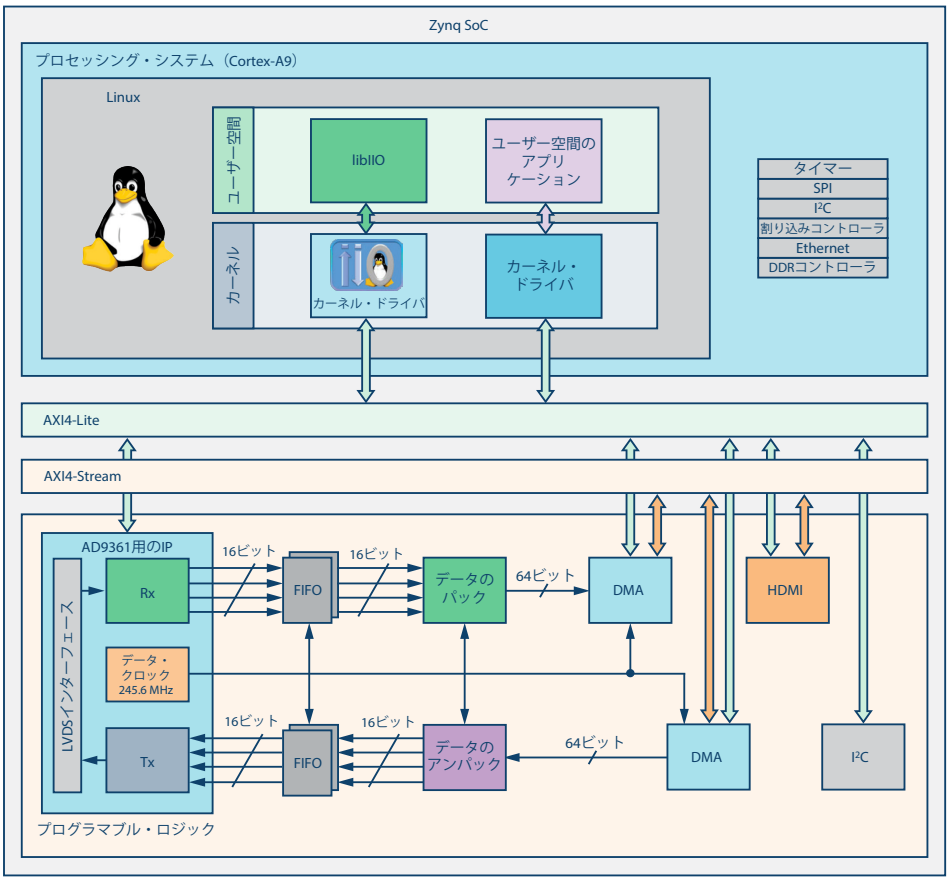


図7. HDLリファレンス設計のブロック図

ュレーション）とAD-FMCOMMS3-EBZ上のAD9361との間のデータ送受信に必要なすべてのIPブロックが含まれています。図7に、このHDLリファレンス設計のブロック図を示しました。

AD9361用のIPコアは、AD9361とZynq SoCの間でデータを送受信するためのLVDSインターフェースと、回路のその他の部分のデータ・インターフェースを実現します。AD9361用IPとDDRメモリの間の高速データ転送にはDMAブロックが使われます。AD9361用IPブロックに対するデータ・インターフェースは、受信用と送信用にそれぞれ4本のデータラインを備えています。これらはAD9361の2個の受信チャンネルと2個の送信チャンネルのI/Qデータに対応しています。各データラインは16ビット幅です。システム内のデータ転送をより効率良く行うために、送受信データは、DMAブロックによって管理される64ビット幅のバスにパッキングされます。パックとアンパックを行うブロックにより、AD9361用IPにつながる16ビットの並列データラインがDMAに接続されます。

ADS-BモデルのHDLコードを、SDRプラットフォームにおいてHDLを適用可能な既存のインフラに配備するには、データ・パスに挿入できるIPコアを構築する必要があります。これによって、受信した信号をリアルタイムに処理し、得られたデータをソフトウェアのレイヤに転送します。このような配備の作業は難易度が高く、時間がかかることがよくあります。HDLで設計した機能を細部にわたって理解するだけでなく、HDLのプログラミングについての十分なスキルも必要になるからです。この作業を簡素化するために、MathWorks社はHDL Coderに「HDLワークフロー アドバイザー」というユーティリティを用意しています。一方、ADIは、SDRプラットフォームであるAD-FMCOMMS3-EBZと「AD-FMCOMMS2-EBZ」、Xilinx社のZC706を対象としたボード・サポート・パッケージ（BSP）を提供しています[13]。

HDLワークフロー アドバイザーは、SimulinkモデルからHDLコードを生成するために必要な作業を正しい手順で進められるようにユーザーを導いてくれます。ユーザーは、「ASIC/FPGA」、「FPGA-in-the-Loop」、「IP Core Generation（IPコアの生成）」といった複数の選択肢の中からターゲットとするワークフローを選択できます。ターゲットとなるプラットフォームとしては、Xilinx社の評価用ボード、Altera社の評価用ボード、またはSDRプラットフォームであるFMCOMMS2/3 C70606を選択できます。その後に行うコードの生成とターゲットへの統合の処理は、HDLワークフロー アドバイザーによって自動的に実施できます。

ADIが提供するBSPは、ボードの定義とリファレンス設計を集めたものです[14]。これによって、既存のHDLリファレンス設計と同等のIPブロックを生成し、それをHDLリファレンス設計に組み込むために必要な情報とツールが、HDLワークフロー アドバイザーに提供されます。図8は、ADS-BモデルのIPコアを生成するようにHDLワークフロー アドバイザーを設定する方法を示したものです。ワークフローとしては「IP Core Generation 」を選択し、ターゲット・プラットフォームとしては「AnalogDevices FMCOMMS2/3 ZC706」を選択する必要があることに注意してください。

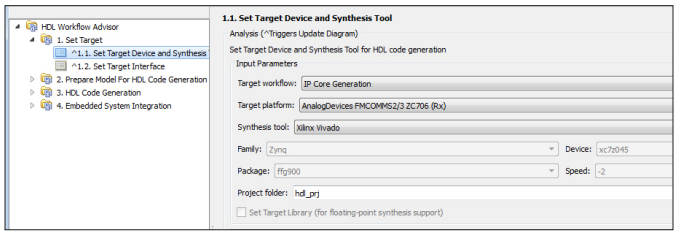


図8. HDLワークフロー アドバイザーの設定

次のステップでは、IPとリファレンス設計の間のインターフェースの構成を行います。モデルの入力では、未処理のI/Qサンプルを受け取ります。モデルの入力ポートはAD9361のデータ・ポートに直接接続します。モデルの出力信号のうち、ここで関心があるのはdata、frame_valid、bit_clkの各信号だけです。dataとframe_validは16ビット幅で、bit_clk信号はクロックとして使用します。これらの信号はBSPのDUT Data x Outインターフェースに接続可能です。つまり、DMAのブロックはこれらの信号に直接アクセスします。その後、dataはソフトウェア・レイヤからアクセスが可能なDDRに転送されます。bit_clk信号は、BSPのDUT Data Valid Outインターフェースに接続され、DMAのサンプリング・レートを制御します。図9に、HDLインターフェースの構成方法を示しました。

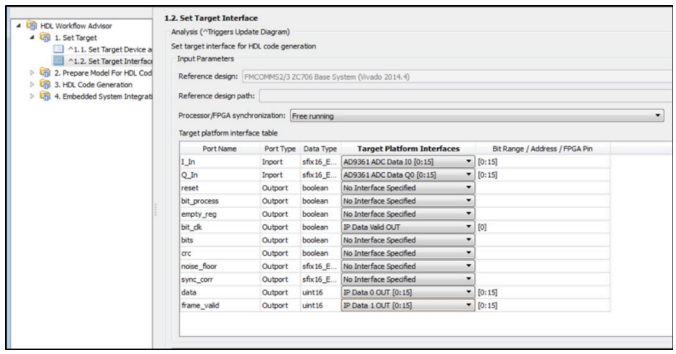


図9. HDLインターフェースの構成

ターゲットとのインターフェースを定義したら、次のステップに進みます。HDLワークフロー アドバイザーのステップ2とステップ3はデフォルトの設定のままとし、ステップ4.1（プロジェクトの作成）を実行することでプロジェクト作成のプロセスを開始します。このステップの結果として、ADS-B用のIPコアがADIのHDLリファレンス設計に組み込まれたVivadoプロジェクトが作成されます。図10に、ADS-B用のIPコアと回路内のその他のブロックとの接続を示しました。

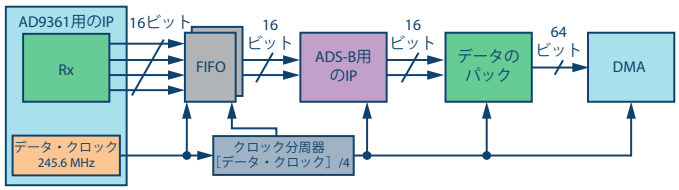


図10. HDLリファレンス設計におけるADS-B用IPの接続

Vivadoプロジェクトからビット・ストリームを生成することで、HDLベースの統合作業は完了します。次の目標はシステム上でLinuxを稼働させることです。そこで、ビット・ストリームを生成した後に「Xilinx SDK FSBL（First Stage Boot Loader）」と標準的なプロセスに従ってLinux用のブート・ファイルを

作成します。新たに構築したHDLベースの回路に対応するLinuxのデバイス・ツリーとイメージ・ファイルが、AD-FMCOMMS3-EBZ用のBSPと共に配布されています。すべてのファイルはLinux用のブート・ファイルと共にSDカードのブート・パーティション上にコピーします。ここには、Xilinx社のZC706上でADIのLinuxディストリビューションを実行するために必要なすべてのファイルが格納されていることになります。

Cコードをプラットフォームに配備する

ここまでに、HDLで構築したADS-B用のIPをSDRプラットフォームの回路に適用し、Linux用のSDカードを作成しました。続いては、ADS-Bデータをデコードするためのソフトウェア・アプリケーションを実装します。このアプリケーションは、「MATLAB CoderでCコードを生成する」のセクションで生成したC言語のコードをベースとし、以下のタスクを実行します。

- ADS-Bの信号を受信するようにAD9361を構成する
- ADS-B用のIPコアからデータを読み出す
- 読み出したデータに含まれる有効なADS-Bフレームを検出する
- ADS-B信号に含まれる情報をデコードして表示する

1つ目と2つ目のタスクを実装するには、libiioのライブラリで提供されている機能を利用すると便利です[15]。このライブラリは、AD9361の構成とデータの送受信を容易に行うためのインターフェース関数を提供しています。ここでは、まずシステムに関する以下のパラメータを設定します。

- 局部発振周波数：1.09GHz
- サンプリング・レート：12.5MHz
- アナログ帯域幅：4.0MHz
- 自動利得制御：fast attackモード

上記のパラメータ以外に、FIR型のデジタル・フィルタ（データレートは12.5MSPS、通過帯域周波数は3.25MHz、阻止帯域周波数は4MHz）をAD9361に追加し、受信側のデータには対象とする帯域の信号成分のみが含まれるように設定します。このFIRフィルタのシステム・パラメータと設計方法については、本シリーズのPart 3を参照してください[3]。

ADS-B用IPの出力データは、DMAブロックによってシステムのDDRメモリへと転送されます。libiioのライブラリには、ADS-B用IPから取得したデータを指定されたサイズでメモリ・バッファに配置する関数や、バッファが一杯になるのを待つ関数、ポインタを介してバッファにアクセスする関数などがあります。バッファが一杯になると、ADS-Bのデコード・アルゴリズムによるデータの処理を開始できます。ADS-B用のIPコアには、2つの出力チャンネルがあります。1つはADS-Bのビット・ストリームに対応し、もう1つはビット・ストリームに含まれる有効なデータ・フレームの終了位置を示すために使われます。2つのチャンネルは、同じデータレートで動作し、互いに同期がとれています。後者のチャンネルでサンプルの値が1であれば、それはデータ・チャンネルにおける有効フレームの最終ビットを表します。両方のチャンネルをパースしたら、ソフトウェアはADS-Bの有効なデータ・フレームをビット・ストリームから抽出し、そのデータをMATLAB Coderによって生成されたデコード用の関数に引き渡します。デコード用の関数は、ADS-Bのデータ・フレームと現在の位置の緯度/経度を、航空機の座標を計算する際の入力として使用します。なお、現在の位置の緯度/経度はアプリケーションのパラメータとして指定されています。デコードされたADS-Bデータは、Simulinkモデルを使用した場合と同じように表示されます。

ADS-Bデータをデコードするためのアプリケーションは、Linux環境下でmakefileを使用してビルドします。ADIはこのアプリケーションのソース・コードとmakefileをGitHubリポジトリで提供しています[16]。

以上で、MathWorks社のHDL CoderとMATLAB Coderを使用してADS-Bモデルから生成したHDL/Cコードをプラットフォームに配備することができました。次に行うべきことは、システムの機能を確認し、結果を評価することです。

システムの検証

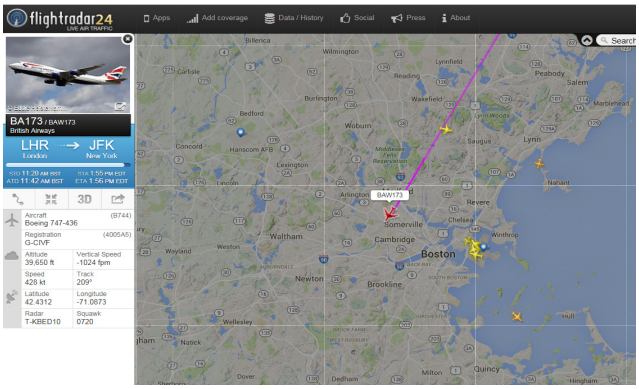
システムの機能を検証するために、まずはAD-FMCOMMS3-EBZの1つの受信ポートと1つの送信ポートの間をループバックの形で接続し、シミュレーションで使用したのと同じADS-B信号を送信します。そのデータを受信してデコードすれば、SDRプラットフォームで実行したアルゴリズムの出力がシミュレーション結果と一致することを確認できます。図11に示したのは、ADS-Bデータをデコードするアプリケーションの出力結果です。これを見ると、Part 3に示した取得済みのデータによるHILシミュレーションの結果と同じであることがわかります。これにより、システムが期待どおりに動作しており、現実のデータを処理する準備が整っているという確証が得られます。

```
Aircraft ID: 400927 is at altitude 39000
Aircraft ID: 400927 is at latitude 42.324, longitude -71.143

Aircraft ID: 400927 is travelling at 468.363107 knots
Direction West at 230.000000 knots, direction South at 408.000000 knots
Aircraft ID: 400927 is going Up at 0.000000 feet/min
```

図11. ループバックによるテストの結果

実際のフィールド・テストでは、マサチューセッツ州ネイティックにあるMathWorks社本社の屋外にSDRプラットフォームで構築したレシーバを配置しました。そして、受信したライブ信号をシステムでデコードすることにより得られたADS-Bの情報と、航空機のリアルタイム追跡ウェブ・サイト（flightradar24.comなど）で提供される情報とを比較しました。その結果、開発したシステムは、アンテナの見通し線上にある航空機から受信した信号を正しくデコードしていることが確認できました。図12に、システムによって検出した航空機の情報と、オンラインで得られる航空機の追跡情報を示しました。両者を比較すると、デコード用のアルゴリズムによって、正しい航空機ID、高度、速度、緯度/経度が得られていることがわかります。




```
Aircraft ID: 4005a5 is at altitude 40300
Aircraft ID: 4005a5 is at latitude 42.398, longitude -71.112

Aircraft ID: 4005a5 is travelling at 427.375713 knots
Direction West at 205.000000 knots, direction South at 375.000000 knots
Aircraft ID: 4005a5 is going Down at 1216.000000 feet/min

Aircraft ID: 4005a5 is at altitude 40275
Aircraft ID: 4005a5 is at latitude 42.396, longitude -71.113

Aircraft ID: 4005a5 is at altitude 40250
Aircraft ID: 4005a5 is at latitude 42.393, longitude -71.115

Aircraft ID: 4005a5 is travelling at 428.253430 knots
Direction West at 205.000000 knots, direction South at 376.000000 knots
Aircraft ID: 4005a5 is going Down at 1344.000000 feet/min

Aircraft ID: 4005a5 is at altitude 40150
Aircraft ID: 4005a5 is at latitude 42.386, longitude -71.121

Aircraft ID: 4005a5 is at altitude 40025
Aircraft ID: 4005a5 is at latitude 42.375, longitude -71.128
```

図12. ライブ・データによるテストの結果

まとめ

本シリーズでは、モデル・ベース設計を活用し、SDRシステムのシミュレーションから製造までのすべての作業の実施方法を4部構成で示しました。シリーズ全体で、ハードウェアへの実装を前提としたSimulinkモデルによって、ADS-B信号のデコード機能を開発するためのすべての工程を網羅しています。シミュレーション用のモデルを設計し、記録済みのADS-Bメッセージをデコードできることを実証したうえで、SDRに対応するハードウェア・プラットフォームによって取得したライブ・データを使い、そのモデルの検証を行いました。この作業により、モデルだけでなく、SDRプラットフォームのアナログ・フロントエンドやレシーバ用のデジタル・シグナル・チェーンの設定が正しいことも検証できます。また、プラットフォームがADS-B信号の受信用に正しくチューニングされていることの確証も得られました。その後、Zynq SoCのプロセッシング・システムとプログラマブル・ロジックで実行する機能にモデルを分割し、それらを最適化したうえでHDLとC言語のコードを自動生成しました。最後に、HDL/CコードをSDRプラットフォームに適用し、民間航空機からのライブ信号を使ってシステムの機能を確認しました。以上により、MathWorks社のモデル化ツールとコード生成ツール、Zynq SoCをベースとしたSDRプラットフォームを使用して、完全に機能するSDRシステムを設計するための手順を示すことができました。

本シリーズで例として取り上げたシステムは、モデル・ベース設計のワークフローと、プログラマブルで集積度の高い無線用ハードウェアであるADIのRFアジャイル・トランシーバIC（AD9361や「AD9364」）を組み合わせることによって、従来の設計手法を適用するよりも迅速かつ低コストで、適切に動作するプロトタイプを開発できることを示しています。このプロトタイプは、以下に示す要因から、ほぼ障害に直面することなく、比較的短期間で構築することができました。

- 適切なHDL/Cコードの生成を可能にするMATLABとSimulinkにより、ADS-B用のレシーバのモデルを構築する能力
- ハードウェア/ソフトウェアを統合するために必要な多くの作業を自動化するHDLワークフロー アドバイザーの機能
- SDR用のプロトタイプを構築するためのその他の統合作業を支援するライブラリ（libiioなど）
- MathWorks社とADIが提供する製品のヘルプ・ドキュメントや技術サポート

ADS-Bは比較的シンプルな規格なので、SDR用プロトタイプの開発にモデル・ベースの手法を適用する方法を例としては最適でした。モデル・ベース設計とZynq SoCをベースとするSDRプラットフォームを採用すれば、これよりも格段に複雑で高度なQPSK（4位相偏移変調）、QAM（直角位相振幅変調）、LTEに対応するSDRシステムも、本シリーズで示したワークフローに従って開発できるはずです。

参考文献

[1] Di Pu, Andrei Cozma, Tom Hill 「製造までの4つのステップ：モデル・ベース設計で実現するソフトウェア無線、Part 1：ADI/Xilinx社のSDR向けラピッド・プロトタイピング用プラットフォーム――その機能、メリット、開発ツールについて学ぶ」Analog Dialogue 49-09

[2] Mike Donovan, Andrei Cozma, Di Pu 「製造までの4つのステップ：モデル・ベース設計で実現するソフトウェア無線、Part 2：MATLABとSimulinkによるモードS信号の検出とデコード」Analog Dialogue 49-10

[3] Di Pu, Andrei Cozma 「製造までの4つのステップ：モデル・ベース設計で実現するソフトウェア無線、Part 3：HILによるモードS信号のデコード用アルゴリズムの検証」 Analog Dialogue 49-11

[4] Analog Devices GitHub repository（ADIのGitHubリポジトリ）

[5] HDL Coder

[6] HDL Coder Block Support（サポートされているブロック）

[7] MATLAB Coder

[8] MATLAB Toolboxes

[9] MATLAB Code Generation Readiness Tool（コード生成の準備状態ツール）

[10] AD-FMCOMMS3-EBZ User Guide（AD-FMCOMMS3-EBZユーザー・ガイド）

[11] Xilinx Zynq-7000 All Programmable SoC ZC706 Evaluation Kit（Xilinx Zynq-7000 All Programmable SoC用の評価キット「ZC706」）

[12] AD-FMCOMMS2-EBZ/AD-FMCOMMS3-EBZ/AD-FMCOMMS4-EBZ HDL/AD-FMCOMMS5-EBZ HDL Reference Design（AD-FMCOMMS2-EBZ/AD-FMCOMMS3-EBZ/AD-FMCOMMS4-EBZ/AD-FMCOMMS5-EBZのHDLリファレンス設計）

[13] Analog Devices BSP for MathWorks HDL Workflow Advisor（ADIが提供するMathWorks HDLワークフロー アドバイザー向けBSP）

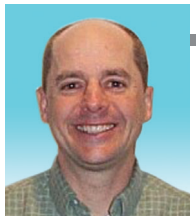
[14] Board and Reference Design Registration System（ボードとリファレンス設計の登録システム）

[15] What Is Libiio?（Libiioとは何か？）

[16] MathWorks Targeting Models—ADSB（MathWorksターゲットモデル-ADSB）

著者：

Mike Donovan（mike.donovan@mathworks.com）は、MathWorks社のアプリケーション・エンジニアリング・グループでマネージャーを務めています。バックネル大学で電気工学の学士号を、コネチカット大学で修士号を取得しています。MathWorks社に入社する前は、レーダー/衛星通信システム分野、ブロードバンド通信分野の業務に従事していました。



Mike Donovan

Andrei Cozma（andrei.cozma@analog.com）は、ADIのエンジニアリング・マネージャーとしてシステム・レベルのリファレンス設計の開発を支援しています。産業用オートメーションと情報科学に関する学士号に加えて、電子工学と電気通信工学の博士号を取得しています。モーター制御、産業用オートメーション、ソフトウェア無線（SDR）、電気通信など、さまざまな分野にわたって設計/開発プロジェクトに従事した経験を持ちます。

Andrei Cozma

この著者が執筆した他の技術文書

FPGAベースのシステムで、モーター制御の性能を向上

Analog Dialogue 49-03

Di Pu（di.pu@analog.com）は、ADIのシステム・モデリング・アプリケーション・エンジニアです。ソフトウェア無線（SDR）に対応するプラットフォームやシステムの設計/開発をサポートしています。特にMathWorks社と密に連携することで、両社の顧客が抱える課題の解決に取り組んでいます。2007年に中国南京にある南京理工大学（NJUST）で理学士の学位を取得しています。また、マサチューセッツ州ウースターにあるウースター工科大学（WPI）で、2009年に電気工学の修士号、2013年に博士号を取得しました。WPIでは、2013年の「Sigma Xi Research Award for Doctoral Dissertation」を受賞しています。

Di Pu