アプリケーション・レポート
JAJA491–December 2015

KNX システムにおける MSPマイコン の活用

Damian Szmulewicz

ABSTRACT

このアプリケーション・レポートは、KNX システムについて紹介しており、MSP マイクロコントローラ（MCU）上で KNX アプリケーションを作成するために使用可能なリソースについて説明しています。KNX の概要は、読者が、KNX システム全体の基本理解を高めることを可能にします。KNX 仕様の詳細な情報は、KNX 関連 Web サイト（knx.org）でご覧いただけます。また、このアプリケーション・レポートは、ステップバイステップの例を使用して、KNX アプリケーションの開発に使用可能なソフトウェアおよびハードウェアの説明を提供して、MSP MCU のプロジェクト作成、アプリケーションのデバッグ、およびシステム・テスト用ツールを紹介します。

目次

List of Figures

List of Tables

1 概要

オートメーション・システムは、あらゆる規模や複雑さのビルディング設備において、人気および重要性がいよいよ高まってきています。利便性、安全性およびエネルギ効率が、ビルディング製品におけるインテリジェント監視および制御のニーズを推進する、重要な要素です。照明やブラインドの制御から、複雑な冷暖房空調設備（HVAC）やエネルギ計測管理システムまで、居住用ビルディングや商業用ビルディングは、より高性能なオートメーション・ソリューションを搭載するようになっています。この傾向は、世界中のメーカーが毎年何千もの製品を市場に発表する原動力となっています。

次の、3 つの主要コンポーネントが、一般的なビルディング・オートメーション・システムを構成します:

- センサ
- 通信チャネル
- アクチュエータ

一連のセンサが、環境からデータを収集し、処理します（温度、湿度、その他）。この知覚情報に基づいて、メッセージが通信チャネルを経由してネットワーク上のさまざまな部分に送信され、そこでアクチュエータが作用を開始します（ACC のオン/オフ切り替え、など）。

煙検出器、侵入センサ、およびアラームなどの安全システムについて考えてみましょう。煙や侵入者が検知されると、アラームはメッセージを受け取り、大きな音を発生させます。アラームは、煙検出器や侵入者検知器から受信するメッセージを正しく解釈する必要があります。しかしながら、煙検出器が通信プロトコル A を使用する A メーカー製であり、一方、侵入センサが通信プロトコル B を使用する B メーカー製であるとすれば何が起きるでしょうか? アラームは、どのようにして両方を正しく解釈できるでしょうか? 何百もの煙検出器メーカーが、それぞれ独自の通信プロトコルを使用したとすれば、どうなるでしょうか? すべてのオートメーション・コンポーネント間の共通言語は、それらの相互運用性に不可欠です。ここで、KNX が必要となります。

2 KNX の概要

KNX はホームおよびビルディング・オートメーション用の、全世界的通信標準です。KNX 協会が KNX 標準を所有しています。1999 年に、欧州インスタレーション・バス協会（EIBA）、欧州ホーム・システム協会（EHSA）、および、BatiBUS Club International（BCI）のメンバーが協力して、KNX 協会を設立しました。KNX は国際標準（ISO/IEC 14543-3）、欧州標準（CENELEC EN 50090 および CEN EN 13321-1）、中国標準（GB/T 20965）として承認されています。KNX は通信言語を定義するだけではなく、シームレスな相互運用性を確実にするために要求される、一連のツールや認定も規定しています。

KNX 協会によって概説されている通り（http://knx.org/media/docs/Flyers/KNX-Introduction-Flyer/KNX-Introduction-Flyer_en.pdf）、KNX システムは以下のメリットをもたらします。

- KNX は国際標準です（将来性が保証されます）。
- KNX は、製品認定によって製品の相互運用性とネットワーク機能を保証します。
- KNX は、メーカーに ISO 9001 の適合を要求することによって、高い製品品質を保証しています。
- KNX には、独自のメーカーである、独立した Engineering Tool Software（ETS）があります。
- KNX は、ホームおよびビルディング制御の、あらゆるアプリケーション分野に使用できます。
- KNX は、さまざまな種類のビルディングに使用できます。
- KNX は、数種類の通信媒体をサポートしています。
- KNX は、ゲートウェイを使用して、他のシステムと組み合わせることができます。
- KNX は、ハードウェアテクノロジやソフトウェアテクノロジには依存していません。

KNX は、25 年以上にわたり、ますます増え続ける開発者によって使用されている、定評あるプロトコルです。4 万社以上の KNX 敷設業者が世界中に存在します。360 社以上のメーカーが、KNX 認定機器を製造しています。10 万種類以上のさまざまな KNX 製品が、7 千種類以上の製品ファミリから市場に提供されています。

この文書は、KNX 通信システムの概要を紹介していますが、KNX 協会から購入可能な KNX 完全仕様書（http://www.knx.org/in/knx/technology/specifications/how-to-order/index.php 参照）の代替とすることを目的とするものではありません。

2.1 KNX コンポーネント

図 1 は、簡単な KNX システムを構成する、8 個の主要コンポーネントを示しています:

1. 物理層
2. 通信スタック
3. アプリケーション・プログラム
4. ETS 製品のオンライン構成と設置
5. メーカー・ツールのオフライン構成
6. USB-KNX インタフェース
7. KNX 認定済み電源
8. KNX バス

以降のセクションでは、それぞれのコンポーネントの詳細について説明します。

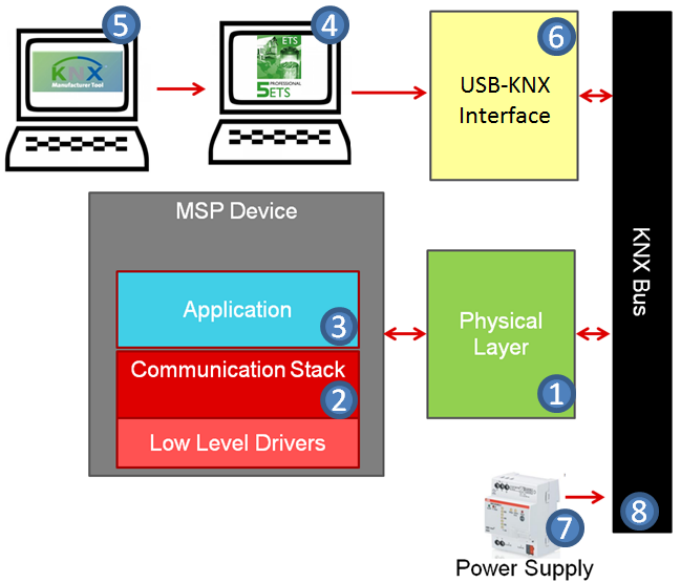


図 1. KNX コンポーネント

2.1.1 物理層

物理層（PHY）の必要性を理解するには、バス信号を理解しなければなりません。図 2 にある通り、KNX は 4 つのタイプの媒体をサポートします。この文書では、ツイストペア媒体を解説します。他のすべての媒体の仕様に関する情報は、www.knx.org を参照してください。

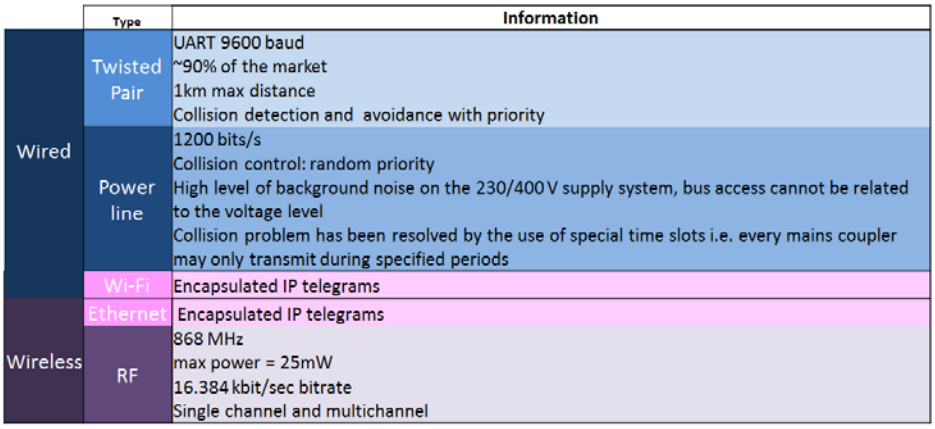


図 2. KNX 媒体

ツイストペア媒体は、20～30 V で作動する、公称電圧 24 V の差動電圧から構成されます。バス上の KNX 信号は、次の 2 つの状態の 1 つを取ります：ロジック 0 またはロジック 1。ロジック 1 は、バス電圧の 直流レベルで、I^2C バスのロジック 1 と類似しています（ここでは、アイドル状態と 1 の間はラインは同じです）。ロジック 0 で、KNX バスのアクションが発生します。ロジック 0 の符号化は、2 つのステージで発生します。最初のステージは、アクティブ・パルスと呼ばれます。アクティブ・パルスは、バス電圧における約 6～9 V の電圧降下です。この電圧ディップは 35 µs 間続き、差動ラインからのトランスミッタ吸い込み電流によって発生します。

それぞれのアクティブ・パルスには、直流レベルより大きい、バス電圧の急激なジャンプが続きます。この急激なジャンプには、69 µs 間持続する直流レベルへの指数関数的減衰が続きます。このパルスはイコライザ・パルスです。通常、チョーク・インジケータが KNX 電源装置に組み込まれます。このチョーク・インジケータは、正のイコライザ・パルスを発生させます。トランスミッタは、正常なメッセージ伝送のために、適正なレベルとタイミングを保証する必要があります。図 3 は、KNX ツイストペア・ライン上の信号符号化を示しています。

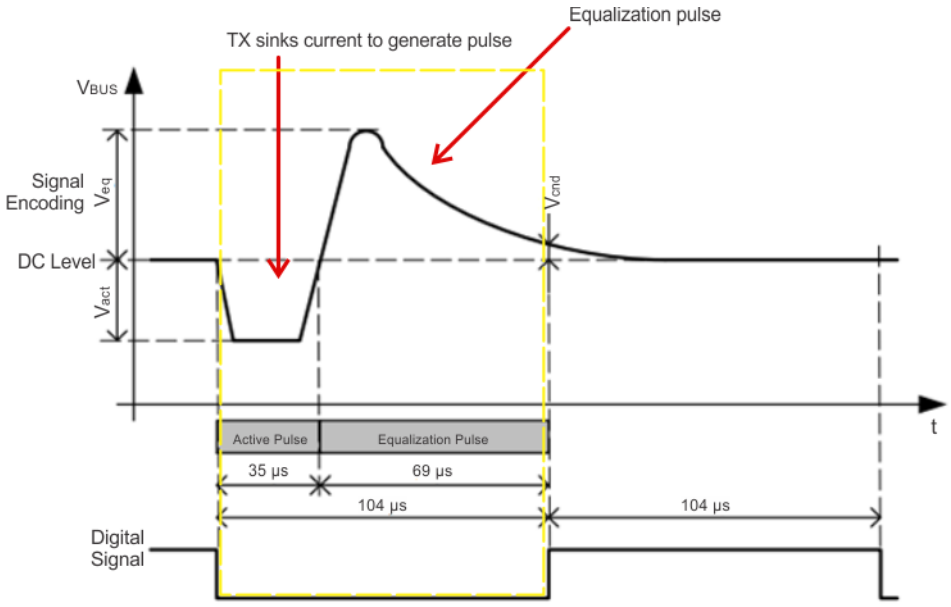


図 3. ツイストペア用の KNX 信号符号化

レシーバ側には、ヒステリシスがある差動コンパレータが必要です。レシーバは、アクティブ・パルスの開始と終了を検出します。アクティブ・パルス開始の検出しきい値は、-0.45 V（標準）で平均バス電圧より低くなっています。アクティブ・パルス終了の検出しきい値は、-0.2 V（標準）です。また、バスから入力コンパレータまでの抑制干渉への内部バンドギャップも必要です。

KNX バスの電圧レベル要件を前提として、MCU 外部のアナログ・コンポーネントが必要です。PHY は、その一部を、通信を可能にするアナログ回路から構成されます。KNX デバイスはバス駆動です。PHY は、アプリケーション MCU 用に安定した DC 3.3 V を出力する、電圧レギュレータを搭載しています。

KNX システムには、通信用の信号レベルを可能にする、アナログ PHY があります。一部の PHY オプションは、アプリケーション MCU からディジタル機能を開放するデジタル・コンポーネントを統合します。これには、衝突制御、反復、確認応答、パリティ、チェックサムが含まれます。PHY に、このようなデジタル機能のためのロジックが含まれない場合は、アプリケーション MCU がそれらを実装する必要があります。

PHY のデジタル機能の重要さを理解するために、衝突処理動作を考えてみましょう。送信デバイスは、送信の間、バスを恒常的に監視する必要があります。ロジック 1 はラインのアイドル状態と同じであるため、別のデバイスから送信されるロジック 0 のみが検出可能です。2 つ以上のデバイスが同時に送信しようとすると、衝突が発生します。0 を送信するデバイスは続行することができますが、1 を送信するデバイスは待機する必要があります。このメカニズムおよび厳格な時間要件が、衝突検知と処理を保証します。

市場では、アナログ専用インタフェース用およびアナログ+デジタル・インタフェース用のいくつかの PHY オプションが入手できます。このようなインタフェースの間にある、大きなトレードオフは複雑さとコストです。アナログ+デジタル PHY は、アプリケーション MCU に最もシンプルなソリューションを提供しますが、アナログのみのソリューションよりコストが高くなります。MSP は、認証ベンダーからの両方のタイプの PHY をサポートします。このようなアナログのみのハードウェア・インタフェースは、ビット・ベース・インタフェースであり、アナログ+デジタルインタフェースは *TP-UART インタフェース*です。この文書は、これらの用語を使用します。

2.1.2 通信スタック

KNX バスでのメッセージは、テレグラムと呼ばれるパケットで送信されます。図 4 に示すように、厳密なタイミング要件を考慮する必要があります。ボタンを押すことによってイベントがトリガされた場合について考えます。t1 = 5.2 ms 以上の間バスが使用されない場合は、送信が開始されることになります。このタイミング要件が満たされると、テレグラムが送信されます。送信が完了すると、受信デバイスは時間 t2 = 1.56 ms 以内にテレグラムの受信を確認する必要があります。トランスミッタが宛先のデバイスから受信確認を受信しないと、トランスミッタは再送信します。

図 4. テレグラム送信と ACK の間のタイミング要件

情報は、図 5 に示すように、8 ビットの文字で、データの同期およびエラー・コントロール用のオーバーヘッドと共に送信されます。スタート・ビットは、データ・バイトの始まりを示します。8 ビットのデータ、パリティ・ビット、ストップ・ビット、2 ビットのポーズがこれに続きます。データの各バイトを送信するには、13 ビットが必要です。

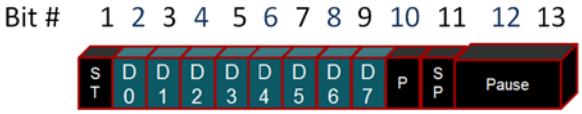


図 5. TP KNX バスでのデータ・フォーマット

図 6 に、テレグラムが 7 フィールドで構成されていることを示しています。セクション 2.1.2.1 ～ 2.1.2.8 で、テレグラムの各フィールドおよび受信確認のコーディングについて説明します。

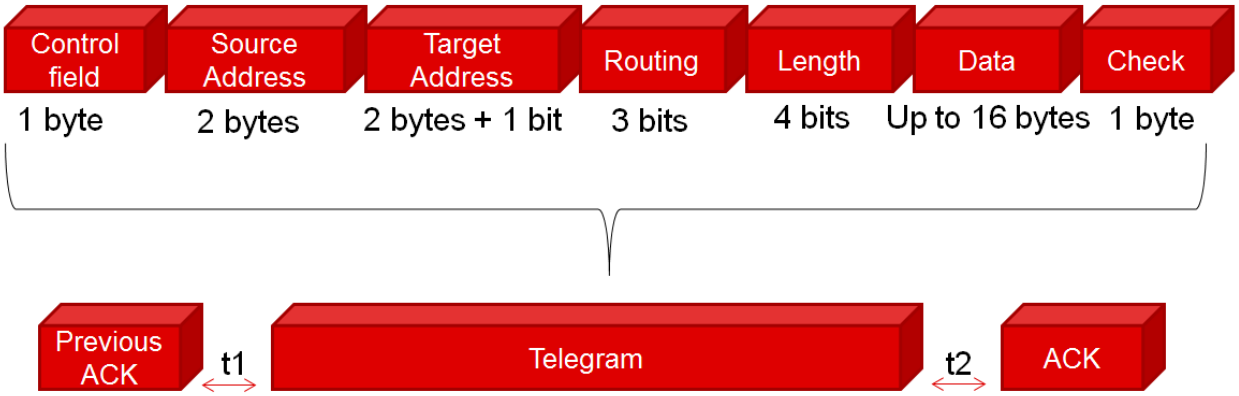


図 6. テレグラム・フィールド

2.1.2.1 *ACK*:

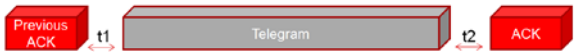


図 7 に、受信確認が 1 バイトのデータであり、3 つの異なる応答を意味することを示しています。

- BUSY
- NACK
- ACK

D7	D6	D5	D4	D3	D2	D1	D0	Description
1	1	0	0	0	0	0	0	BUSY = device unable to process new information
0	0	0	0	1	1	0	0	NACK = Reception incorrect
1	1	0	0	1	1	0	0	ACK = Reception correct

図 7. KNX ACK バイトの説明

2.1.2.2 制御フィールド:

制御フィールドは 1 バイトで、さまざまな目的に使用されます。アドレス指定されたバス・デバイスの一つが NACK を返して送信が繰り返される場合、コントロール・バイトでの繰り返しビットが 0 に設定され、これが再送信であることが示されます。繰り返しビットにより、受信デバイスはコマンドを 1 回だけ実行することが保証されます。コントロール・バイトにより、メッセージの優先順位も設定されます。図 8 に制御フィールドの機能説明が示されています。

D7	D6	D5	D4	D3	D2	D1	D0	Description
1	0	R	1	P	P	0	0	PP = TX priority, R = repeate, others = fixed
				0	0			System functions (highest priority)
				1	0			Alarm functions (alarm)
				0	1			High operational priority (normal)
				1	1			Low operational priority (auto)
		0						Repeat

図 8. コントロール・バイトのビット・エンコーディング

2.1.2.3 ソース・アドレス：

KNX ネットワークの各デバイスには、2 バイトのアドレスがあります。KNX アドレス指定を理解するには、KNX トポロジを考慮してください。図 9 に、KNX ネットワークの概要を示します。デバイス・レベルで開始して、1 ライン当たり最大 256 KNX デバイス（Dn）まで接続できます。ケーブル長を増やすには、各ラインにライン・リピータ（LRn）を追加します。システムでさらに多くのデバイスが使用される場合、メイン・ラインを介してライン・カプラを使用し、最大 15 ラインまでを一緒に接続して、3840 デバイスを有効にすることができます。最大 15 ラインでエリアが形成されます。バックボーン・ラインにより、KNX ツイストペア・バスが拡張されます。最大 15 エリアがバックボーン・カプラ（BCn）を使用して相互接続でき、KNX ネットワーク当り合計最大 57600 デバイスに対応できます。各ラインおよびライン・セグメントには、独自の電源が必要です。電源仕様により、システムでのデバイスの実際の最大数が制限されます。

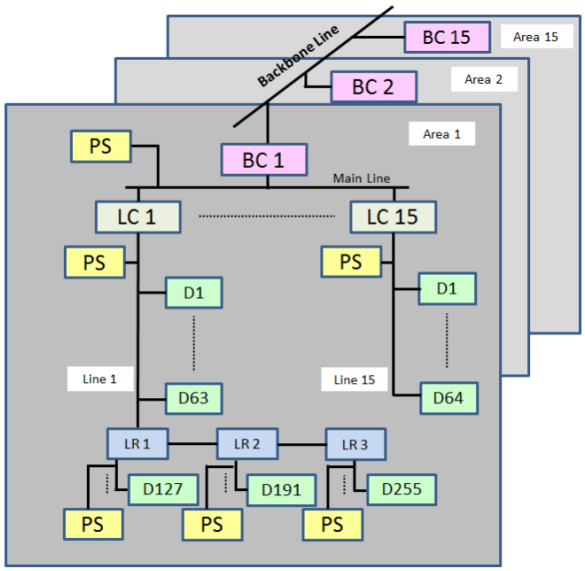


図 9. KNX トポロジ：BCn = バックボーン・カプラ、LCn = ライン・カプラ、LRn = ライン・リピータ、PS = 電源、Dn = KNX デバイスまたはノード

KNX トポロジの詳細については、*KNX TP1 トポロジ*（http://www.knx.org/fileadmin/template/documents/downloads_support_menu/KNX_tutor_seminar_page/basic_documentation/Topology_E1212c.pdf）を参照してください。

ネットワークで特定のデバイスを識別するには、エリア、ラインおよびデバイスの番号がソース・アドレス・フィールドにある必要があります。図 9 に、デバイス・アドレス D1 のみが与えられている場合、そのトランスミッタはライン 1 での D1 またはライン 15 での D1 の可能性があります。同様に、エリア番号を指定する必要があります。ソースアドレスには、送信デバイスの情報一式が含まれています。図 10 に示されているように、下位 8 ビットは固有のバス・デバイス・アドレスです。アドレスの最上位バイト（MSByte）の下位ニブルによりライン番号が指定され、最上位ニブルにエリア番号が含まれています。

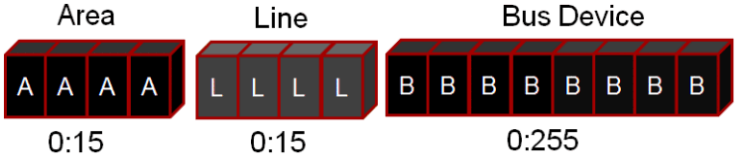


図 10. KNX ソース・アドレスには、エリア、ライン、およびバス・デバイスの番号が含まれる

ライン $_{10}$ およびエリア 3 上に個別アドレス 20_{10} を持つデバイスについて考えます。するとソース・アドレスが、図 11 に示されているように、3/10/20 と指定されます。

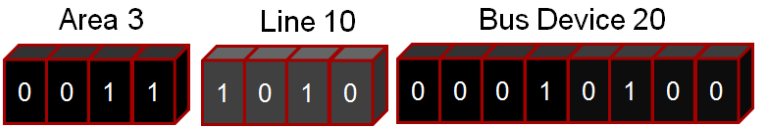


図 11. ソース・アドレス例

2.1.2.4 ターゲット・アドレス:

ターゲット・アドレスには、送信されたメッセージを処理する目的のデバイス・アドレスが含まれています。図 12 に、KNX ネットワーク上のデバイスのエリア、ライン、および固有アドレスもこのアドレスに含まれていることが示されていますが、このアドレスが単一のノードを意図したものか複数のデバイスを意図したものかを指定する追加のビットがあります。追加のビットは、KNX システムの有用な機能です。室内の電球 20 個を制御するために 1 個のスイッチを使用するシステムについて考えます。各電球には固有アドレスがあり、各ターゲット・デバイスをアドレス指定するにはスイッチは同じメッセージを 20 回送信すること必要があります。グループ・ビットを 1 に設定することにより、固有アドレスは無視され、指定されたエリアの指定されたラインでの複数のデバイスが、そのメッセージを処理します。単一のメッセージで 20 個の電球を制御できるようになります。

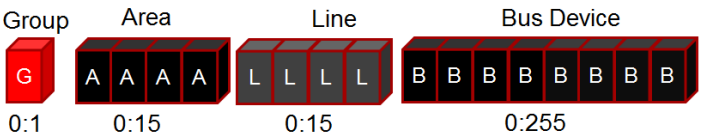


図 12. KNX ターゲット・アドレス

ターゲット・アドレスは、ポイント・ツー・ポイントの接続の固有アドレスです（コネクション型通信）。マルチキャストまたはブロードキャスト・アドレシング（コネクションレス型通信）には、グループ・アドレスと呼ばれるアドレスはターゲット（レシーバ）アドレスとして使用され、図 13 に示す構造になります。

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	D7
	Main Group				Subgroup											T
	Main Group				Middle Group			Sub Group								

図 13. 15 ビット（ビット D15 は予約ビット）でコード化されたグループ・アドレス

最初の行は、グループ・アドレスを 2つの階層（メイン/サブ）で示す方法を示し、2 行目は 3つの階層（メイン/中間/サブ）を示しています。バスでの 2 レベルおよび 3 レベルのグループ・アドレスには、物理的な差異はありません。

2.1.2.5 ルーティング・ビット:

ルーティング・ビットは 3 ビット・カウンタを形成し、KNX ネットワークでメッセージをどこまで送信できるかを制限します。ルーティング値はトランスミッタによって初期化され、システムのカプラによって変更されます。各カプラはルーティング・カウンタをデクリメントさせ、値がゼロより大きい限り、テレグラムを渡します。カウンタがゼロに等しい場合、メッセージはカプラによって送信されません。この機能によってさまざまなライン間の短絡が抑制され、バスでのメッセージがループされつづけることを防止します。

ルーター（ライン・カプラ）のフィルタ・テーブルにより、このフィルタ処理が行われます。このフィルタ処理を無効にするには、ルーティング・カウンタを 7 に等しくします。ルーティング番号が 7 の場合、カプラは、ルーティング・ビットをデクリメントせずに送信を続行します。

2.1.2.6 長さフィールド:

長さフィールドには、データ・フィールドに多数の有用なデータがあります。データの長さは送信されるデータ・サイズによって異なり、1 ビットと 16 バイトの間で変えることができます。レシーバは長さフィールドを使用してデータ受信が適切であることを確認します。長さフィールドを計算するとき、データ・ビットのみが考慮され、データ・フローや制御ビット（パリティ、開始、停止など）は考慮されません。

2.1.2.7 データ・フィールド:

データ・フィールドには、ランプをオンなどのテレグラムの実際のデータが含まれます。メッセージは標準化されたデータ・ポイント・タイプにエンコードされ、デバイスの相互運用性が確保されます。データ・ポイント・タイプにより、すべてのデバイスの同じ言語での通信が確保されます。1 組の電球を制御するスイッチを、例として取り上げます。電球には、オンとオフの 2 つの状態があるだけです。KNX データ・ポイント・タイプのリストに、DPT_Switch と呼ばれるタイプがあります。図 14 に、DPT_Switch およびその他の 1 ビット・データ・タイプを示します。DPT_Switch は、オンまたはオフの状態を示す 1 ビット・メッセージです。このタイプは照明スイッチに使用できます。

Format:	1 bit: B_1
octet nr	1
field names	b
encoding	B
Range:	b = {0,1}
Unit:	None.
Resol.:	(not applicable)

Datapoint Types

ID:	Name:	Encoding: b		
1.001	DPT_Switch	0	=	Off
		1	=	On
1.002	DPT_Bool	0	=	False
		1	=	True
1.003	DPT_Enable	0	=	Disable
		1	=	Enable
1.004	DPT_Ramp	0	=	No ramp
		1	=	Ramp
1.005	DPT_Alarm	0	=	No alarm
		1	=	Alarm
1.006	DPT_BinaryValue	0	=	Low
		1	=	High
1.007	DPT_Step	0	=	Decrease
		1	=	Increase
1.008	DPT_UpDown	0	=	Up
		1	=	Down
1.009	DPT_OpenClose	0	=	Open
		1	=	Close

図 14. KNX データポイント・タイプの例

アプリケーションには、さまざまなデータ・ポイント・タイプが必要です。データ・ポイント・タイプのリストについては、*Interworking*（http://www.knx.org/fileadmin/template/documents/downloads_support_menu/KNX_tutor_seminar_page/Advanced_documentation/05_Interworking_E1209.pdf）を参照してください。

2.1.2.8 確認フィールド:

KNX 通信プロトコルは、2 つのレベルのエラー確認を提供します。1 つ目はバイト・レベルです。図 15 に示すように、テレグラムでのデータの各バイトには、専用の偶数パリティ・ビットがあります。つまり、パリティ・ビット P は 0 または 1 の値を取り、すべてのビット(D0:D7 および P)の二進和を 0 に等しくします。

図 15. KNX テレグラムの各バイトの偶数パリティ

テレグラムの最後のバイトは(確認フィールドと呼ばれる)、2 回目の確認が行われます。このフィールドは、テレグラムのすべての文字の各ビット位置での奇数パリティを確認することによって、作成されます。図 16 に示すように、3 バイトのデータがあることを考慮します。各バイトには、データ・フィールド内に独自の偶数パリティ・ビットがあります。チェック・バイトの各ビットは、そのビット位置に対するすべてのデータ・バイトの各ビットの合計に基づいて、二進和が 1 に等しくなるように、計算されます。

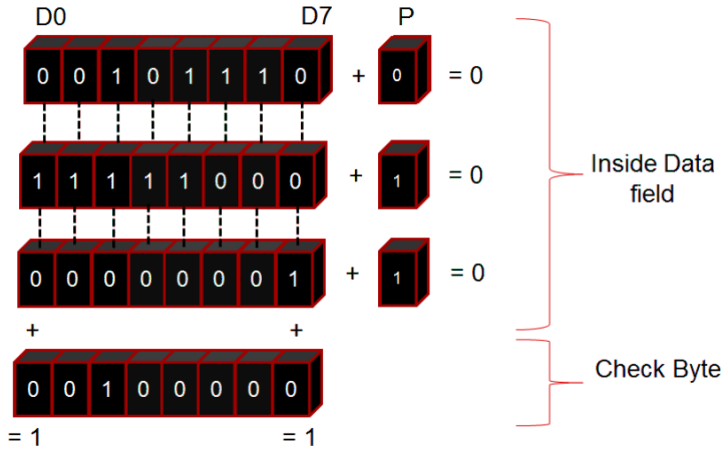


図 16. バイトおよびテレグラム・レベルのエラー確認

2.1.3 アプリケーション

KNX 通信スタックと共にアプリケーション・ソフトウェアは、本製品の特定の機能を実行および可能にします。選択した PHY に基づき、異なるデバイス・リソースを使用できない場合があります。例えば、アナログ・プラス・デジタル PHY は、このアプリケーションに利用できなくなっている UART インターフェースを使用しています。最適なデバイスとスタック・オプションの選択には、アプリケーションに必要なリソースの初期の評価が必須です。

2.1.4 ETS ツール

ETS はメーカーに依存しない構成ソフトウェア・ツールです。ETS は、KNX システムでのホームおよびビルディング・オートメーションのインストールするために使用できる、唯一の構成ツールです。KNX 協会により、ETS は以下の利点を提供します（http://www.knx.org/in/software/ets/about/index.php?navid=948232948232 を参照）。

- ETS ソフトウェアの KNX 規格への最大の互換性が保証されています。
- すべての KNX メーカーからの認定された製品による製品データベースを、ETS にインポートできます。
- ETS の、以前の ETS バージョンの製品データおよびプロジェクトへの下位互換性（ETS2 にまで遡る）により、作業結果が保護され、編集が可能になります。
- すべての設計者およびインストーラは、各 KNX プロジェクトに対して同じ ETS ツールを使用し、各 KNX 認定デバイスを使用します。（高信頼度のデータ交換が保証されます。）

ETS には、ホームまたはビルディング・オートメーションでオートメーション・システムを作成するための、認定された KNX 製品のデータベースが含まれています。空間的特性およびデバイス・アドレスを割り当てる機能により、ETS はシステム・インストレーションを簡易化し、メーカー相互の互換性を維持します。ETS は KNX バスを介してデバイスを直接構成します。コンピュータをバスに接続する特殊なハードウェアが必要です。2.1.6 で、このハードウェアについて説明します。新しい KNX 製品を ETS で構成するには、メーカー・ツールでデータベース・エントリを作成する必要があります。

ETS の詳細情報、チュートリアル、ETS デモ・バージョンのダウンロードについては、ETS 学習キャンパスを http://wbt5.knx.org で参照してください。

2.1.5 メーカー・ツール

KNX 製品は、認定されており、インストールに関するデータベース・エントリとして利用できる必要があります。メーカー・ツールにより、開発者はこのようなエントリを自身の製品用に作成できます。メーカー・ツールは、ETS 製品エントリを作成およびテストし、製品に KNX 協会の認定を受けるためにも必要です。製品が認定されると、メーカーは、インストーラが使用するダウンロード可能な製品カタログとして、このエントリを提供します。

2.1.6 USB-KNX インターフェース

ETS を使用して KNX バスでデバイスを構成するには、コンピュータはツイストペア・バスを介して通信できる必要があります。この目的に、図 17 に示す、Tapko 製の UIM-KNX 42 という名前のデバイスを使用できます。UIM-KNX 42 により、PC と KNX バスの間の双方向データ接続が確立されます。このデバイスにより、アドレッシング、パラメータ設定、可視化、プロトコル作成、バス・デバイス診断が可能になります。USB コネクタは、KNX バスから電気的に絶縁されています。この KNX–USB インターフェースにより、バスにある各デバイスをアドレス指定できます。KNX–USB インターフェースと接続されているデバイスの間の通信は、フレキシブルな共通 EMI プロトコルにより、UIM-KNX 42 デバイスを使用して処理されます。このプロトコルは、現在および将来のアプリケーション用に設計されています。

図 17. PC-デバイス通信用の USB-KNX インターフェース

2.1.7 認定されている電源

図 9 に示すように、KNX トポロジの各ラインには、認定されている電源が必要です。電源はネットワークへ電力を供給し、通信にも使用されます。図 18 に、認定されている電源の例を示します。電源は通常 30 V 定格であり、図 3 のイコライゼーション・パルスに使用されるチョーク・インダクタがこれに含まれています。さまざまな電流定格の電源は市場で広く入手できます。電源選択にヘルプが必要な場合は、E2E™に関する TI 専門家（http://e2e.ti.com/support/microcontrollers/msp430/）にお問い合わせください。

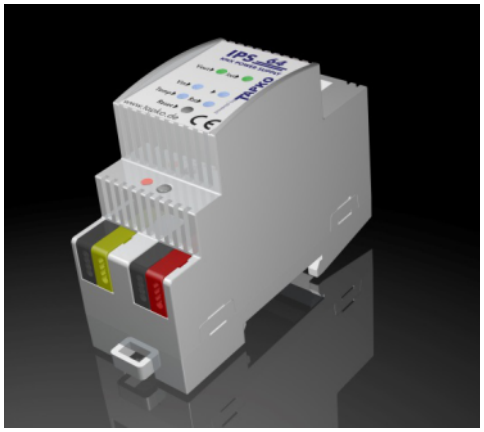


図 18. KNX 認定済み電源

3 MSP-Tapko 製品

TI は、お客様が KNX を使用してビルディング・オートメーション・ソリューションを開発できるようにするため、Tapko Technologiesをパートナーとしています。このセクションでは、KNX に対応可能な MSP ホームおよびビルディング・オートメーション・アプリケーションの開発開始に利用できる、ハードウェアおよびソフトウェアについて説明します。ハードウェアおよびソフトウェア・ソリューションは、さまざまなニーズに利用できます。需要に応じて、最も費用効果が高いソリューションをが選択できます。このドキュメントは、オプションの概要を説明します。KNX のさらなるサポートについては、MSP 低消費電力 MCU フォーラム（http://e2e.ti.com/support/microcontrollers/msp430/）にお問い合わせください。

3.1 オプション 1：KIMaip

製品を市場に出す最速の方法は、Tapko 製 KIMaip インターフェース製品とアプリケーションを実行する低消費電力 MSP マイコンと組み合わせて使用することです。KIMaip は、KNX バスに接続するための使い易いインターフェース・モジュールです。アプリケーション・コントローラから KIMaip モジュールへのアクセスは、I²C バスを介して達成されます。KIMaip は、認定された KNX 通信スタック（KAIstack）を実行するマイコン、および KNX バス（KAIphys）への高性能 KNX インターフェースで構成されます。この設計により、バスからの高出力電源の直接使用が可能になります。このモジュールはアプリケーション・コントローラに直接接続することを目的とし、デバイスのベースとして最適化されます。このモジュールは、ガルバニック絶縁の必要がないアプリケーションに適しています。KIMaip モジュールでは、KNX 機能の実装が簡単にし、アプリケーション開発のオーバーヘッドを除外することで、市場に出す時間を短縮しています。図 19 に、KIMaip を使用するときのシステム図が示されています。

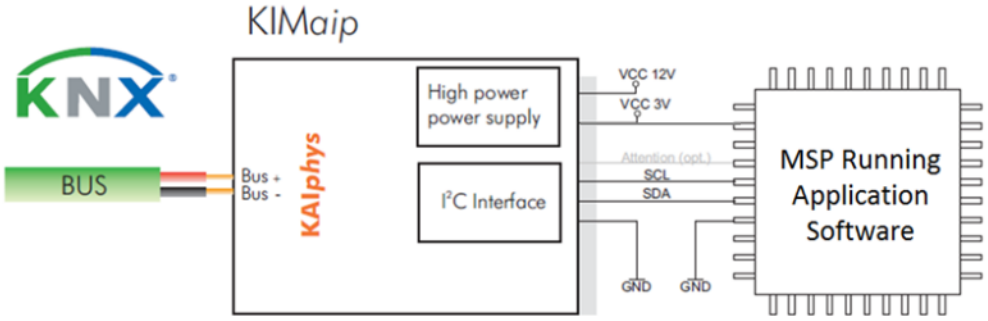


図 19. KIMaip オプション

KIMaip インターフェース製品には、以下の特性があります。

- 提供する KIMaip バス・モジュールはスレーブであり、MSP コントローラはマスターです。
- I²C のメッセージのバイナリ・データを表示します。
- グループ通信およびインターフェース・オブジェクトへのアクセスを提供します。
- ETS 設定可能なパラメータ・エリアへのアクセスを提供します。
- オブジェクト・データは、外部ユーザー・アプリケーション・コントローラに保管されます。
- オプションの *KNX データ受信アテンション*のピンを提供します。

KIMaip 機能の一部には、以下があります。

- 高効率 KNX 物理層 KAIphys は、アプリケーション MCU 用に 12 V で最大 35 mA、または 3 V で 30 mA の出力電力を提供します。
- アプリケーション・コントローラおよびアプリケーション・ハードウェアのモジュール電源への直接接続を提供します。
- 無料ダウンロードできる汎用 ETS データベース・エントリを介した設定を提供します。
- 253 のグループ・オブジェクトを提供します。

KIMaip は、すべての KNX データ・タイプをサポートしており、データをオブジェクト・データと共に受信すると即座に表示します。アプリケーション・コントローラは、どのような KNX 固有コードも実行しません。KIMaip は、I²C を介して受信した直後にテレグラムを生成します。KIMaip は、ビット・ベース（アナログのみ）でも、TP-UART（アナログ・プラス・デジタル）インターフェースでもありません。

3.2 オプション 2：KAIphys および KAIstack を KAIlink-BIT と併用

KAIphys は、ビット・ベースのインターフェース・ソリューションです。KAIphys は、標準コンポーネントに基づく最初の KNX インターフェースであり、すべてのユーザーが利用でき、ASIC なしで実装されます。KAIphys は、高レベルの信号処理品質とバスからの高エネルギを組み合わせた使用での高度の柔軟性を可能にする、優れたソリューションです。この製品により、コスト効率の高い KNX ソリューションが可能になります。顧客固有の ASIC が必要でない事実、および各種モジュールでのさまざまなオプションからの選択の可能性により、将来の製品の合理化およびハードウェア設計に対する卓越した可能性が提供されます。KAIphys により、高い柔軟性と共に最高度の性能が提供されます。革新的な回路は、KNX 対応バス・デバイス用のテクノロジ・プラットフォーム KNX 高性能インターフェース（KAI）のハードウェア・コンポーネントです。KAIphys および KAIstack は、KNX デバイス一式の基礎を形成します。KAI のハードウェアおよびソフトウェア・コンポーネントは、条件に適応できます。KAIphys は KNX 認定取得済みです。KAIphys が含まれているソリューションを実装するには、モジュールのライセンスを Tapko から取得できます。

KAIstack は、KAI のメイン・ソフトウェア・コンポーネントであり、KNX デバイスに必要な機能一式を提供します。換言すると、KAIstack には KNX 標準に必要なエレメントが含まれており、ISO/OSI リファレンス・モデルに密接に依存して実装されたいくつかの異なる構成で認定されています。アプリケーション関連部品、モジュール通信スタック内部部品、メディア依存部品、およびターゲット CPU 関連問題の間には明確な構造上の相違があります。高性能実装メソッドにより、最適化されたリソース使用をもたらす高効率コーディングが可能になります。KAIstack を使用したアプリケーション開発により、スタックの簡単な構成によって設計プロセスでの早期決断の重荷が軽減され、その一方で、システムの信頼性が増大し、デバイスの安定性が向上します。

KAIlink-BIT は、物理的ビット・ベースのインターフェース（KAYphys）をソフトウェア・スタック（KAIstack）に関連付ける、ソフトウェア・ドライバです。KAIstack を KAIlink-BIT と組み合わせることにより、完全なソフトウェア・ソリューションを実現できます。

3.3 オプション 3：TP-UART および KAIstack を KAIlink-UART と併用

SIEMENS TPUART2、ON semi NCN5120、ELMOS E981.03 などのいくつかのハードウェア TP-UART インターフェース・チップ・ソリューションは、市場で入手できます。PHY メディア・モジュールは、各チップ・ソリューションに使用できます。Tapko は、高速プロトタイピングおよび開発を可能にする認定されたプラグ・アンド・プレイ・メディア・モジュールを、提供しています。

3.2 に示すように、KAIstack は KNX 通信スタックです。ソフトウェアの観点から、オプション 2 と TP-UART インターフェース・オプションの間の差異は、物理層とインターフェース接続するドライバです。KAIstack と組み合わせた KAIlink-UART により、TP-UART での KNX 開発のソフトウェア・パケット一式が提供されます。KAIstack および KAIlink の役割をより良く理解するため、図 20 に、これらのソフトウェア・パッケージがどのように KNX ソリューションに適合されているかを示します。KAIstack をビット・ベースの PHY と相互接続するには、KAIlink-BIT が必要です。KAIstack を P-UART PHY と相互接続するには、KAIlink-UART ドライバが必要です。

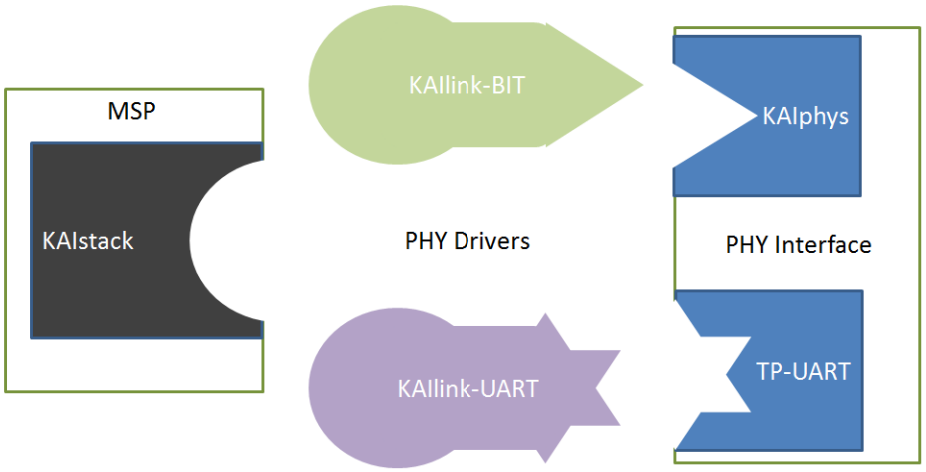


図 20. KNX ソフトウェアおよびハードウェアのエコシステム

3.4 オプション 4：お客様のスタック・ソフトウェアおよびハードウェア

物理層に利用可能なモジュールがオプションではない高容量アプリケーションでは、TI および Tapko は、完全にカスタマイズされた KNX ソリューションを提供できます。このオプションにより、お客様はすべてを特定のアプリケーション用に最適化できますが、コストが掛かります。このオプションの詳細については、E2E フォーラム（http://e2e.ti.com/support/microcontrollers/msp430/）の MSP 専門家にお問い合わせください。

4 初めての MSP での KNX

このセクションでは、KNX 対応アプリケーションの開発を開始するハードウェアおよびソフトウェアについて説明し、新しい KNX アプリケーションを利用可能なツールを使用して作成する方法を示します。

4.1 ハードウェア

図 21 に、KNX の開発に最低限必要なハードウェアを示します。

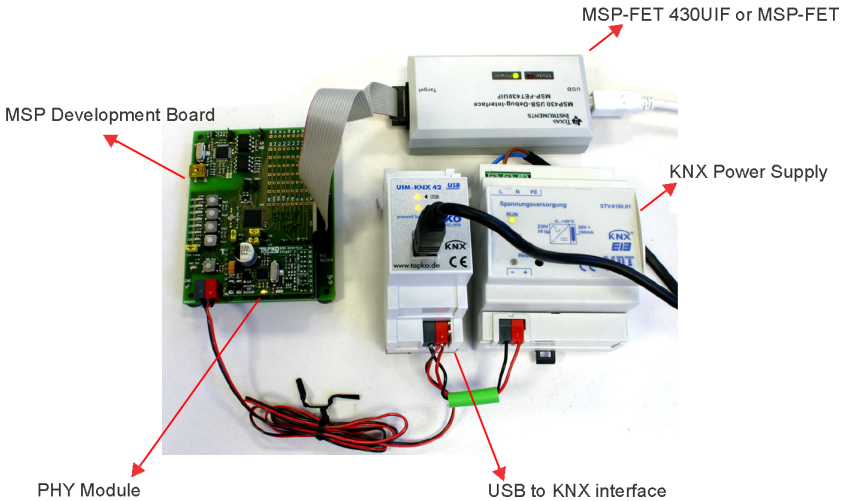


図 21. MSP での KNX 開発のための最低限のハードウェア

図 21 の各コンポーネントの詳細情報を入手できる場所を、表 1 で一覧で示しています。

表 1. ハードウェア・コンポーネント・リソース

コンポーネント	Link
MSP-FET	http://www.ti.com/tool/msp-fet
USB-KNX インターフェース	http://www.tapko.de/uim-knx42.html
MSP 開発ボードと PHY モジュール	http://www.tapko.de/kaistack-for-ti.html
KNX 電源	http://www.tapko.de/contact.html

追加のソフトウェアおよびハードウェア・リソースは、TI Designsまたはアプリケーション・レポートとして入手できることがあります。最新のリソースについては、www.ti.com で KNX を検索してください。

4.2 ソフトウェア

評価を開始するため、Tapko は、KAIstack および KAIlink の無償バージョンを下記の制限付きで提供しています。

- 16 のグループ・アドレス、16 の関連付け、16 の通信オブジェクトのみが提供されます
- インターフェース・オブジェクトは提供されません
- ネットワーク層ルート・カウントは 1 に設定されます
- トランスポート層の繰り返しは提供されません
- ETS が物理アドレスを変更することはできません。
- サポートされるデバイス派生品は 1 つのみです

ソフトウェア・インストーラを http://www.tapko.de/kaistack-for-ti.html からダウンロードします。

KAI デモに加えて、MSP 用 IAR Embedded Workbenchのライセンスが必要です。IAR の期限付き評価ライセンスについては、http://supp.iar.com/Download/SW/?item=EW430-EVAL にお進みください。

注: コード・サイズに起因して、MSP 用フル・バージョンの IAR Embedded Workbenchが必要です。

4.2.1 デモ・ソフトウェアの概要

このセクションでは、Tapko 評価ソフトウェア・パッケージの概要を説明します。KAIstack インストーラは 2 つのディレクトリを作成します。図 22 にインストール画面が示されており、ここで各生成されたフォルダの保存先フォルダを選択できます。

注: KAIstack のディレクトリ・パスに空白スペースを含めることはできません。KAIstack のパスにスペースがある場合、インストーラは正常に完了しますが、後でプロジェクトのコンパイルに失敗します。TI は、KAIstack をドライブのルート(C:\KAIstack_Demo など)にインストールすることを推奨しています。

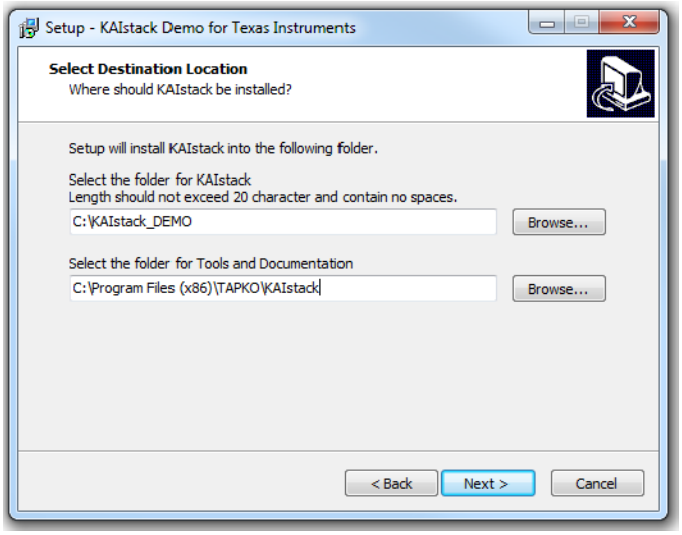


図 22. KAIstack、ツール、ドキュメントのインストール・ディレクトリ

最初のフォルダは KAIstack ディレクトリであり、これには 5 つのディレクトリ、新規プロジェクトの作成に有用な AppWizard、ETS エントリおよびその他の KNX 関連アクティビティの作成でユーザーをガイドする GNU Make ファイルおよびドキュメントが含まれます。See 図 23.

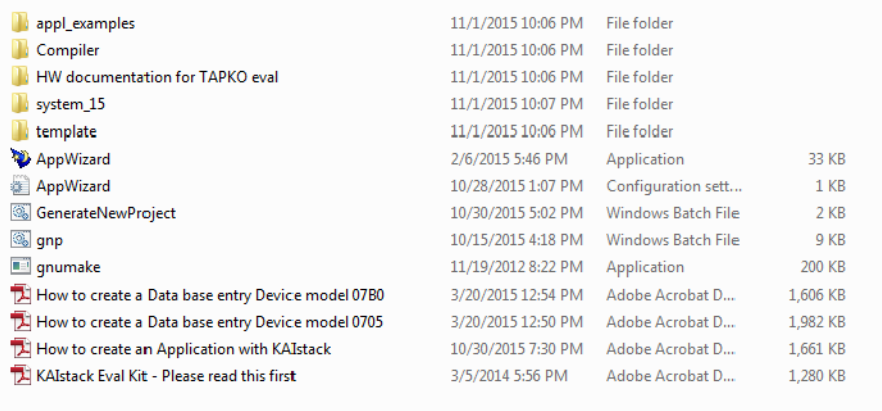


図 23. KAIstack デモ・ディレクトリ構造

AppWizard は、新規 KNX プロジェクトを開始するフレームワークを作成するアプリケーションです。4.3.1 に、AppWizard を使用するステップ・バイ・ステップの手順が示されています。system_15 ディレクトリに、スタックを適切に実行するために必要な、アプリケーションに依存しないファイルが含まれています。TI は、これらのファイルを変更することは推奨しません。コンパイラ・ディレクトリには、IAR によって KNX プロジェクトをコンパイルおよびリンクするために必要なファイルが含まれています。TI は、これらのファイルのいずれを変更することも推奨しません。appl_examples により、機能 KNX プロジェクトを生成できるようにするサンプル・コードが提供されています。このコードは、新規プロジェクトを開発するとき、優れたリファレンスとなります。インストーラの第 2 ディレクトリは、ツールおよびドキュメント用です。図 24 に KAIstack リファレンス・マニュアルを示しています。

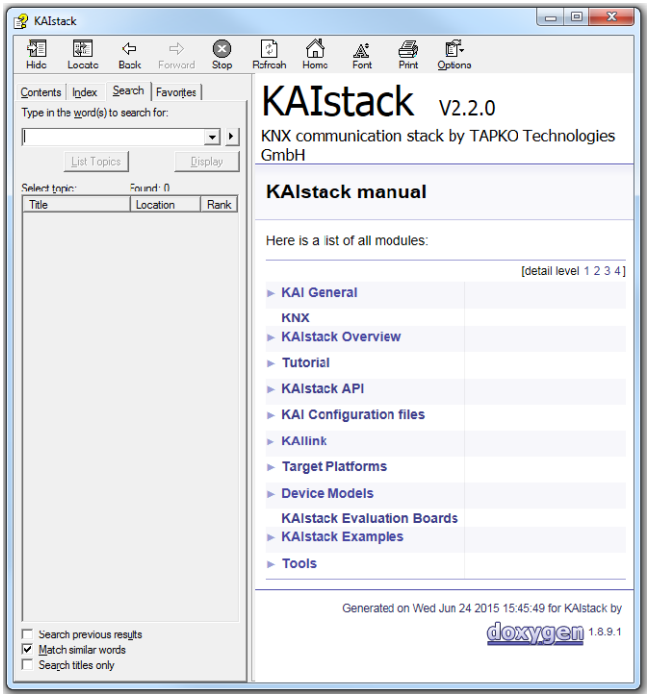


図 24. KAIstack リファレンス・マニュアル

4.2.2 スタック・ソフトウェア・フロー

このセクションでは、図 25 に示されているソフトウェア・フローの概要を説明します。スタックの説明一式については、KAIstack リファレンス・マニュアルでの KAIstack API を参照してください。

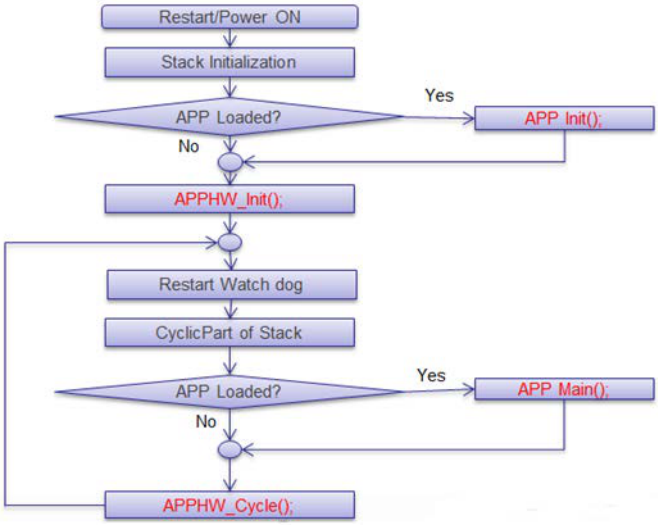


図 25. KAIstack ソフトウェア・フロー

KAIstack を実行するデバイスの電源が投入されると、スタックは初期化されます。この初期化には、KAIlink を含むすべてのローレベル・ドライバが含まれます。アプリケーションがデバイスにロードされると、KAIstack スケジューラは APP_Init() 関数を呼び出します。APP_init() でのユーザー・コードにより、アプリケーション変数が初期化されます。KNX で、デバイス・モデルにより、バスを介してアクセスできるメモリ・エリアが定義されます。これらのメモリ・エリアは、ポインタを使用して物理メモリの範囲にマップされます。このマップされたエリアは、仮想 BCU_RAM メモリです。仮想 BCU_RAM メモリ（RAMflag および通信オブジェクト値）は、APP_Init() が呼び出される前にクリアされます。

KAIstack スケジューラは、APPHW_Init() を呼び出します。APPHW_Init() は、APP_Init() と類似していますが、アプリケーション変数ではなくアプリケーションペリフェラルを初期化します。APPHW_Init() で、タイマ、シリアル・インターフェース、アナログ・フロントエンドを設定できます。初期化が完了すると、ソフトウェア・スケジューラは APP_Main() を周期的に呼び出します。デフォルトでは、スタックは APP_Main() をできるだけ速く呼び出し（数百マイクロ秒程度）、これはバスのアクティビティによって異なります。呼び出し間のこの遅延を遅くできますが、通信オブジェクトがブロックされることなく、アプリケーションがすべてのメッセージに確実に反応させるため、10 ms を超えないようにする必要があります。APP_Init() および APPHW_Init() と同様に、プログラムの周期的な部分には PPHW_Cycle() と APP_main() があります。APPHW_Cycle() は常に呼び出されますが、APP_Main() が呼び出されるのは、アプリケーションがデバイスにロードされて構成されている場合のみです。

4.3 カスタム KNX アプリケーションの作成

KAIstack デモをインストールした後、AppWizard を使用して新規 KNX アプリケーションを作成します。このセクションでは、新規 KNX プロジェクトを作成して、これにアプリケーションを追加するために必要なステップについて説明します。ここでは IAR（http://supp.iar.com/Download/SW/?item=EW430-EVAL）および ETS（http://wbt5.knx.org）をインストールすることを推奨しています。

4.3.1 AppWizard

MSP で KNX プロジェクトの作成を開始する方法は、以下のステップのとおりです。図 26 に AppWizard のインターフェースを示しています。

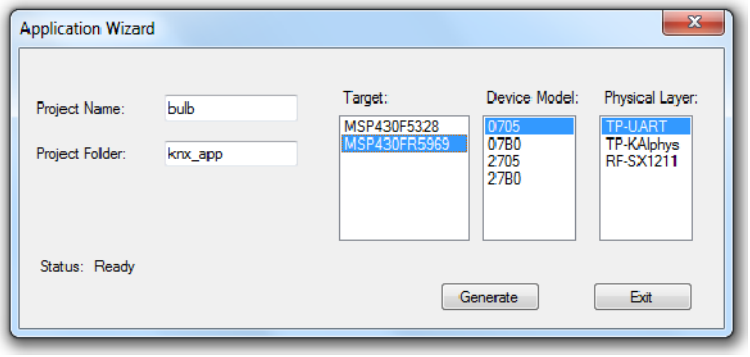


図 26. AppWizard グラフィカル・ユーザー・インターフェイス

1. プロジェクト名とプロジェクト・ディレクトリを選択します。
2. ターゲット・デバイスを選択します。
3. デバイス・モデル 0705 または 07B0 を選択します（両方も 250 データ・ポイントを超える）

注: 2 で始まるDevice Modelは RF インターフェース用であり、本書の範囲外です。Device Modelの詳細については、KAIstack レファレンス・マニュアルでのデバイス・モデルを参照してください。

4. 物理層の選択：TP-UART または TP-KAIphys（ビット・ベース PHY）

注: RF-SX1211 は RF インターッフェース用です。

5. [*生成*] をクリックします。

注: 図 27 に示すように、フォルダ名が含まれているディレクトリが KAIstack フォルダ中に作成されます。このディレクトリには、KNX プロジェクトのコンパイルおよびデバッグに必要なファイルが含まれています。

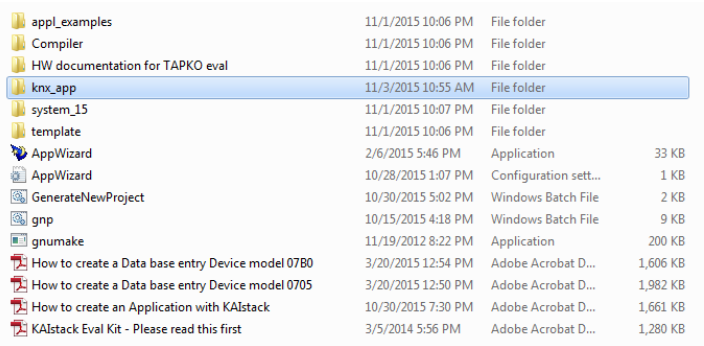


図 27. AppWizard によって生成された knx_app ディレクトリ

6. 図 28 に示すように、以下のディレクトリが新規プロジェクト・フォルダ内に作成されます。
   - dummy
   - output
   - src
   - tmp
   - workspace

   2 つのWindows®コマンド・スクリプト（.cmd）がプロジェクトを構築するために作成されます。

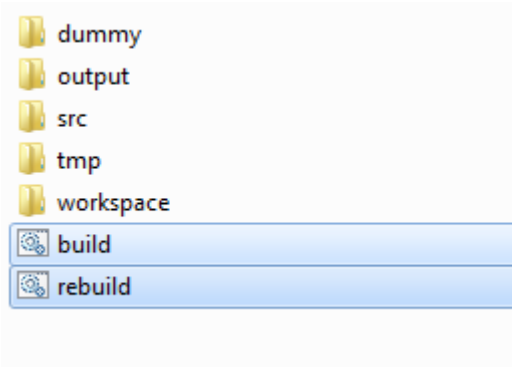


図 28. AppWizard によって生成される knx_app のコンテンツ

4.3.2 生成されたファイル

このセクションでは、各ファイルの目的について説明します。4.3.2.1 は、これらのファイルを変更して新規アプリケーションを作成する方法について説明しています。dummy、output、および tmp ディレクトリは、初期状態では空です。これらのフォルダは、プロジェクトがコンパイルされるときに、ビルドおよび再ビルドコマンド・ファイルによって使用されます。ビルドによって作成された最も重要なファイルは、\output ディレクトリ内にあるデバッグ・ファイル（.d43）です。4.5 に、デバッグ・ファイルの使用について説明します。\scr ディレクトリには、アプリケーションの実行に必要なソース・ファイルおよびヘッダー・ファイルが含まれています。図 29 に、\src 内で作成された 6 つのファイルを示しています。アプリケーション用に、\src にさらにファイルを追加できます。

図 29. src フォルダ内のファイル

4.3.2.1 app.h

このファイルには、KNX が適切にコンパイルするために必要なパラメータの定義が含まれています。最初に一式の識別番号が以下のように定義されます。

- KNX 製品には、KNX 協会が割り当てるメーカー ID（APP_MANUF_ID）が必要です。
- KNX 製品には、製品メーカーが管理するアプリケーション ID（APP_APPL_ID）があります（APP_MANUF_ID と APP_APPL_ID を組み合わせて、そのハードウェアを特定する 6 バイトの固有の値 KNX_HW_TYPE が作成される）。

以下の 2 つの値が、アプリケーション・バージョンおよび製品を購入するための発注型番の追跡を維持するため定義されます。

- バージョン番号（APP_APPL_VERSION）は、製品メーカーが管理します。
- 10 バイトのアプリケーションオーダ番号（APP_ORDER_NR）を指定することができます。この値はシステムによっては使用されませんが、インストーラがプロジェクトで使用するために ETS に表示できます。

通信オブジェクトを追加するために必要な情報が定義されます。通信オブジェクトを理解するため、図 30 に示すオフィス・フロアについて考えてください。

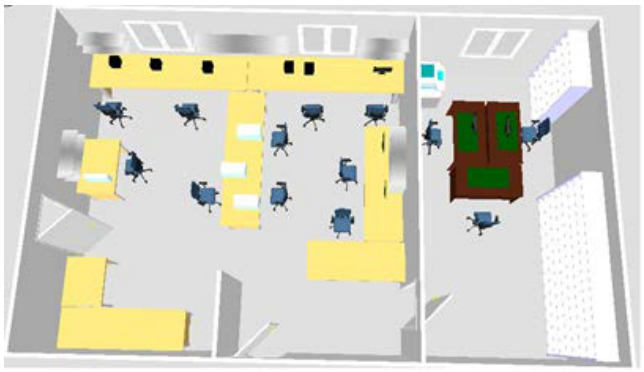


図 30. 通常のオフィス・フロア

オフィス・フロアの各部屋には、その部屋の状態を示すさまざまなセンサがあります。図 31 に示すように、各センサには、その特定のデバイスの状態を定義する変数が 1 つあります。これらの変数はネットワーク変数です。

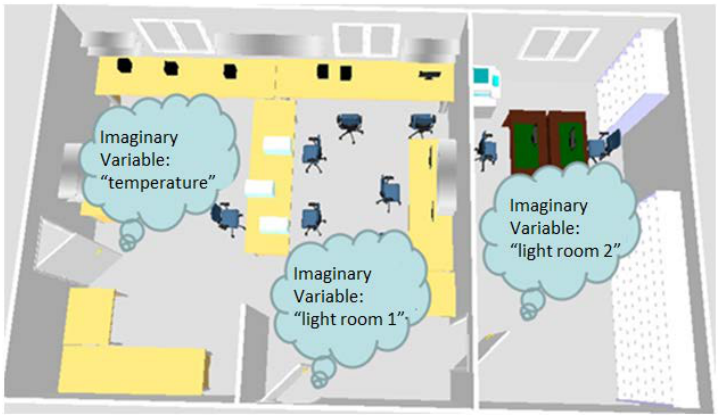


図 31. オフィス・ネットワーク変数

デバイスには特定の仮想変数のコピーが存在することがあります。図 32 に示すように、複数のデバイスに同じ仮想変数の個別のコピーがあることがあります。

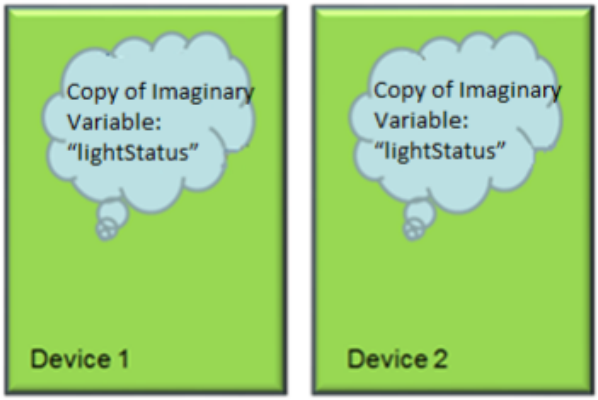


図 32. 仮想変数 *lightStatus* のコピーがある 1 つ以上のデバイス

図 33 に示すように、仮想変数の各コピーは、通信システムによってデバイス全体で同期が維持されます。これらのコピーは通信オブジェクトです。

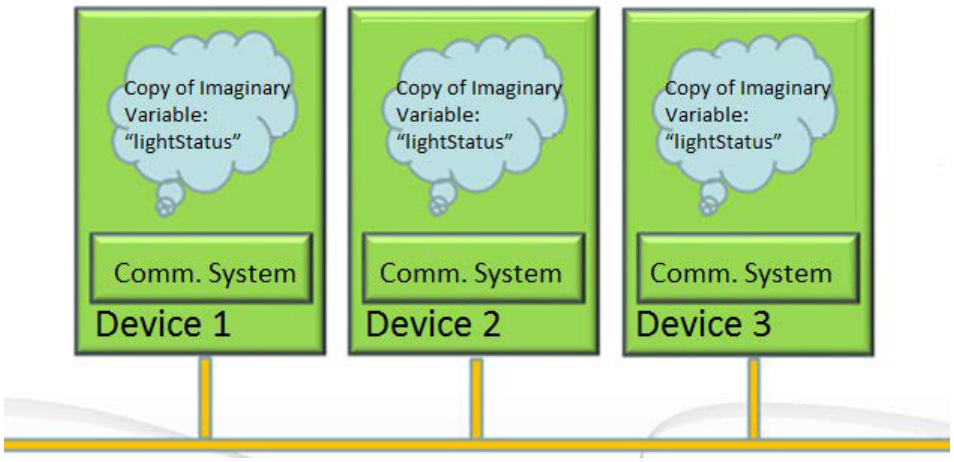


図 33. 通信システムにより、仮想変数の同期が維持されます

5 個の電球と 1 個のスイッチがある部屋を考えてください。スイッチをオンにすると、電球がオンになります。この構成では、5 個の電球が同じグループ通信オブジェクトに属し、スイッチが異なる通信オブジェクトに属します。オブジェクトがグループ化されないと、スイッチは 5 つの個別のメッセージ（各電球に 1 つ）を送信する必要があります。通信オブジェクトの合計数は、APP_objectTabSize に保管されます。4 つの入力オブジェクトと 4 つの出力オブジェクトがあるシステムは、APP_objectTabSize を 8 に定義する必要があります。通信オブジェクトは、BCU-RAM の一部としても定義されます。図 34 に、1 つの入力オブジェクト（in0）と 1 つの出力オブジェクト（out0）の BCU-RAM 構造の例を示します。

```
typedef struct {
  unsigned char ramFlags[APP_objectTabSize];
  unsigned char in0;
  unsigned char out0;
} APP_Ram;
```

図 34. 1 つの入力オブジェクト（in0）と 1 つの出力オブジェクト（out0）の BCU-RAM 構造

4.3.2.2 app_data.c

app_data.c には、メモリ・マッピング用のヘッダー・ファイルが含まれています。一般に、ETS パラメータが app.h で定義されていない限り、このファイルを変更する必要はありません。

4.3.2.3 cotab.h

このファイルは 2 つの主要な目的で使用されます。1 つ目は、デバイス固有アドレス（KNX_DEF_INDIVIDUAL_ADDR）の定義を有効にすることです。このアドレスは、後でインストール中に ETS によって変更されることがあります。このデバイス・アドレスにより、ピア・ツー・ピアの方法でデバイスにアクセスできます。このファイルの 2 番目に重要なタスクは、アプリケーションの通信オブジェクトを定義することです。app.h では、オブジェクトの数と RAM 変数名が定義されます。cotab.h では、通信オブジェクトは、名前、グループ・アドレス、BCU-RAM 構造での変数へのポインタ、データ・タイプ、ETS フラグ、設定フラグを取得します。図 35 に例を示します。名前は、オブジェクトにアクセスするためにアプリケーションで使用され、任意の文字列にすることができます。グループ・アドレスは、図 13 の規約に従って右記の 3 つの値で定義されます：メイン・グループ、中間グループ、サブグループ。

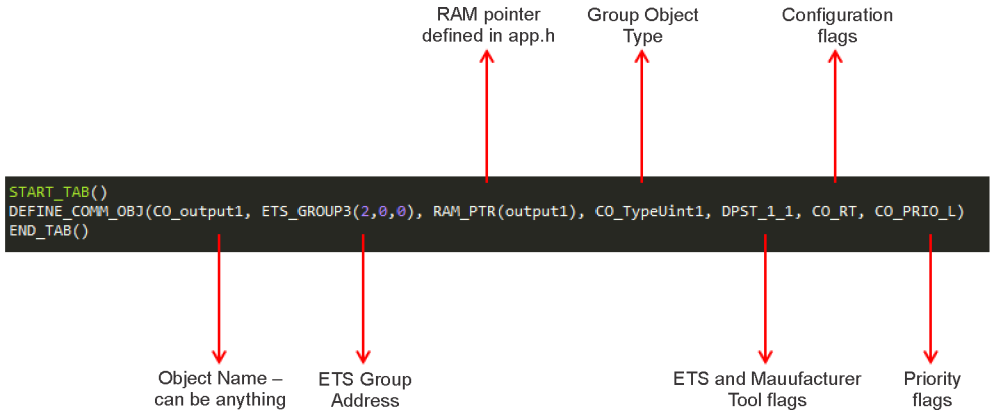


図 35. 通信オブジェクト定義のマクロ

マクロの順序によって、グループ通信オブジェクトの順序が定義されます。関連するアドレスには制約があります。グループアドレスのグループ通信オブジェクトへの割り当ては、一対一です。グループ・アドレスは昇順の必要があります。例えば、ETS_GROUP (2,0,0) は ETS_GROUP (2,0,1) に先行し、ETS_GROUP (1,6,8) に後続する必要があります。この例では、グループ・オブジェクト・タイプは CO_TypeUint1 として定義され、図 14 に示すような 1 ビットの符号なしデータ・タイプに使用できます。グループ・オブジェクト・タイプおよび優先フラグの完全なリストについては、KAIStack マニュアルの「 KAIstack API」→「グループ通信」→「リファレンス」を参照してください。ETS およびメーカーのツール・フラグは、ETS インストール・ファイル（通常は \Program Files (x86)\ETS5\knx_master.xm）の knx_master.xml ファイルに定義されているデータ・ポイント・タイプです。この例では、DPST_1_1 が使用されます。

図 36 では、knx_master.xml の DPT_Switch として、このフラグの定義が示します。ETS およびメーカーのツール・フラグの完全なリストについては、knx_master.xml を参照してください。

```
<DatapointSubtype Id="DPST-1-1" Number="1" Name="DPT_Switch" Text="switch" Default="true">
 <Format>
  <Bit Id="DPST-1-1_F-1" Cleared="Off" Set="On" />
 </Format>
</DatapointSubtype>
```

図 36. DPST_1_1 の DPT_Switch としての knx_master.xml 定義

設定フラグにより、通信オブジェクトの操作のタイプ（読み取り専用など）が定義されます。図 37 に、フラグの完全なリストを示します。

	CO_commEnable	CO_transmitEnable	CO_writeEnable	CO_readResponseEnable	CO_readEnable	
CO_T	X	X				Transmit only
CO_RT	X	X			X	Transmit, Read
CO_W	X		X			Write
CO_WU	X		X	X		Write, ReadResponseUpdate
CO_RWU	X		X	X	X	Write, ReadResponseUpdate, Read
CO_R	X				X	Read

図 37. 通信オブジェクトの設定フラグ

4.3.2.4 main.c

リセット後に、main が呼び出されます。main で周辺機器およびスタックが初期化され、ソフトウェアによって App_main() が周期的に呼び出されます。

4.3.2.5 project.h

このヘッダーにより、デバイス・モード、ターゲット・コンパイラ、デバイス派生品、評価ボード情報、スタック・サイズが定義されます。AppWizard は、ユーザーが選択した構成に基づいて、このファイルを自動的に実装します。

4.3.2.6 bulb.c

Appwizard は、プロジェクト名に基づいてこのファイルに命名します。bulb.c には、初期化およびランタイムの巡回動作のための、図 25 に示すアプリケーション関数が含まれています。bulb.c には、ユーザー・コードがアプリケーション用に追加されます。

注: KAIstack のインストールに含まれるコード例では、文字 KSD_ で始まる関数が使用されます。\appl_examples 内の例で使用されるこれらの関数は、\system_15\targets\TI\msp430_common\KAIstackEval_MSP430xxx で定義されます。

4.3.3 アプリケーションの KNX プロジェクトへの追加

KNX アプリケーションが KNX スイッチのステータスを読み取る簡単な電球の場合、スイッチのステータスが変わると必ずオンまたはオフに変わります。電球にユーザー制御のステータス押しボタンがある場合、ボタンが押されると、電球のステータスがバスを介して送信されます。この例では、ピン 1.1 が押しボタン・スイッチに接続されており、ピン 1.0 が電球（または LED）に接続されているとします。このアプリケーションの一般的なプロジェクトでは、3 つのソース・ファイルと 2 つまたは 3 つのヘッダー・ファイルがあります。以下のファイルを使用していると仮定します。

- init.c
- init.h
- app_main.c
- sensor.c
- sensor.h

init.c には、アプリケーションを実行するために必要なすべての初期化ファンクションが含まれています。void ioInit() を利用できると仮定します。sensor.c には、void bulbStatusUpdate 関数（符号なし文字ステータス）があり、これによって電球のオンまたはオフを切り替えます。

app_main.c は初期化ファンクションを init.c から呼び出し、スイッチからの KNX メッセージを利用できるか、定期的に確認します。メッセージが受信されると、読み取り値が以前の読み取り値と比較され、電球に変更が必要な場合は、電球の状態が変更されます。スイッチが押されると、電球のステータスがバスを介して送信されます。ユーザー押しボタンには、割り込みが使用されます。ヘッダー・ファイルには、app_main.c がこの関数にアクセスするために必要な関数宣言が記載されています。図 38 に、ソース・ファイル用疑似コードを示します。TI は、同様のソース・ファイルおよびヘッダー・ファイルを作成することによって次のステップに従うことを推奨します。

app_main.c

```
#include "init.h"
#include "sensor.h"

int main(void) {

    ioInit();

    unsigned char sendStatus = NO;
    unsigned char bulbStatus = OFF;
    unsigned char bulbStatusNew;

    while(1){

        if message received over KNX bus
        {
            check new status of KNX_switch
            if bulbStatusNew != bulbStatus
            {
                bulbStatusUpdate(KNX_switch value)
            }
        }
        if(sendStatus equals YES)
        {
            sendStatus = NO;
            send a KNX message with the current status of the lightbulb
        }
    }
}

//*************************************************************************
// Interrupt Service Routine - Port 1
//*************************************************************************
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    switch (__even_in_range(P1IV, P1IV_P1IFG7)) {

    case P1IV_NONE:   break;                    //No Interrupt pending
    case P1IV_P1IFG0: break;                    //P1IV P1IFG.0
    case P1IV_P1IFG1:                           //P1IV P1IFG.1 - Button
    // Clear P1.1 IFG
    P1IFG &= ~BIT1;
    // Set sendStatus to trigger a message on the bus
    sendStatus = YES;
    break;
    case P1IV_P1IFG2: break;                    //P1IV P1IFG.2
    case P1IV_P1IFG3: break;                    //P1IV P1IFG.3
    case P1IV_P1IFG4: break;                    //P1IV P1IFG.4
    case P1IV_P1IFG5: break;                    //P1IV P1IFG.5
    case P1IV_P1IFG6: break;                    //P1IV P1IFG.6
    case P1IV_P1IFG7: break;                    //P1IV P1IFG.7
    }

}
```

init.c

```
#include "init.h"

void ioInit(void) {
    // For FRAM devices, disable the GPIO power-on default high-impedance mode
    PM5CTL0 &= ~LOCKLPM5;

    // Alarm button
    // Direction is input
    P1DIR &= ~BIT1;
    // Internal pull-up on P3.4
    P1OUT |= BIT1;
    // Enable pull-up resistor
    P1REN |= BIT1;
    // Lo/Hi edge
    P1IES &= ~BIT1;
    // Clear all P3 interrupt flags
    P1IFG = 0;
    // interrupt enabled
    P1IE |= BIT1;

    // Output lightbulb
    // Direction is output
    P1DIR |= BIT0;
    // Initial state is OFF
    P1OUT &= ~BIT0;

}
```

sensor.c

```
#include "sensor.h"

void bulbStatusUpdate (unsigned char status) {

    if (status == ON)
        P1OUT |= BIT0;
    else
        P1OUT &= ~BIT0;

}
```

図 38. app_main.c、init.c、および sensor.c 用疑似コード

図 39 に、ヘッダー・ファイル用コードを示します。この例では、MSP430FR5969 が仮定されています。

sensor.h

```
#include "msp430fr5969.h"
#define     ON      1
#define     OFF     0


void bulbStatusUpdate (unsigned char status);
```

init.h

```
#include "msp430fr5969.h"
#define     YES     1
#define     NO      0

void ioInit(void);
```

図 39. sensor.h および init.h 用コード

4.3.3.1 KNX ファイルの設定

アプリケーション・コードを KNX プロジェクトに追加するには、4.3.2 の KAIstack ファイルを設定します。開発段階では、app.h の ID に任意の番号を付けることができます。システムのテストに重要なパラメータは、以下のとおりです。

- アドレス・テーブル・サイズ
- 関連表のサイズ
- 通信オブジェクトのサイズ
- BCU-RAM 構造

APP_addrTabSize および APP_assocTabSize は、デバイス・モデル、およびアプリケーションでの通信オブジェクトの数によって異なります。デバイス・モデルにより、割り当て可能なグループ・アドレスの最大数が確立されます。図 40 に、デバイス・モデル 0705 に対する最大数を示します。特定のデバイス・モデル情報については、KAIstack リファレンス・マニュアルを参照してください。

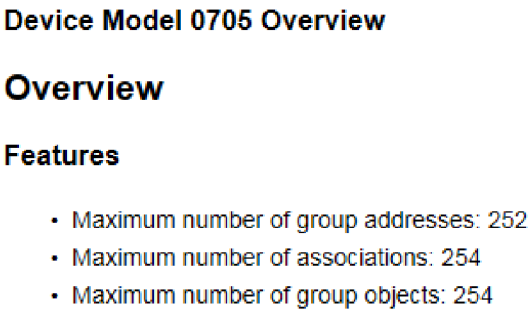


図 40. デバイス・モデル 0705 の最大グループ・アドレス例

テーブルに対して最大数のメモリ割り当てを行うことは、必ずしも必要ではありません。4 つの通信オブジェクトがそのアプリケーションに必要な場合、254 アドレスを予約すると、リソースが静的に浪費されます。通信オブジェクトが 4 つの場合、サイズが等しい 4 つのテーブルで十分です。簡単なスイッチの例には、次の 2 つの通信オブジェクトがあるだけです:バスからのメッセージを受信する入力、およびバス上でメッセージを送信する出力。APP_objectTabSize は、2 に等しい必要があり、アドレス・テーブルの最も効率的なサイズも 2 です。図 41 に、その設定を示します。

```
//*****************************************************************************
// Size of system tables
//*****************************************************************************

// Size of the address table (excluding physical address)
// With this constant the amount of memory for the address table is reserved.
// The maximum allowed size depends on the device model.
#define APP_addrTabSize      2

// Size of the association table
// With this constant the amount of memory for the association table is reserved.
// The maximum size depends on the device model.
#define APP_assocTabSize     2

// Number of group communication objects.
// The maximum size depends on the device model.
// For this example we have 1 output and 1 input,
// so the total number of objects is 2
#define APP_objectTabSize    2
```

図 41. app.h での通信テーブルの設定

BCU-RAM 構造は、図 42 に示すように、通信オブジェクトが含まれるように定義する必要があります。出力通信オブジェクトは statusOut で、入力通信オブジェクトは statusInです。名前は重要ではありませんが、アプリケーションで、通信オブジェクトの値を変更するため、およびそれをバスを介して送信するために使用されます。

```
//*****************************************************************************
// Definition of BCU-RAM
//*****************************************************************************
typedef struct {
  unsigned char ramFlags[APP_objectTabSize];            // RAM-flags
  unsigned char statusIn;                                    // object value: input
  unsigned char statusOut;                                   // object value: output
} APP_Ram;
```

図 42. BCU-RAM には、出力通信オブジェクトの名前が含まれます

注: C99 タイプ（uint8_t など）を使用するには、stdint.h が app.h に含まれている（#include<stdint.h>）必要があります。このファイルはデフォルトでは含まれていません。また、\workspace\gmake の app_make.gmic に以下の行がある必要もあります。

ADD_INCLUDE_PATH += -I $(PATH_APPL)/../../../Compiler/MSP430_IAR6/inc/dlib/c/

図 43 を参照してください。

```
SOURCES_APPL_BASIC = $(PATH_APPL)/bulb.c \
                     $(PATH_APPL)/app_data.c \

SOURCES_APPL_FIX =  $(SOURCES_APPL_BASIC) \
                    $(PATH_APPL)/main.c \

SOURCES_APPL_ETS  = $(PATH_APPL)/app_data.c \

DEPENDENCIES_APPL = $(wildcard $(PATH_APPL)/*.h)

ADD_INCLUDE_PATH += -I $(PATH_APPL)/../../../Compiler/MSP430_IAR6/inc/dlib/c/
```

図 43. stdint.h ファイルを修正するための、ポイント・コンパイラへの ADD_INCLUDE_PATH

cotab.h には以下の変更が必要です。

1. KNX_CURRENT_ADDR_TAB_LEN および KNX_CURRENT_ASSOC_TAB_LEN を、システムでの通信オブジェクトの数と等しくします（図 44 では 2）。

```
/**
defines the current length of the address table
@see APP_addrTabSize
*/
#define KNX_CURRENT_ADDR_TAB_LEN 2

/*'
defines the current length of the association table
@see APP_assocTabSize
*/
#define KNX_CURRENT_ASSOC_TAB_LEN 2
```

図 44. cotab.h での通信オブジェクトの数に基づく仮想アドレス長

2. 通信オブジェクトを宣言します。

この場合、次の 1 ビット・オブジェトが 2 つ必要です：入力用に 1 つおよび出力用に 1 つ。図 45 に、statusIn をグループ・アドレス 2/0/0 の 1 ビット低優先度入力オブジェクトとして設定するためのコード、および statusOut をグループ・アドレス 2/0/1 の 1 ビット低優先度出力オブジェクトとして設定するためのコードを示します。app.h の RAM-BCU 構造での通信オブジェクトの名前は、通信オブジェクトの宣言のための RAM_PTR のパラメータとして使用されます。

```
START_TAB()
DEFINE_COMM_OBJ(CO_statusIn,  ETS_GROUP3(2,0,0), RAM_PTR(statusIn), CO_TypeUint1, DPT_1, CO_RWU, CO_PRIO_L)
DEFINE_COMM_OBJ(CO_statusOut, ETS_GROUP3(2,0,1), RAM_PTR(statusOut), CO_TypeUint1, DPST_1_1, CO_RT, CO_PRIO_L)
END_TAB()
```

図 45. 出力通信オブジェクトの宣言

app_data.c、main.c、または project.h での変更は必要ありません。

4.3.3.2 アプリケーション・ファイルの追加

KNX プロジェクトは、アプリケーション・コードと統合する事ができます。プロジェクトをアプリケーション・コードと統合するには、以下のようにします。

1. 図 46 に示すように、app_main.c を除くすべてのアプリケーション・ファイルを KNX \scr ディレクトリにコピーします。

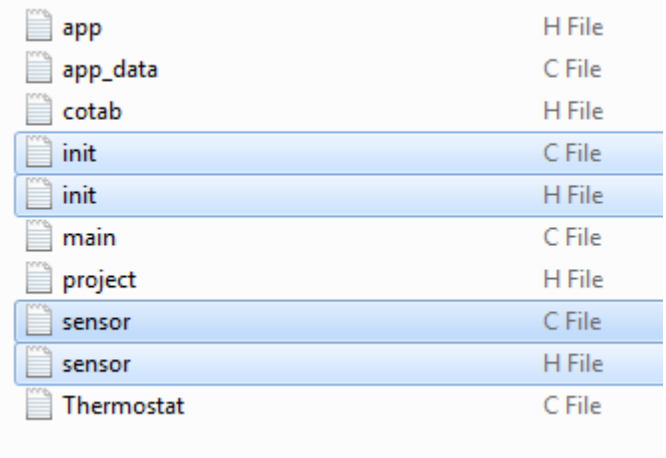


図 46. アプリケーション・ファイルを KNX プロジェクトの src ディレクトリにコピーします

注: ファイルをこのディレクトリに配置するだけでは、コンパイラがこれらを見つけるには不十分です。

2. 図 47 に示すように、\workspace\gmake の app_make.gmic ファイルを変更して SOURCES_APPL_BASIC を更新して、これらのファイルがコンパイラの検索パスに含まれるようにします。

注: init.c および sensor.c が追加されました。アプリケーションのソース・ファイルは、このファイルにリストされる必要があります。

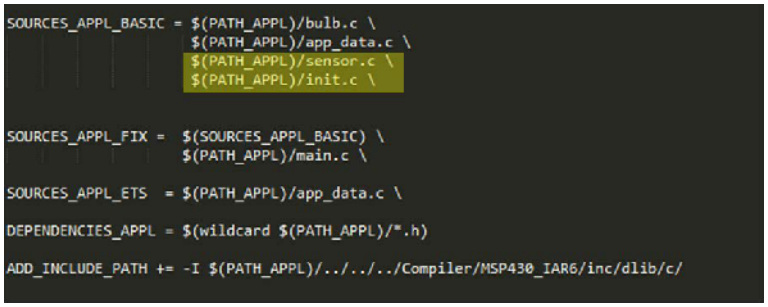


図 47. ソース・ファイルを app_make.gmic に追加

3. app_main.c を KNX プロジェクトに追加します（図 48 を参照）。
   (a) app_main.c からのヘッダー・ファイルを bulb.c に含めて、bulb.c での機能呼び出しがそれぞれのヘッダー・ファイルで確実に宣言されるようにします。
   (b) ハードウェア初期化関数呼び出しを APPHW_Init() 内に配置します。

   注: initIO() は、init.h で宣言される必要があります。

   (c) app_main.c からの変数を bulb.c に追加します。
   (d) APP_Init() の変数および通信オブジェクトを初期化します。
   (e) 巡回コードを APP_main() に追加します。

入力通信オブジェクトが KNX バスのメッセージによって変更された場合、フラグが設定されます。フラグを確認するため、スタックにより、入力オブジェクトをその引数として扱う CheckUpdateFlag 関数が提供されます。前回の確認以降にこのオブジェクトが変更されている場合、この関数は TRUE を返します。変更されていない場合、この関数は FALSE を返します。その入力オブジェクトに対するメッセージが到着しているかを確認するため、if 文を使用できます。すると入力通信オブジェクトの値は、特定の BCU-RAM 値（readInput = OBJ_VALUE → inputObject）にアクセスすることにより、読み取ることができます。メッセージをバスで送信する必要がある場合、出力通信オブジェクト statusOut が変更（OBJ_VALUE → statusOut = 値）されます。関数 SetTransmitRequest() により、メッセージが KNX バスで送信されます。SetTransmitRequest() に渡されるパラメータは、cotab.h で宣言されたオブジェクト CO_statusOut（statusOut ではなく）です。

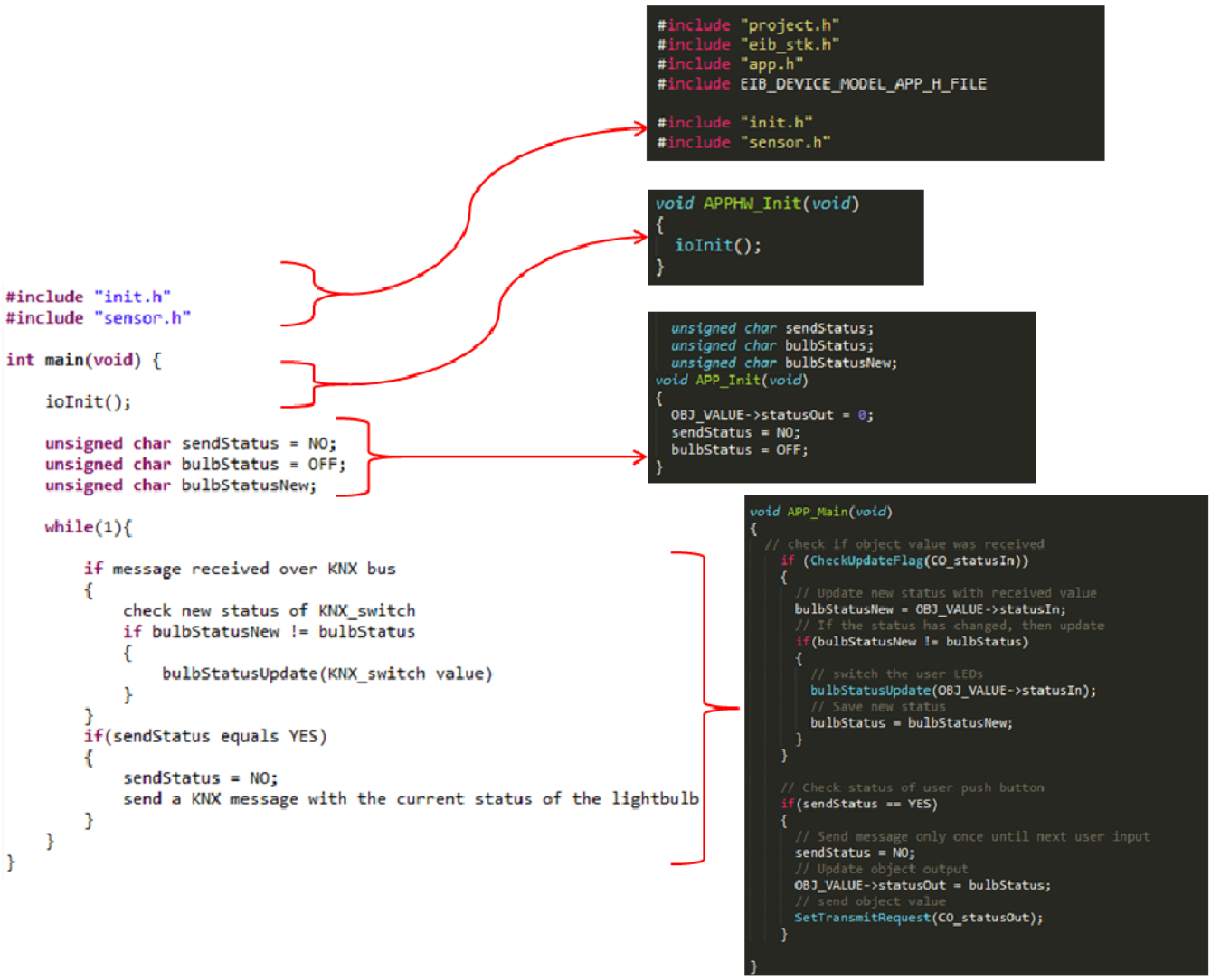


図 48. app_main.c の bulb.c への追加

プロジェクトはコンパイルの準備が完了しています。このセクションを閉じる前に、bulb.c の 2 つの付加的な要素に注意してください。一つ目は App_Save() 関数です。KNX バスで電源が切れたときはいつでも、このイベントを、PHY ボードは IO 信号によって MSP デバイスに通知します。この信号が受信されると、KAIstack により、APP_Save 関数が呼び出されます。APP_Save には、デバイスが遮断される前に実行する必要がある重要コードが含まれている必要があります。1 つの例として、重要変数のメモリへの保存があります。二つ目のポイントは、割り込みの使用です。割り込みは、他の MSP アプリケーションでの場合のように、bulb.c で割り込みサービス・ルーチン（ISR）を宣言することにより使用できます。

図 49 に、この例で説明されているポート割り込みに対するこのような宣言の例を示します。

```
//*****************************************************************************
// Interrupt Service Routine - Port 1
//*****************************************************************************
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    switch (__even_in_range(P1IV, P1IV_P1IFG7)) {

    case P1IV_NONE:   break;                    //No Interrupt pending
    case P1IV_P1IFG0: break;                    //P1IV P1IFG.0
    case P1IV_P1IFG1:                          //P1IV P1IFG.1 - Button
    // Clear P1.1 IFG
    P1IFG &= ~BIT1;
    // Set sendStatus to trigger a message on the bus
    sendStatus = YES;
    break;
    case P1IV_P1IFG2: break;                    //P1IV P1IFG.2
    case P1IV_P1IFG3: break;                    //P1IV P1IFG.3
    case P1IV_P1IFG4: break;                    //P1IV P1IFG.4
    case P1IV_P1IFG5: break;                    //P1IV P1IFG.5
    case P1IV_P1IFG6: break;                    //P1IV P1IFG.6
    case P1IV_P1IFG7: break;                    //P1IV P1IFG.7
    }

}
```

図 49. bulb.c でのポート 1 ISR 宣言

4.4 KNX プロジェクトのコンパイル

KNX プロジェクトは、IAR、CCS などの IDE では構築できません。KNX プロジェクトをコンパイルするには、以下のようにします。

1. AppWizard によって生成されたコマンド・ファイルを使用して、プロジェクトを構築します。
2. 以下のオプションから選択します。
   - オプション 1：build.cmd または rebuild.cmd をクリックします。
   - オプション 2：ファイル・エクスプローラからコマンド・ファイルで実行します。
     (a) プロジェクト・ディレクトリ（この例では \knx_app）に移動して、Windows でコマンド・ラインを開きます。
     (b) Shift キーを押したまま電球ディレクトリを右クリックします。
     (c) [コマンド・ウィンドウをここで開く] を選択します。
     (d) dir を入力します。
     (e) Enter を押して、コマンド・ラインが適切なディレクトリにあることを確認します。
   - 図 50 に示すように、build.cmd がコマンド・ラインにリスト表示されていることを確認します（リスト表示されていない場合、フォルダは不適切です）。

注: このアクションにより、現在のフォルダにあるすべてのファイルとディレクトリがリスト表示されます。

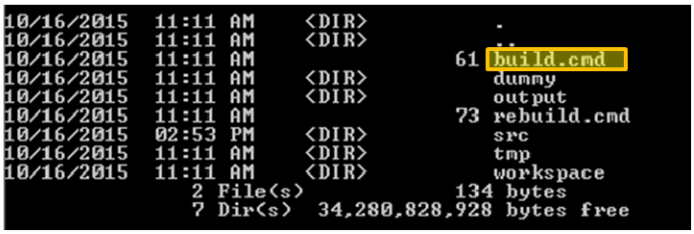


図 50. 電球内の構築プログラムの検索

適切なディレクトリで、プロジェクトを以下のように構築します。

(A) 「*build rebuild*」を入力します。

(B) Enter を押します。

エラーまたは警告が発生すると、これらがリスト表示されます。構築が正常に完了すると、エラーがないことがコマンド・ラインに表示され（図 51 を参照）、デバッグ・ファイル bulb.d43 が \output 内に図 52 のように作成されます。

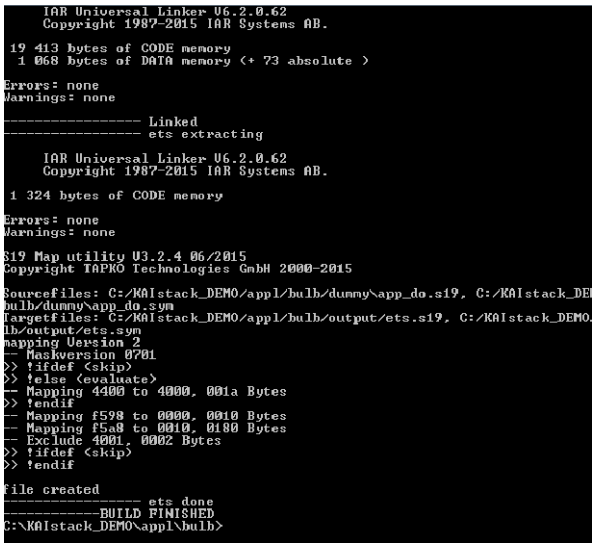


図 51. 構築の正常な完了

図 52. プロジェクト構築後に \output に生成されたファイル

.d43 ファイルは、デバイスのプログラミングおよびデバッグ用です。

4.5 KNX 対応アプリケーションのダウンロードおよびデバッグ

デバッグ・ファイル（.d43）が生成されると、これをデバイスのメモリにダウンロードして、IAR 環境でデバッグできます。このセクションでは、KNX プロジェクトのプログラミングおよびデバッグに必要なステップについて説明します。図 53 は、AppWizard が \bulb\workspace\iar に作成した IAR IDE ワークスペース・ファイル（.eww）です。

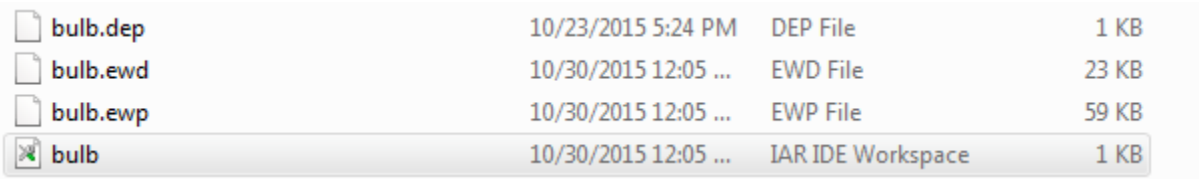


図 53. \workspace\iar 内の IAR ワークスペース・ファイル

KNX プロジェクトをプログラミングおよびデバッグするには、図 53 の IAR IDE ワークスペース・ファイルを開きます。

注: 図 54 に示すように、このアクションによって、IAR が起動し、デバッグ（.d43）ファイルがロードされます。これが IAR の初めての使用である場合、.eww ファイルを開くために IAR を使用するように Windows で指定する必要がある場合があります。.eww ファイルを開くには、以下のようにします。

1. ファイルを右クリックします。
2. [*ファイルを開く...*] を選択します。
3. PC で実行可能 IAR を参照します。

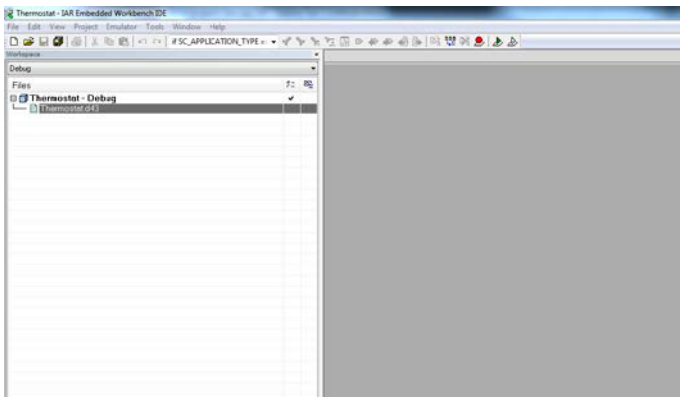


図 54. デバッグ・ファイルが含まれている IAR ワークスペース

デバッグ・ファイルがデフォルトでロードされていない場合、または異なるファイルをロードする場合、以下のようにします。

1. プロジェクトの名前を右クリックします。
2. [*追加*] を選択します。
3. 図 55 に示すように、[*ファイルを追加*] を選択します。これにより Windows エクスプローラが開きます。

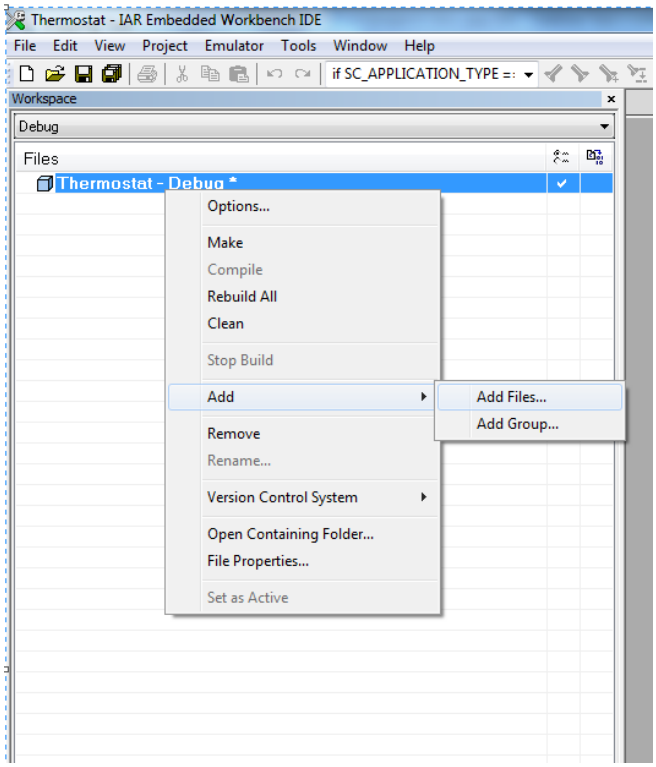


図 55. デバッグ・ファイルのワークスペースへの追加

4. \output に移動します。
5. デバッグ・ファイルを選択します。

注: .d43 ファイルをこのウィンドウに図 56 のように表示するには、ファイル・タイプを [すべてのファイル] に設定する必要があります。

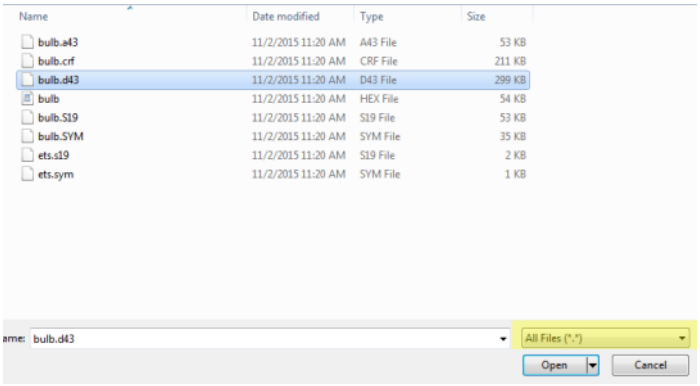


図 56. デバッグ・ファイルの IAR ワークスペースへの手動での追加

インストールされている IAR のバージョンが最新バージョンではない場合、ウィンドウに図 57 のエラーが表示されます。プロジェクトを新規作成して、そのファイルを実行しているバージョンの IAR に追加するには、以下のようにします。

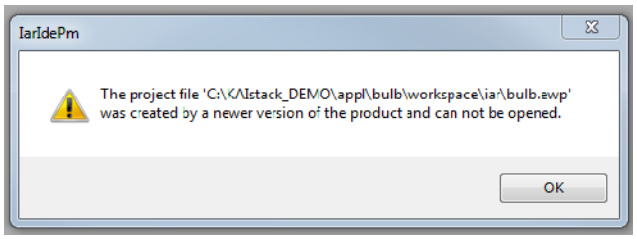


図 57. バージョン管理による IAR オープン・ワークスペース・エラー

1. エラー・メッセージで [OK] をクリックします。
2. [*プロジェクト*] をクリックします。
3. 図 58 のように、[*プロジェクトを新規作成*] をクリックします。

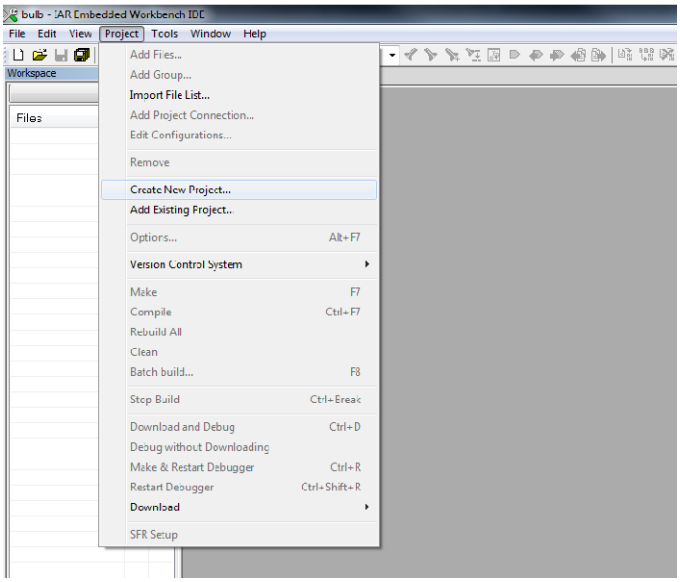


図 58. IAR でのプロジェクトの新規作成

注: 図 59 のように、[プロジェクトの新規作成] ウィンドウが開きます。

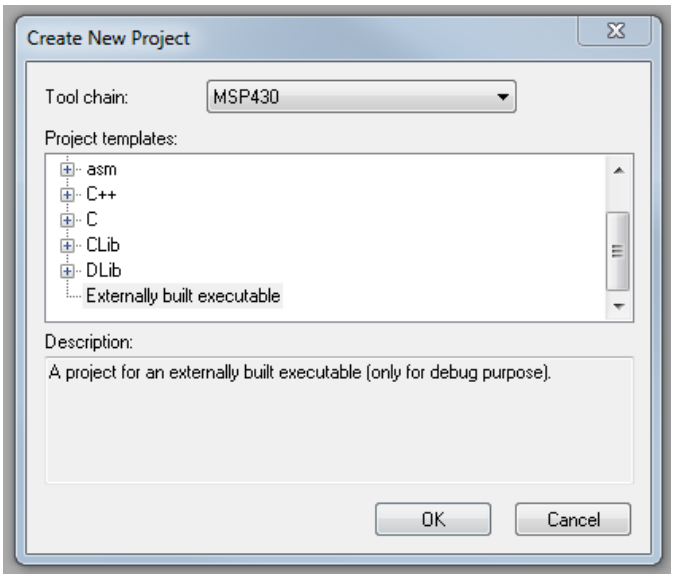


図 59. [プロジェクトを新規作成] ウィンドウ

4. ツール・チェーンとして MSP430 を選択します。
5. プロジェクト・テンプレートに対する、[*実行可能外部ビルド*] までスクロールします。
6. [OK] をクリックして保存します。
7. プロジェクトをコンピュータの任意の場所に保存します。

プロジェクトは、プログラミング対象の特定のデバイスに対して設定する必要があります。プロジェクトを特定のデバイスに対して設定するには、以下のようにします。

1. IAR のプロジェクト・ファイル（青色のキューブ付き）を右クリックします。
2. [オプション...] を選択します。

注: このアクションにより、図 60 のオプション・ウィンドウが開きます。

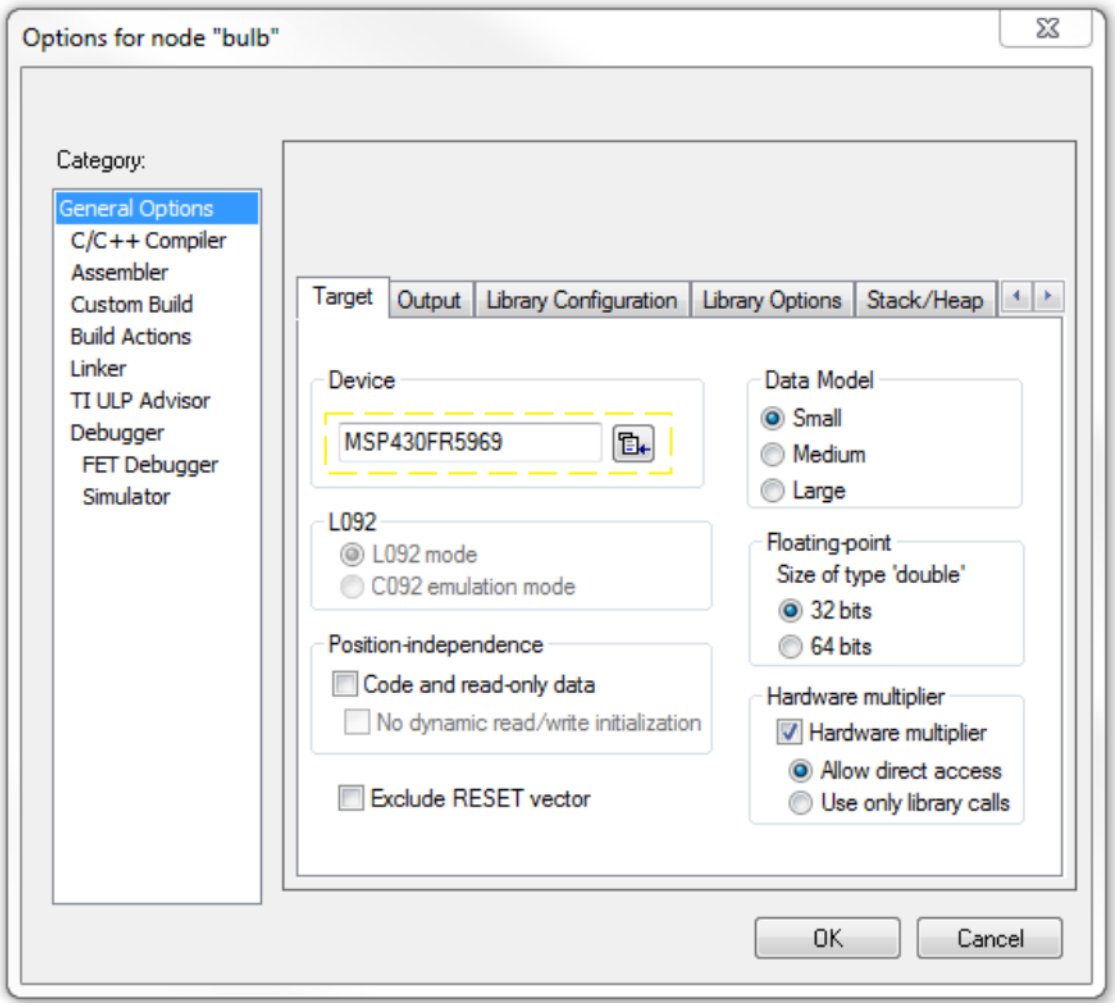


図 60. プロジェクトの設定の選択

3. [全般オプション] に移動します。
4. 適切なデバイスを選択します。

デフォルトでは、IAR でのデバッガはシミュレータに設定されます。この設定を変更すると、コードは MSP メモリにダウンロードされません。この設定を変更するには、以下のようにします。

1. [デバッガ] タブをクリックします。
2. ドライバとして [*FET* デバッガ] を選択します。
3. [OK] をクリックします。

図 61 のエラーが、[ダウンロードおよびデバッグ] ボタンを押した後に発生する場合、プロジェクト用に選択したデバイスは、ハードウェアと同じではありません。

図 61. ハードウェアと、IAR で選択したデバイスとの不整合によるエラー

プロジェクトの準備が完了しています。

1. デバッグ・ファイルを図 55 のように追加します。

注: TI は、コードをダウンロードする前に、IAR ですべてのソース・ファイルを開くことを推奨します。ソース・ファイルをワークスペースに追加しないでください。[*ファイルを追加*] をクリックして IAR でソース・コードを開くことはしないでください。ファイルを開くには、ファイルを Windows エクスプローラから図 62 の IAR の濃い灰色のウィンドウにドラッグします。IAR でファイルを開き、コードをデバイスにダウンロードしたら、ブレークポイント、監視変数、メモリ、レジスタを配置して、IAR デバッグ・ツールを最大限に活用できます。

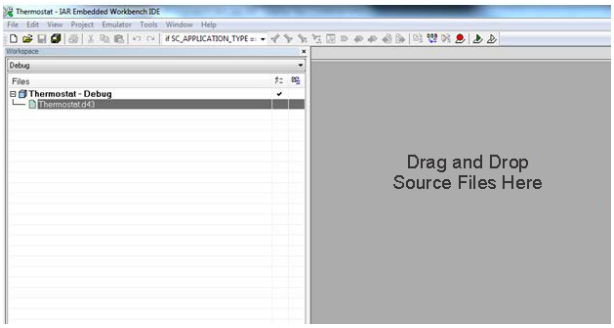


図 62. ソース・ファイルをデバッグ用に追加

2. 図 63 の [*ダウンロードおよびデバッグ*] ボタンを押すと、コードがダウンロードされます。

図 63. コードをデバイスにダウンロードし、デバッグ・セッションを開始するボタン

IAR 環境でソース・ファイルを変更できますが、プロジェクトを Windows コマンド・ラインで再構築し、IAR での構築オプションのクリックを控える必要があります。

4.6 スタック・デバイス・リソース

KAIstack が機能するためには、ハードウェアおよびソフトウェアのリソースが必要です。設計者は、スタック・リソースがアプリケーション・ファームウェアによって変更されていないことを確認する必要があります。KAIstack によって特定の MSP 用に使用されるハードウェア・リソースの完全なリストについては、KAIstack マニュアルで MSP430_TF（TP-UART の場合）または MSP430_DF（ビット・ベース PHY の場合）を検索してください。

TP-UART（KAIlink UART）構成には、UART チャンネルおよび 2 つの IO ラインが必要です。ビット・ベースの PHY バージョン（KAIlink ビット）では、タイマ、IO ライン、NMI ピンが使用されます。ビット・ベース PHY には、外部 16 MHz クリスタルが必要です。通信スタックのフットプリントにより、MSP で利用できるメモリが減少します。実際のメモリ要件は、スタックの選択された構成によって異なります。

スタックでは、約 20 KB の不揮発性メモリおよび 1.5 KB の揮発性メモリが使用されます。スタック構成により、コア周波数が制御されます。各 MSP の周波数オプションについては、KAIstack マニュアルを参照してください。

4.7 KNX プロジェクトの ETS によるテスト

IAR を使用するデバイスにファームウェアをダウンロードすると、ETS を使用して通信が動作していることを確認できます。MSP ボードおよび PHY ハードウェアに加えて、KNX 電源と USB-KNX インターフェースがテストに必要です。通信が動作していることを確認するために ETS を使用するには、以下のようにします。

1. USB-KNX インターフェースを電源、アプリケーション・ハードウェア、PC に接続します。
2. 図 64 のように、ETS を開いて、プロジェクトを新規作成します。

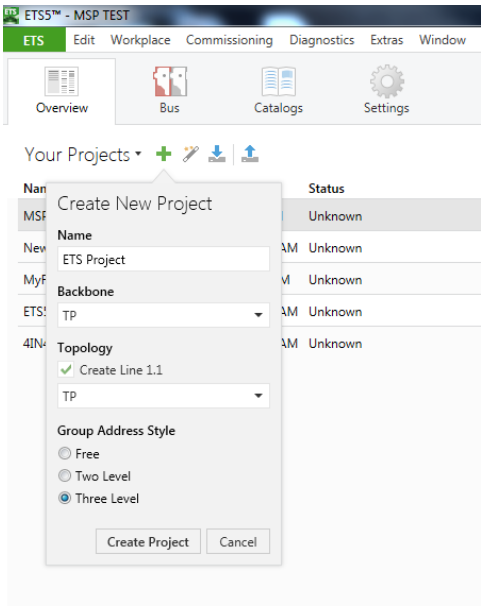


図 64. ETS プロジェクトの新規作成

3. メイン ETS ウィンドウを開きます。
4. Tapko USB インターフェースがウィンドウの左下隅で、図 65 のように選択されていることを確認します。

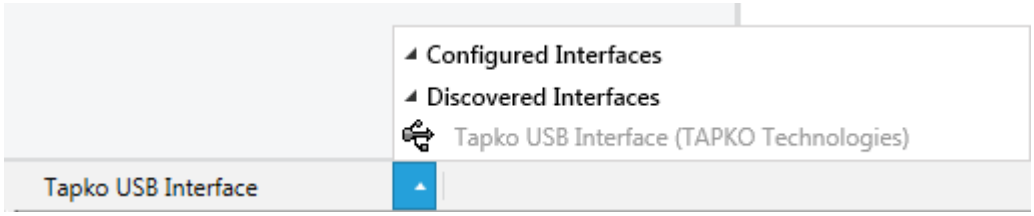


図 65. Tapko USB インターフェースが ETS で選択されていることを確認

5. [*診断*] をクリックします。
6. [*バス・モニタリング*] を選択します。

注: このステップにより診断ウィンドウが開き、ここですべての受信 KNX メッセージを表示できます。メッセージは、このウィンドウから特定の通信オブジェクトに送信できます。

プログラミング・モードのデバイスをチェックすることにより、スタックが適切に実行されていることが確認されます。KNX で、プログラミング・モードを使用して、デバイスの固有アドレスを ETS から変更できます。プログラミング・モードを有効にして、バス上のテスト用デバイスが検出されることを確認します。

1. [診断] ウィンドウの [*固有アドレス*] で、[*プログラミング・モード*] を選択します。
2. [*開始*] をクリックします。
3. デバイスをプログラミング・モードにするプログラミング・モード・ボタンまたはスイッチについて、使用する特定のボードのハードウェア・マニュアルを確認します。

注: ハードウェアがプログラミング・モードにあり、すべてが適切に実行されている場合、cotab.h で宣言されたデバイス・アドレス（KNX_DEF_INDIVIDUAL_ADDR）が [診断] ウィンドウに表示されます（図 66 を参照）。

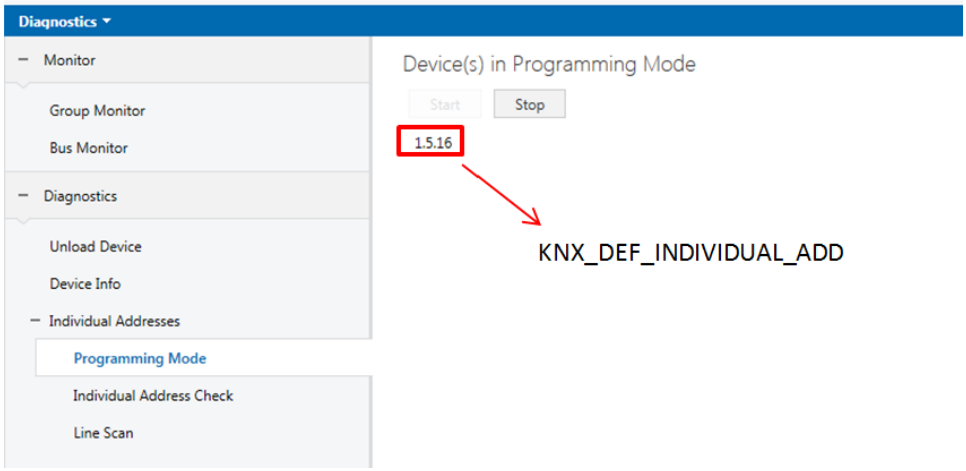


図 66. プログラミング・モードになっているデバイスの固有アドレス

注: デバイスの固有アドレスが [診断] ウィンドウに表示される場合、スタックは実行中であり、通信システムは適切に設定されています。

4. [*停止*] をクリックします。
5. 次のテストを継続します。

注: プログラミング・モードでアドレスが表示されない場合、ハードウェアの接続を確認します。ソフトウェアのデバッグが必要なことがあります。

メッセージが PC でデバイスから受信できるか確認します。電球アプリケーションには、図 67 に示す通信オブジェクトがあります。グループ・アドレス 2/0/0 の入力オブジェクト 1 つと、グループ・アドレス 2/0/1 の出力オブジェクト 1 つが宣言されます。

```
START_TAB()
DEFINE_COMM_OBJ(CO_statusIn,  ETS_GROUP3(2,0,0), RAM_PTR(statusIn), CO_TypeUint1, DPT_1, CO_RWU, CO_PRIO_L)
DEFINE_COMM_OBJ(CO_statusOut, ETS_GROUP3(2,0,1), RAM_PTR(statusOut), CO_TypeUint1, DPST_1_1, CO_RT, CO_PRIO_L)
END_TAB()
```

図 67. ETS でテストするための通信オブジェクト

1. [診断] ウィンドウの [*モニタ*] で、[*グループ・モニタ*] を選択し、デバイスによってバスを介して送信された KNX メッセージを表示します。
2. [*開始*] をクリックします。
3. ハードウェアのボタン（この例ではピン 1.0）を押して、メッセージをバスで送信します。

注: 図 68 に、時刻、優先度、固有ソース・アドレス、出力オブジェクト・アドレス、データ・ポイント・タイプ、表示される情報の値などの、情報一式を示しています。この場合、グループ・アドレス 2/0/1（図 67 の CO_statusOut）の出力オブジェクトが、1 ビットのメッセージを値 0 で送信しました。

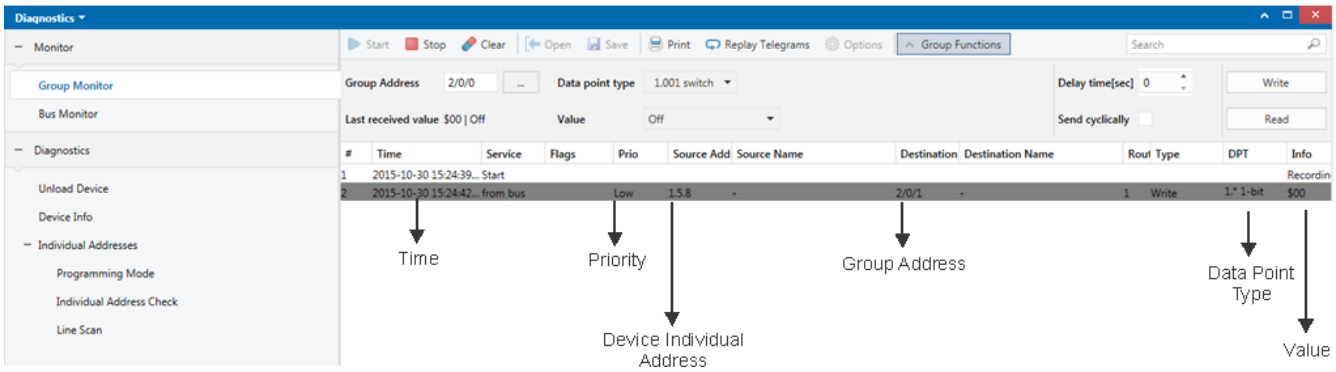


図 68. ETS によって出力通信オブジェクトから表示されるメッセージ

メッセージを、ETS から図 67 で宣言されている入力通信オブジェクトに送信できます。メッセージを設定する必要があります。メッセージを設定するには、以下のようにします。

1. 入力オブジェクトのグループ・アドレス(この場合、入力にはグループ・アドレス 2/0/0 があります)を選択します。
2. データ・ポイント・タイプを選択します(この例の入力はデータ・ポイント・タイプ CO_TypeUint1 であり、これは 1 ビット値であるため、1.001 スイッチ・オプションが選択されます(このステップでは任意の 1 ビット・オプションが機能します))。
3. [*値*] フィールドで 0(オフ)または 1(オン)を選択します。
4. [*書き込み*] をクリックします。

注: ETS により、1 ビットのメッセージがこの値と共にグループ・アドレス 2/0/0 に送信されます。ピン 1.1 が、ETS によって書き込まれた値に基づいて変更されます。図 69 に、その設定を示します。

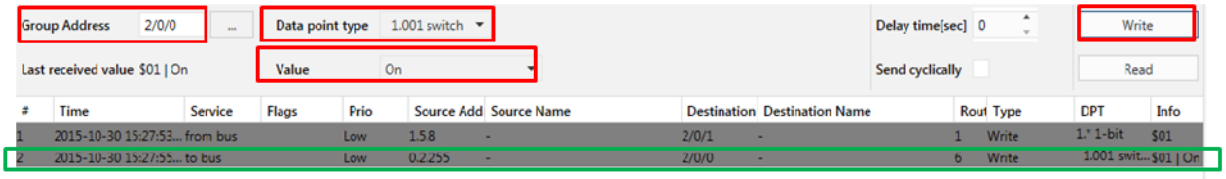
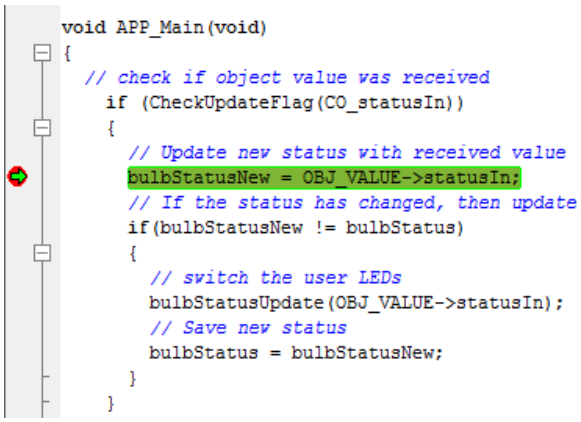


図 69. ETS からグループ・アドレス 2/0/0 の入力オブジェクトへのメッセージの送信

MSP アプリケーションでのメッセージ受信を確認するには、APP_Main() で statusIn 用フラグが図 70 のようにチェックされているかを確認します。メッセージの受信をデバッグするため、if 文の中にブレークポイントを入れます。



```
void APP_Main(void)
{
  // check if object value was received
    if (CheckUpdateFlag(CO_statusIn))
    {
      // Update new status with received value
      bulbStatusNew = OBJ_VALUE->statusIn;
      // If the status has changed, then update
      if(bulbStatusNew != bulbStatus)
      {
        // switch the user LEDs
        bulbStatusUpdate(OBJ_VALUE->statusIn);
        // Save new status
        bulbStatus = bulbStatusNew;
      }
    }
```

図 70. 入力オブジェクト値の変更を確認する

5 リファレンス

1. *KNX 協会ウェブサイト*(www.KNX.org)
2. *KNX 基礎コース・ドキュメント*、2013 年 12 月
3. *KNX 上級コース・ドキュメント*、2013 年 11 月
4. *Tapko Technologies GmbH ウェブサイト*、(www.tapko.de)
5. *Serial Data Transmission and KNX Protocol*(英語)、KNX 協会、http://www.knx.org/fileadmin/template/documents/downloads_support_menu/KNX_tutor_seminar_page/tutor_documentation/05_Serial%20Data%20Transmission_E0808f.pdf
6. *Interworking*(英語)、KNX 協会、http://www.knx.org/fileadmin/template/documents/downloads_support_menu/KNX_tutor_seminar_page/Advanced_documentation/05_Interworking_E1209.pdf
7. *ETS5 for Beginners*(英語)、KNX 協会、http://www.knx.org/media/docs/Flyers/ETS5-For-Beginners/ETS5-For-Beginners_en.pdf
8. *How to become an ETS App Developer*(英語)、KNX 協会、http://www.knx.org/media/docs/Flyers/How-to-Become-An-ETS-App-Developer/How-To-Become-An-ETS-App-Developer_en.pdf
9. *KNX Development Getting Started – KNX System Components*(英語)、KNX 協会、http://www.knx.org/media/docs/Flyers/KNX-Development-Getting-Started/KNX-Development-Getting-Started_en.pdf
10. *Texas Instruments ISO 9001 Certification*(英語)、http://focus.ti.com/pdfs/qlty/TI_286691_ISO9001_Final_Certificate.pdf
11. *The worldwide STANDARD for home and building control*(英語)、KNX 協会、http://www.knx.org/media/docs/Flyers/KNX-Introduction-Flyer/KNX-Introduction-Flyer_en.pdf

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265