Introduction à Modbus

Date de publication: août 04, 2014

Introduction

Le protocole industriel Modbus a été développé en 1979 afin de rendre possibles les communications entre des matériels d'automatisation. Implémenté à l'origine comme un protocole du niveau application destiné au transfert de données sur une couche série, ce protocole s'est élargi pour comprendre des implémentations sur port série, TCP/IP et UDP (User Datagram Protocol). Il s'agit aujourd'hui d'un protocole communément répandu, utilisé par de nombreux matériels pour des communications simples, fiables et efficaces sur une multitude de réseaux modernes.

Table des matières

Introduction à Modbus

Le protocole Modbus est généralement utilisé pour des communications réseau de type SCADA (Supervisory Control and Data Acquisition) entre matériels. Par exemple, un serveur peut être utilisé pour commander un PLC (Programmable Logic Controller) ou un automate programmable (PAC), qui peuvent à leur tour commander un capteur, une vanne, un moteur ou tout autre matériel embarqué.

Pour répondre à ces besoins, le protocole Modbus a été conçu comme un protocole de requête/réponse avec un modèle de données et de fonction souple, ce qui explique en partie qu'il soit encore utilisé aujourd'hui.

1. Le cycle requête/réponse

Le protocole Modbus suit une architecture maître/esclave dans laquelle un maître transmet une requête à un esclave et attend sa réponse. Cette architecture donne au maître un contrôle total sur le flux d'information, ce qui présente des avantages par rapport aux réseaux série multipoint plus anciens. Même sur les réseaux TCP/IP modernes, le maître dispose d'un degré élevé de contrôle sur le comportement de l'esclave, ce qui s'avère utile dans certaines conceptions.

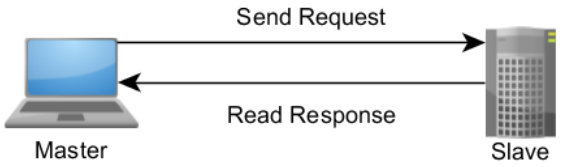

Figure 1. Relation maître/esclave, requête/réponse entre des périphériques Modbus

Dans Modbus, cette requête est constituée d'un ensemble de données organisées par couche. La première couche est l'unité de données d'application (ADU, Application Data Unit), généralement considérée comme étant le "type" de protocole Modbus utilisé. Il existe 3 ADU : ASCII, unité terminale distante (RTU, Remote Terminal Unit) et TCP/IP.

TCP est un format moderne qui permet une gestion efficace des requêtes et des réponses Modbus dans les logiciels, ainsi qu'une mise en réseau plus efficace à l'aide de connexions spécialisées et d'identifiants pour chaque requête. RTU et ASCII sont des formats d'ADU série plus anciens dont la principale différence réside dans le fait que le type RTU utilise une représentation binaire compacte alors que le type ASCII envoie toutes les requêtes sous forme de suites de caractères ASCII.

Pour la plupart des applications, l'ADU choisie dépend du réseau physique voulu (Ethernet, série ou autre), du nombre de périphériques sur le réseau et des ADU supportées par les matériels maîtres et esclaves présents sur le réseau. Du point de vue de l'application utilisant le protocole Modbus, les données devraient simplement être exposées comme si l'ADU n'existait pas.

Chaque ADU comprend une unité de données de protocole (PDU, Protocol Data Unit), qui représente le noyau du protocole Modbus. Chaque PDU contient un code de fonction et les données qui lui sont associées. Chaque code de fonction a une réponse bien définie et ce code peut être considéré comme la commande envoyée à l'esclave.

Des erreurs peuvent parfois se produire. Modbus définit un PDU spécifique pour les exceptions, permettant au maître de savoir ce qu'il s'est passé. La plupart des drivers le convertissent en une forme que l'application ou le langage utilisés peuvent comprendre.

2. Le modèle de données Modbus

Modbus gère l'accès aux données de manière simple et souple. Modbus supporte d'origine deux types de données : une valeur booléenne et un entier non signé 16 bits.

Dans les systèmes SCADA, il est courant que les matériels embarqués aient certaines valeurs définies comme entrées, comme les gains ou les paramètres PID (proportionnelle, intégrale, dérivée), tandis que d'autres sont définies comme sorties, comme la température actuelle ou la position de la vanne. Pour répondre à ce besoin, les valeurs des données Modbus sont divisées en quatre gammes (voir tableau 1). Un esclave peut définir jusqu'à 65 536 éléments dans chaque gamme.

Bloc mémoire	Type de données	Accès du maître	Accès de l'esclave
Seuils	Booléen	Lecture/Écriture	Lecture/Écriture
Entrées discrètes	Booléen	Lecture seule	Lecture/Écriture
Registres internes	Mot non signé	Lecture/Écriture	Lecture/Écriture
Registres d'entrée	Mot non signé	Lecture seule	Lecture/Écriture

Tableau 1. Blocs du modèle de données Modbus

Dans beaucoup de cas, les capteurs et d'autres matériels génèrent des données d'un type autre que les types booléen et entier non signé. De manière générale, les matériels esclaves convertissent ces larges types de données en registres. Par exemple, un capteur de pression peut répartir une valeur à virgule flottante 32 bits entre deux registres de 16 bits.

Modbus expose ces valeurs de façon complètement conceptuelle, ce qui signifie qu'elles peuvent ne pas réellement exister en mémoire. Par exemple, un périphérique esclave peut être défini de sorte que les registres internes et les registres d'entrée partagent la même mémoire si ce comportement est logique pour l'esclave. Dans la plupart des cas, un esclave stocke chaque type de données qu'il supporte dans une mémoire séparée et limite le nombre d'éléments de données auxquels le maître peut accéder. Cette souplesse est une option en raison de la façon dont les données sont exposées via le comportement bien défini des codes de fonction Modbus.

3. Codes de fonction Modbus

Les codes de fonction Modbus déterminent la façon dont le maître peut accéder aux données et les modifier. Contrairement aux gammes de données, qui sont conceptuelles, les codes de fonction ont un comportement bien défini. Lorsqu'un esclave doit exécuter un code de fonction, il utilise les paramètres de la fonction pour exécuter ce comportement bien défini. La figure 2 illustre ce lien entre une requête de fonction et la mémoire réelle du périphérique.

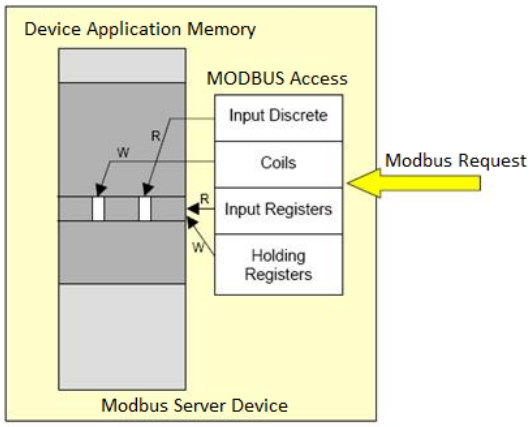

Figure 2. Mappage entre un code de fonction, les gammes de données et la mémoire réelle d'un périphérique esclave.

Les codes de fonction les plus courants sont nommés d'après la gamme de données conceptuelle qu'ils modifient ou à laquelle ils accèdent. Par exemple, "read holding registers" (lire les registres internes) extrait les données de la mémoire définie comme registres internes et les renvoie au maître. Le tableau 2 dresse la liste des codes de fonction les plus courants.

				Code	Sub Code	(hex)	Section
Data Access	Bit Access	Physical Discrete Inputs	Read Discrete Inputs	02		02	6.2
		Internal Bits or Physical Coils	Read Coils	01		01	6.1
			Write Single Coil	05		05	6.5
			Write Multiple Coils	15		0F	6.11
	16 bits access	Physical Input Registers	Read Input Register	04		04	6.4
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03	6.3
			Write Single Register	06		06	6.6
			Write Multiple Registers	16		10	6.12
			Read/Write Multiple Registers	23		17	6.17
			Write Mask Register	22		16	6.16
			Read FIFO Queue	24		18	6.18
	File Record Access		Read File Record	20		14	6.14
			Write File Record	21		15	6.15
			Read Exception Status	07		07	6.7
			Diagnostic	08		08	6.8

Tableau 2. Codes de fonction courants

Initiation à Modbus en LabVIEW

NI propose trois principaux mécanismes afin de communiquer avec des périphériques Modbus : (1) un serveur OPC de haut niveau, (2) un serveur d'E/S Modbus et (3) une API Modbus de bas niveau introduite dans le logiciel NI LabVIEW 2014 par le biais des modules LabVIEW Real-Time et LabVIEW Datalogging and Supervisory Control (DSC).

4. API LabVIEW Modbus

L'API Modbus de bas niveau est l'option à privilégier lorsque votre application requiert un niveau élevé de contrôle du séquencement et du cadencement des requêtes Modbus. L'API de bas niveau représente également la meilleure option lorsque la souplesse est un élément crucial. En revanche, la souplesse et la puissance offertes par l'API LabVIEW Modbus implique que le code de votre application soit plus complexe afin que l'API soit correctement gérée. Pour vous aider à mieux appréhender cette complexité, LabVIEW vous propose deux exemples.

Exemple d'introduction à Modbus

Le premier exemple de projet, **Modbus Library.lvproj**, présente brièvement les fonctionnalités de l'API. Il démontre également les différences entre une implémentation sur PC et sur une cible temps réel. La figure 3 présente le code utilisé dans l'exemple du maître Modbus sur cible temps réel.

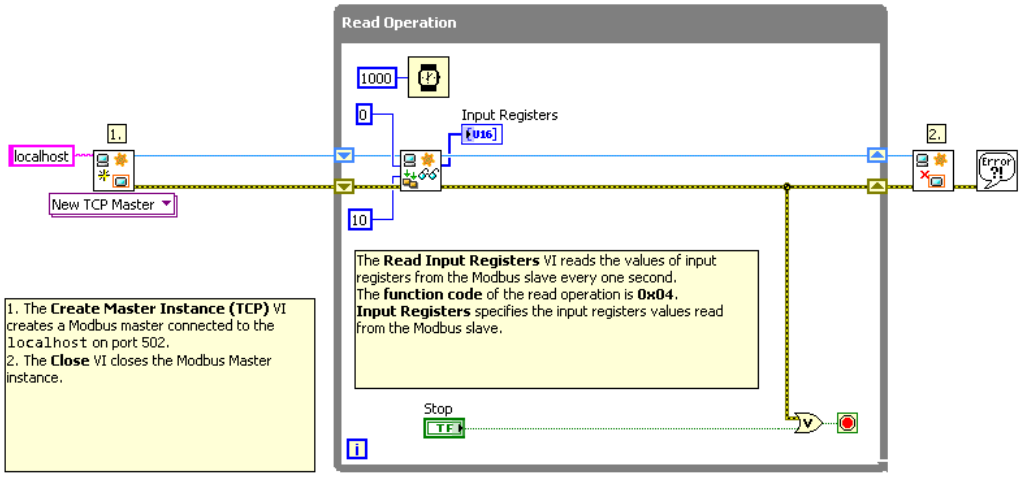

Figure 3. VI Master on RT Target

Cet exemple présente les exigences de base d'une application Modbus utilisant l'API LabVIEW. Tout d'abord, une instance Modbus est créée. Dans le cas présent, il s'agit d'un maître TCP. Cependant, vous pouvez modifier cet exemple et utiliser un maître série à l'aide du sélecteur d'instance polymorphe.

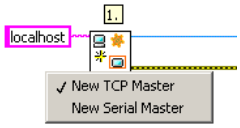

Figure 4. Modification du type de maître Modbus

Une fois l'instance créée, vous pouvez commencer à interroger le matériel esclave pour obtenir des données. Cet exemple illustre l'utilisation du code de fonction **Read Input Registers** (Lire les registres d'entrée). Tous les codes de fonction Modbus supportés par l'API figurent sur la palette appropriée. En raison de l'implémentation du protocole, l'API esclave dispose de fonctions supplémentaires, que le maître ne peut pas implémenter. Par exemple, un esclave peut écrire dans la gamme des registres d'entrée, tandis qu'un maître ne peut que lire cette gamme. La figure 5 représente les codes de fonction.

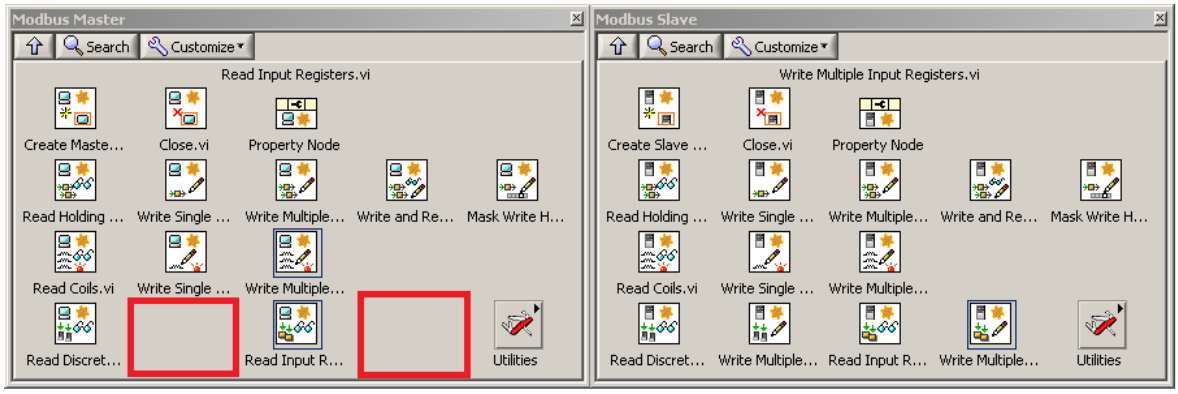

Figure 5. Palettes Modbus Master et Modbus Slave avec les codes de fonction

Enfin, l'instance Modbus est fermée, libérant ainsi la mémoire qui lui est associée. Toutes les références sont également fermées, y compris les références de connexion TCP ou série NI-VISA utilisées par cette instance.

Seul l'exemple du maître a été étudié jusqu'à présent. Cependant, chaque exemple suit le même modèle de base, connu de la plupart des utilisateurs LabVIEW : ouvrir, lire/écrire et fermer.

Pour finir, bien que l'API semble identique, il est important d'en comprendre une différence clé. Si votre matériel est un maître, il doit envoyer une requête via le réseau à l'esclave approprié pour acquérir des données. En revanche, l'esclave a son propre stockage de données local et peut y accéder rapidement.

Exemple du maître redondant

L'exemple de base peut être suffisant pour certaines applications ; cependant cela risque de ne pas être le cas pour des applications complexes dont le but est de communique avec un capteur ou une passerelle. Afin de combler ce manque, un exemple d'application illustre comment utiliser deux maîtres pour communiquer avec un esclave donné. Si l'un des maîtres échoue et perd la connexion avec l'esclave ou avec l'interface homme-machine (IHM), l'autre maître prend le relais.

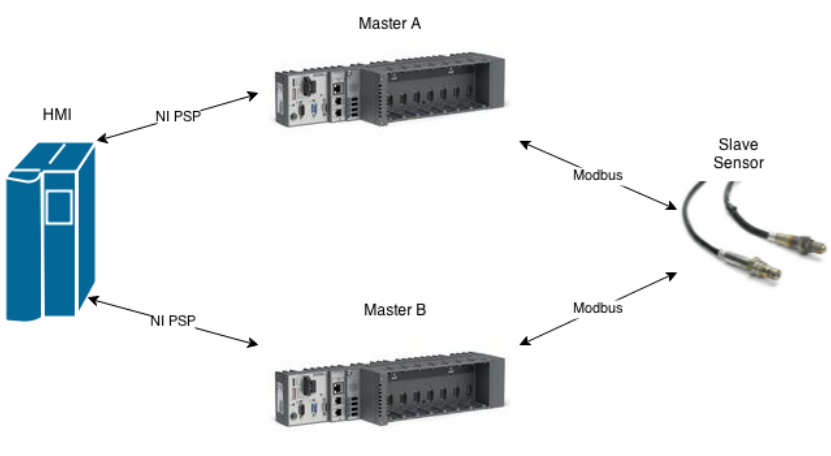

Figure 6. Conception de l'exemple du maître redondant

Si cette conception répond aux besoins de votre application, ou si vous êtes intéressé par un exemple plus complexe de communication Modbus, reportez-vous à l'exemple de projet **Redundant Modbus Masters.lvproj** dans l'Outil de recherche d'exemple.

5. Serveurs d'E/S Modbus

Les serveurs d'E/S Modbus, qui sont disponibles dans les modules LabVIEW DSC et LabVIEW Real-Time, constituent un moteur de haut niveau pour communiquer via Modbus. Plutôt que de spécifier un code de fonction que vous souhaitez envoyer, vous enregistrez l'ensemble de données auquel vous aimeriez accéder et le serveur d'E/S se charge de planifier les requêtes automatiquement à l'intervalle spécifié.

Pour utiliser des serveurs d'E/S, vous devez ajouter un nouveau serveur d'E/S à la cible souhaitée dans votre projet. Comme dans le cas de l'API de bas niveau, vous pouvez choisir entre un maître ou un esclave Modbus, qui permettent d'accéder à des paramètres supplémentaires. Par exemple, un maître a un intervalle d'interrogation défini, correspondant à l'intervalle auquel chaque requête est envoyée à l'esclave, tandis que les esclaves doivent attendre de recevoir ces requêtes et n'ont pas de cadencement prédéfini.

Après avoir créé le serveur d'E/S, vous spécifiez les éléments du périphérique que vous souhaitez lire. Contrairement à l'API de bas niveau, dans laquelle vous devez générer et traiter les requêtes vous-même, les serveurs d'E/S Modbus vous donnent le choix entre une grande variété de formats et de types de données. Par exemple, vous pouvez lire le registre interne à l'adresse 0 en associant une variable à l'élément 400001, lire le premier octet de ce registre en sélectionnant 400001.1 et lire le nombre flottant simple précision stocké dans les registres 0 et 1 en sélectionnant F400001.

Après avoir sélectionné les variables auxquelles vous voulez accéder, vous pouvez les lire ou les écrire en utilisant des nœuds variable partagée sur le diagramme. Vous avez même la possibilité d'attribuer un alias aux variables.

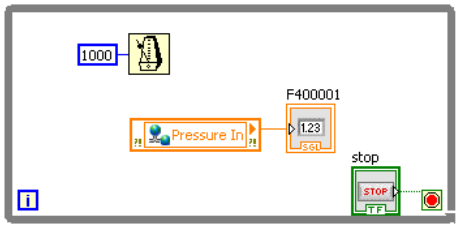

Figure 7. Application simple de serveur d'E/S

La difficulté de programmation d'une application de serveur d'E/S est minimale et le développement plus facile à comprendre. Soyez cependant conscient que cette facilité d'utilisation est accompagnée de quelques limitations. Les données ne se mettent à jour qu'à l'intervalle spécifié et il n'est pas possible d'ajouter ou de supprimer des données demandées au moment de l'exécution. Si ces limitations sont acceptables dans le cadre de votre application, les serveurs d'E/S représentent donc la meilleure option multi-plateforme.

Pour obtenir plus d'informations et un guide détaillé, consultez la page Connect LabVIEW to Any PLC With Modbus.

6. Serveurs NI OPC avec des serveurs d'E/S OPC ou OPC UA

Pour des applications complexes comprenant plusieurs périphériques esclaves qui communiquent par l'intermédiaire de protocoles différents, le serveur d'E/S Modbus classique risque de ne pas suffire. Une solution courante consiste à utiliser un serveur OPC, qui joue le rôle d'agrégateur de données pour tous les systèmes, puis à utiliser les serveurs d'E/S OPC inclus dans le module LabVIEW DSC pour communiquer avec ce serveur OPC.

La figure 8 représente un exemple d'une telle architecture, incluant des serveurs NI OPC qui utilisent Modbus pour communiquer directement avec les capteurs et OPC UA pour communiquer avec un contrôleur d'automatismes programmable (PAC) NI CompactRIO. Une fois les données agrégées dans les serveurs NI OPC, un serveur d'E/S OPC peut récupérer ces données et les partager avec l'application LabVIEW.

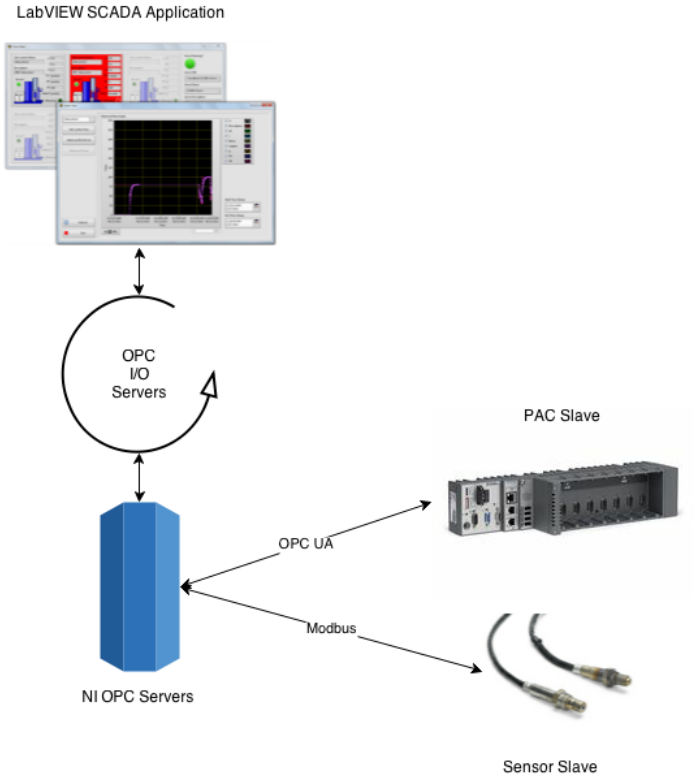

Figure 8. Application SCADA utilisant Modbus, des serveurs NI OPC et des serveurs d'E/S OPC

Vous pouvez aussi développer une architecture similaire qui utilise le driver OPC UA inclus dans le module LabVIEW DSC à la place des serveurs d'E/S OPC. Cependant, le driver OPC UA est un driver de bas niveau et ne procure pas la même facilité d'utilisation que les serveurs d'E/S OPC.

Pour développer une application de ce type, vous devez tout d'abord générer une configuration valide pour que les serveurs NI OPC puissent communiquer avec vos périphériques esclaves. Pour cela, il faut générer des voies, qui définissent la configuration d'un driver, et des périphériques, qui définissent une extrémité individuelle pour ce driver. Après avoir configuré un périphérique, vous pouvez générer des balises.

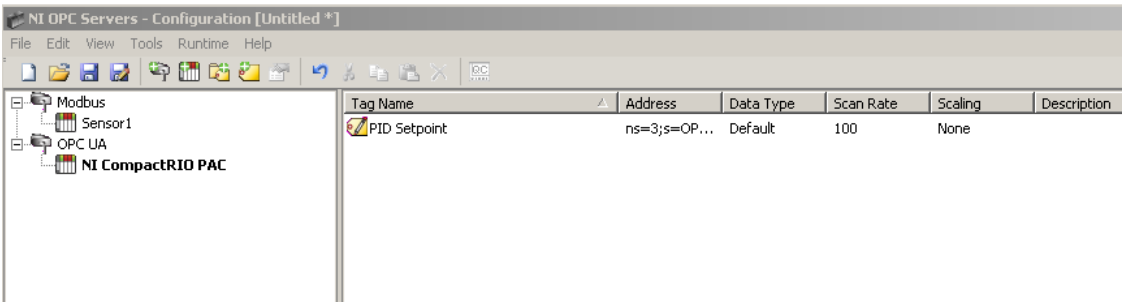

Figure 9. Exemple de configuration au niveau des serveurs NI OPC pour l'architecture représentée ci-dessus.

Une fois que vous avez configuré les serveurs NI OPC, vous pouvez configurer un serveur d'E/S OPC pour communiquer avec ces balises. Alors que les serveurs d'E/S Modbus sont configurés pour accéder aux registres, les serveurs d'E/S OPC sont configurés pour accéder aux balises du serveur OPC.

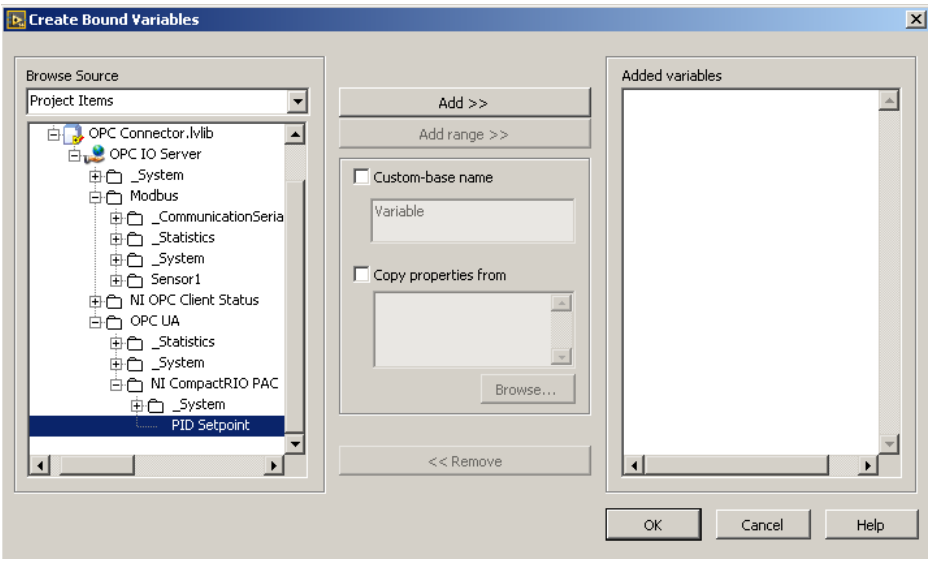

Figure 10. Configuration des serveurs d'E/S OPC

Ce processus de liaison entraîne la génération de variables que vous pouvez ensuite utiliser dans votre application.

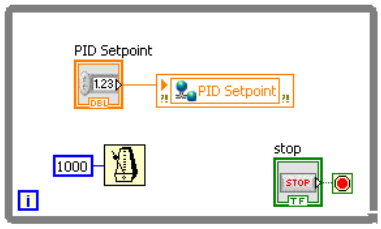

Figure 11. Application simple utilisant des serveurs d'E/S OPC

Vous trouverez une présentation détaillée de ce processus sur la page Connect LabVIEW to Any PLC Using OPC.

Répondre aux besoins de votre application

Modbus est un protocole simple, que vous pouvez utiliser de différentes manières pour implémenter des applications puissantes.

En ce qui concerne la communication Modbus, NI propose trois options principales qui offrent une large gamme de fonctionnalités permettant de répondre aux besoins de votre application. Tout d'abord, une API de bas niveau garantit un contrôle précis du protocole, ainsi que de hautes performances, mais au détriment de la facilité d'utilisation. L'utilisation de cette API de bas niveau implique que tout doit être fait manuellement. Pour des applications de suivi plus simples, les serveurs d'E/S Modbus proposent une API plus simple et plus facile à utiliser pour accéder à des données Modbus ou les distribuer. Bien que faciles à utiliser, les serveurs d'E/S ne permettent pas d'avoir un contrôle sur le protocole aussi précis que ne peuvent l'exiger certaines applications. Enfin, pour des systèmes complexes et de grande ampleur, il peut s'avérer avantageux d'opter pour un serveur OPC complet pouvant servir d'agrégateur de données. Puis utilisez simplement un outil comme le driver LabVIEW OPC UA ou des serveurs d'E/S OPC pour permettre à votre application d'accéder aux données.

Étude approfondie du protocole Modbus

Date de publication: août 05, 2014

Introduction

Modbus est un protocole industriel qui a été développé en 1979 afin de permettre les communications entre des appareils d'automatisation. Initialement mis en place comme protocole d'application ayant pour but de transférer des données via une couche série, Modbus s'est étendu et inclus désormais des implémentations série, TCP/IP ainsi que le protocole UDP (User Datagram Protocol). Ce document détaille la mise en œuvre du protocole.

Table des matières

1. Qu'est-ce que le protocole Modbus ?

Modbus est un protocole de requête-réponse mis en œuvre en utilisant une relation maître-esclave. Dans une relation maître-esclave, la communication a toujours lieu par paires (un appareil doit initier une requête puis attendre une réponse), et l'appareil qui initie, le maître, est en charge d'initier toutes les interactions. En règle générale, le maître est une IHM (interface homme-machine) ou un superviseur de contrôle et d'acquisition de données (SCADA), et l'esclave est un capteur, un automate programmable industriel (PLC) ou un contrôleur d'automatisme programmable (PAC). Le contenu de ces requêtes et réponses ainsi que les couches réseau à travers lesquelles ces messages sont envoyés sont définis par les différentes couches du protocole.

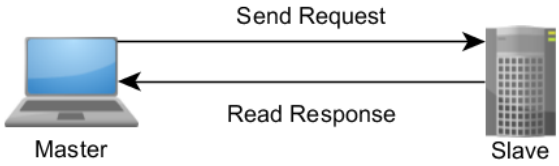

Figure 1. Relation de mise en réseau maître-esclave

Couches du protocole Modbus

Lors de la mise en œuvre initiale, Modbus était un protocole unique construit sur une couche série, donc il ne pouvait pas être divisé en plusieurs couches. Au fil du temps, diverses unités de données d'applications (ADU) ont été introduites pour soit changer le format de paquet utilisé via une couche série soit permettre l'utilisation de réseaux TCP/IP et UDP. Cela a mené à la séparation du protocole noyau, qui définit l'unité de données de protocole (PDU), et de la couche réseau, qui définit l'unité de données d'application (ADU).

2. Unité de données de protocole (PDU)

Le PDU et le code qui le définit composent le noyau des Spécifications du protocole d'application Modbus. Ce document de spécifications définit le format du PDU, les différents concepts de données utilisés par le protocole, l'utilisation de codes de fonction pour accéder à ces données et la mise en œuvre et les restrictions spécifiques de chaque code de fonction.

Le format PDU du protocole Modbus est défini comme code de fonction suivi d'un ensemble de données associé. La taille et le contenu de ces données sont définis par le code de fonction, et le PDU entier (code de fonction et données) ne peut excéder 253 octets. Chaque code de fonction a un comportement spécifique que les esclaves peuvent mettre en œuvre de manière flexible en fonction du comportement désiré de leur application. Les spécifications du PDU définissent les concepts fondamentaux pour l'accès et la manipulation de données ; cependant, un esclave peut gérer des données de façon non explicitement définie dans les spécifications.

Accès aux données dans Modbus et le modèle de données Modbus

En général, les données accessibles par Modbus sont stockées dans l'une des quatre banques de données ou gammes d'adresse : bobines, entrées discrètes, registres de maintien et registres d'entrée. Comme c'est le cas de la plupart des spécifications, les noms peuvent varier selon l'industrie ou l'application. Par exemple, les registres de maintien peuvent être appelés "registres de sortie", et les bobines peuvent être appelées "sorties numériques ou discrètes". Ces banques de données définissent le type et les droits d'accès des données contenues. Les appareils esclaves disposent d'un accès direct à ces données, qui sont hébergées localement sur ces appareils. Les données accessibles par Modbus sont généralement un sous-ensemble de la mémoire principale de l'appareil. Par contraste, les maîtres Modbus doivent demander un accès à ces données par le biais de divers codes de fonction. Le comportement de chaque bloc est décrit au Tableau 1.

Bloc de mémoire	Type de données	Accès par le maître	Accès par l'esclave
Bobines	Booléen	Lire/Écrire	Lire/Écrire
Entrées discrètes	Booléen	Lecture seule	Lire/Écrire
Registres de maintien	Mot non signé	Lire/Écrire	Lire/Écrire
Registres d'entrée	Mot non signé	Lecture seule	Lire/Écrire

Tableau n°1. Blocs du modèle de données Modbus

Ces blocs vous offrent la possibilité de restreindre ou d'autoriser l'accès à différents éléments de données, mais aussi d'apporter des mécanismes simplifiés à la couche d'application pour accéder à différents types de données.

Les blocs sont entièrement conceptuels. Ils peuvent exister comme adresses de mémoire séparées dans un système donné, mais peuvent aussi se chevaucher. Par exemple, le bloc de bobines peut exister au même emplacement en mémoire que le premier bit du mot représenté par le bloc de registre de maintien. Le schéma d'adressage est entièrement défini par l'appareil esclave et son interprétation de chaque bloc de mémoire est une part importante du modèle de données de l'appareil.

Adressage du modèle de données

Les spécifications définissent chaque bloc comme contenant un espace d'adressage de 65 536 (2^{16}) éléments maximum. Au sein de la définition du PDU, Modbus définit l'adresse de chaque élément de données allant de 0 à 65 535. Cependant, chaque élément de données est numéroté de 1 à n, n ayant une valeur maximale de 65 536. C'est-à-dire que la bobine 1 fait partie du bloc de bobines à l'adresse 0, alors que le registre de maintien 54 est à l'adresse 53 dans la section de la mémoire que l'esclave a définie comme registre de maintien.

Les gammes complètes autorisées par les spécifications n'ont pas besoin d'être mises en œuvre par un appareil donné. Par exemple, un appareil peut choisir de ne pas mettre en œuvre de bobine, d'entrée discrète ou de registre d'entrée, mais de plutôt n'utiliser que les registres de maintien 150 à 175 et 200 à 225. Cela est tout à fait acceptable, et les tentatives d'accès non valides seraient gérées par des exceptions.

Gammes d'adressage de données

Bien que les spécifications définissent différents types de données comme existant dans divers blocs et assignent une gamme d'adresse locale à chaque type, cela ne résulte pas nécessairement en un schéma d'adressage intuitif servant pour la documentation ou la compréhension de la mémoire accessible par Modbus d'un appareil donné. Pour simplifier la discussion sur les emplacements de blocs de mémoire, un schéma de numérotation a été introduit, ajoutant des préfixes à l'adresse des données en question.

Par exemple, plutôt que de référencer un élément sous le registre de maintien 14 à l'adresse 13, un manuel d'utilisateur mentionnerait un élément de données à l'adresse 4 014, 40 014 ou 400 014. Dans chaque cas, le premier nombre spécifié est 4 et représente les registres de maintien, et l'adresse est spécifiée en utilisant les nombres restants. La différence entre 4XXX, 4XXXX et 4XXXXX dépend de l'espace d'adressage utilisé par l'appareil. Si les 65 536 registres sont utilisés, la numérotation 4XXXXX devrait être utilisée dans la mesure où elle permet une gamme allant de 400 001 à 465 536. Si seuls quelques registres sont utilisés, il est courant d'utiliser la gamme de 4 001 à 4 999.

Dans ce schéma d'adressage, un préfixe est assigné à chaque type de données, comme l'indique le Tableau 2.

Bloc de données	Préfixe
Bobines	0
Entrées discrètes	1
Registres d'entrée	3
Registres de maintien	4

Tableau 2. Préfixes de gammes de données

Les bobines existent avec un préfixe de 0. Cela signifie qu'une référence de 4001 peut faire référence au registre de maintien 1 ou à la bobine 4001. De ce fait, il est recommandé que toutes les nouvelles mises en œuvre utilisent un adressage à 6 chiffres avec des zéros devant et que cela soit indiqué dans la documentation. Ainsi, le registre de maintien 1 est référencé comme 400 001 et la bobine 4001 est référencée comme 004 001.

Valeurs de départ de l'adressage de données

La différence entre des adresses de mémoire et des numéros de référence est plus compliquée à cause de l'indexation spécifique d'une application donnée. Comme nous l'avons précédemment mentionné, le registre de maintien 1 est à l'adresse 0. Généralement, les numéros de référence sont indexés à un, ce qui signifie que la valeur de départ d'une gamme donnée est 1. Ainsi, 400 001 devient littéralement le registre de maintien 00001, qui est à l'adresse 0. Certaines mises en œuvre choisissent de commencer leurs gammes à 0, ce qui veut dire que 400 000 devient le registre de maintien à l'adresse 0. Le Tableau 3 démontre ce concept.

Adresse	Numéro de registre	Numéro (indexation à 1, standard)	Numéro (indexation à 0, alternatif)
0	1	400001	400000
1	2	400002	400001
2	3	400003	400002

Tableau 3. Schémas d'indexation de registre

Les gammes d'indexation à 1 sont courantes et vivement recommandées. Dans tous les cas, la valeur de départ de chaque gamme devrait se trouver dans la documentation.

Types de données larges

Le standard Modbus fournit un modèle de données relativement simple qui n'inclut pas de type de données supplémentaire hormis un mot non signé et une valeur de bit. Bien que cela suffise pour certains systèmes (pour lesquels les valeurs de bit correspondent à des bobines et des relais et les valeurs de mot correspondent à des valeurs de C A/N non mises à l'échelle) ce modèle n'est pas suffisant pour des systèmes plus avancés. Par conséquent, de nombreuses mises en œuvre Modbus incluent des types de données qui vont au-delà des limites de registre. Le Module NI LabVIEW Datalogging and Supervisory Control (DSC) et le KEPServerEX définissent tous deux un certain nombre de types de référence. Par exemple, les chaînes stockées dans un registre de maintien suivent une forme standard (400 001) mais sont suivies d'un nombre décimal ainsi que de sa longueur et l'ordre des octets de sa chaîne (400 001,2 H, chaîne à deux caractères du registre de maintien 1 dans laquelle l'octet élevé correspond au premier caractère de la chaîne). Cela est requis car chaque requête a une taille finie, donc un maître Modbus doit connaître les limites exactes de la chaîne au lieu de rechercher une longueur ou un séparateur tel que NULL.

Accès aux bits

En plus d'autoriser l'accès à des données qui dépassent une certaine limite de registre, certains maîtres Modbus supportent les références à des bits individuels au sein d'un registre. Cela est bénéfique car ça permet aux appareils de combiner des données de tout type dans la même gamme de mémoire sans avoir à diviser les données binaires entre les gammes de bobines et celles d'entrées discrètes. Cela est généralement référencé en utilisant un séparateur décimal et l'indice ou le nombre de bit, selon la mise en œuvre. C'est-à-dire que le premier bit du premier registre peut être 400 001,00 ou 400 001,01. Il est recommandé que toute documentation spécifie le schéma d'indexation utilisé.

Endianness (ordre des octets) de données

Les données de registres multiples, telles que la valeur à virgule flottante, peuvent facilement être transférées vers Modbus en divisant ces données au sein de deux registres. Dans la mesure où cela n'est pas défini par le standard, l'endianness (ou ordre des octets) de cette séparation n'est pas défini. Bien que chaque mot non signé doive être envoyé dans l'ordre des octets (Big Endian) du réseau pour répondre au standard, de nombreux appareils inversent l'ordre des octets des données. La Figure 2 en montre un exemple non courant mais valide.

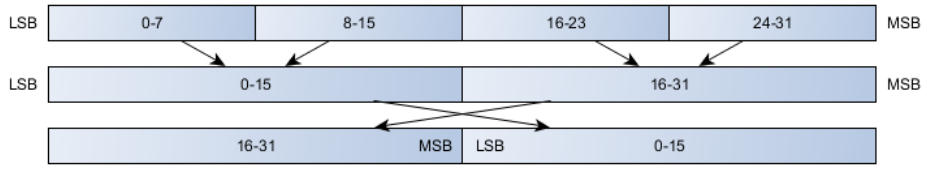

Figure 2. Inversion de l'ordre des octets de données multi-octets

Il incombe au maître de comprendre comment l'esclave stocke les informations en mémoire et de les décoder correctement. Il est recommandé que la documentation reflète l'ordre des mots utilisé par le système. L'ordre des octets peut aussi être ajouté comme option de configuration de système, avec des fonctions d'encodage et de décodage sous-jacentes, si une flexibilité de mise en œuvre est requise.

Chaînes

Les chaînes peuvent facilement être stockées dans des registres Modbus. Pour davantage de simplicité, certaines mises en œuvre requièrent que la longueur de chaînes soit un multiple de deux, en ajoutant des valeurs NULL là où il y a des espaces supplémentaires. L'ordre des octets est aussi une variable dans les interactions de chaînes. Le format de chaîne peut ou non inclure un NULL comme valeur finale. Pour illustrer cette variabilité, certains appareils stockent les données, comme l'indique la Figure 3.

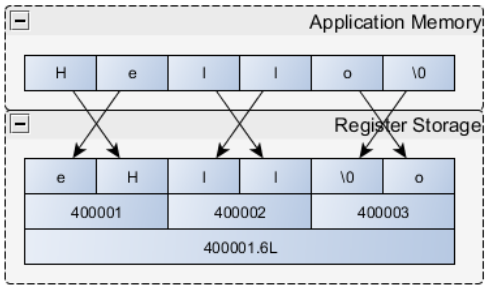

Figure 3. Inversion de l'ordre des octets dans les chaînes Modbus

Comprendre les codes de fonction

Contrairement au modèle de données qui peut varier de manière significative d'un appareil à l'autre, les codes de fonction et leurs données sont définis explicitement par le standard. Chaque fonction suit un modèle. D'abord, l'esclave valide les entrées telles que le code de fonction, l'adresse de données et la gamme de données. Ensuite, il exécute l'action requise et envoie une réponse appropriée au code. Si une étape de ce processus échoue, une exception est renvoyée au demandeur. Le transport de données de ces requêtes est le PDU.

Le PDU Modbus

Le PDU comprend un code de fonction d'un octet suivi d'un maximum de 252 octets de données ayant une fonction spécifique.

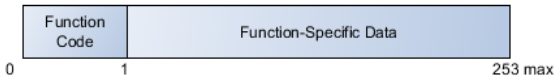

Figure 4. Le PDU Modbus

Le code de fonction est le premier élément à valider. Si le code de fonction n'est pas reconnu par l'appareil qui reçoit la requête, il répond par une exception. Si le code de fonction est accepté, l'appareil esclave commence à décomposer les données selon la définition de la fonction.

Étant donné que la taille du paquet est limitée à 253 octets, les appareils sont restreints quant à la quantité de données qui peut être transférée. Selon le code, les codes de fonction les plus courants transfèrent entre 240 et 250 octets de données réelles à partir du modèle de données esclave.

Exécution de la fonction esclave

Comme le définit le modèle de données, différentes fonctions sont définies pour accéder à différents blocs conceptuels de données. Une mise en œuvre courante est de faire en sorte que les codes accèdent à des emplacements de mémoire statique ; cependant, d'autres comportements sont disponibles. Par exemple, les codes de fonction 1 (lire des bobines) et 3 (lire des registres de maintien) peuvent accéder au même emplacement physique en mémoire. Par contraste, les codes de fonction 3 (lire des registres de maintien) et 16 (écrire des registres de maintien) peuvent accéder à des emplacements entièrement différents en mémoire. Ainsi, l'exécution de chaque code de fonction est mieux considérée comme faisant partie de la définition du modèle de données esclave.

Indépendamment du comportement observé, il est prévu que tous les appareils esclaves suivent un simple diagramme d'états pour chaque requête. La Figure 5 en montre un exemple pour le code 1, lire des bobines.

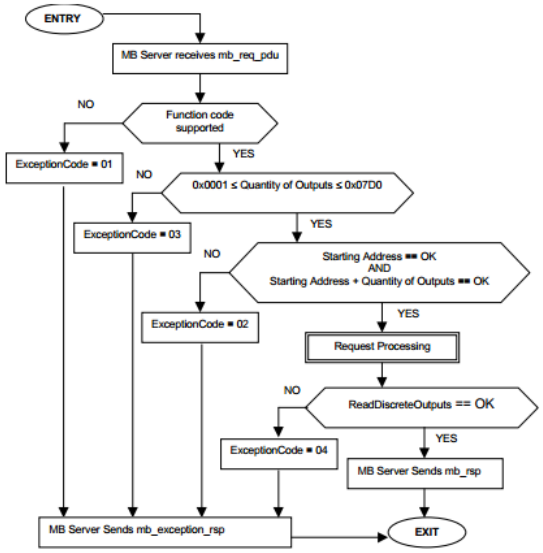

Figure 5. Diagramme d'états de Lire des bobines provenant des Spécifications du protocole Modbus

Chaque esclave doit valider le code de fonction, le nombre d'entrées, l'adresse de départ, la gamme totale et l'exécution de la fonction définie par l'esclave qui est en charge de la lecture.

Bien que les gammes d'adresses statiques soient illustrées dans le diagramme d'états ci-dessus, les besoins réels du système peuvent entraîner une variation de ces gammes par rapport aux nombres définis. Dans certains cas, les appareils esclaves ne peuvent pas transférer le nombre maximum d'octets défini par le protocole. C'est-à-dire qu'au lieu de permettre à un maître de demander des entrées 0x07D0, il ne peut répondre qu'avec 0x0400. De la même manière, un modèle de données esclave peut définir la gamme de valeurs de bobine acceptable comme adresse 0 à 500. Si un maître fait une requête pour 125 en commençant à l'adresse 0, ce n'est pas un problème. Par contre, si un maître fait la même requête en commençant à l'adresse 400, la bobine finale sera à l'adresse 525, ce qui est en-dehors de la gamme de cet appareil et entrainerait une exception 02 comme le définit le diagramme d'états.

Codes de fonction standard

La définition de chaque code de fonction standard se trouve dans les spécifications. Même pour les codes de fonction les plus courants, il existe des non concordances inhérentes entre les fonctions activées sur le maître et ce que l'esclave peut gérer. Pour répondre à cela, les premières versions des spécifications TCP Modbus définissaient trois classes de conformité. Le document Modbus Conformance Test Specification officiel ne référence pas ces classes et définit plutôt la conformité sur base de la fonction ; cependant, ces classes peuvent tout de même être pratiques pour la compréhension. Il est recommandé que toute documentation accompagne les spécifications de test et définisse leur conformité en fonction des codes supportés, plutôt qu'en fonction des classifications de l'ancien système.

Codes de classe 0

Les codes de classe 0 sont généralement considérés comme étant le strict minimum pour un appareil Modbus fonctionnel, dans la mesure où ils offrent à un maître la possibilité de lire et d'écrire dans le modèle de données.

Code	Description
3	Lire des registres multiples
16	Écrire des registres multiples

Table 4. Conformité des codes de classe 0

Codes de classe 1

Les codes de fonction de classe 1 incluent les autres codes nécessaires pour accéder à tous les types du modèle de données. Dans la définition initiale, cette liste incluait le code de fonction 7 (lire une exception). Cependant, ce code est défini par les spécifications actuelles comme un code série uniquement.

Code	Description
1	Lire des bobines
2	Lire des entrées discrètes
4	Lire des registres d'entrées
5	Écrire une bobine simple
6	Écrire un registre simple
7	Lire un statut d'exception (série uniquement)

Table 5. Conformité des codes de classe 1

Codes de classe 2

Les codes de fonction de classe 2 sont des fonctions plus spécialisées et moins couramment mises en œuvre. Par exemple, Lire/Écrire des registres multiples peut **faciliter** la réduction du nombre total de cycles de requête/réponse, mais le comportement peut tout de même être mis en application avec des codes de classe 0.

Code	Description
15	Écrire des bobines multiples
20	Lire un enregistrement de fichier
21	Écrire un enregistrement de fichier
22	Masquer Lire un registre
23	Lire/Écrire des registres multiples
24	Lire FIFO

Table 6. Conformité des codes de classe 2

"Modbus Encapsulated Interface"

Le code Modbus encapsulated interface (MEI), fonction 43, est utilisé pour encapsuler d'autres données au sein du paquet Modbus. Actuellement, deux nombres MEI sont disponibles, 13 (CANopen) et 14 (Device Identification).

La fonction 43/14 (Device Identification) est utile car elle permet le transfert d'un maximum de 256 objets uniques. Certains de ces objets sont prédéfinis et réservés, tels que le nom du fournisseur et le code du produit, mais les applications peuvent définir d'autres objets à transférer comme ensembles de données génériques.

Ce code n'est pas couramment mis en œuvre.

Exceptions

Les esclaves utilisent des exceptions pour indiquer un nombre de mauvaises conditions, allant de requêtes mal informées à des entrées incorrectes. Cependant, les exceptions peuvent aussi être générées comme des réponses d'application à des requêtes non valides. Les esclaves ne répondent pas aux requêtes générées avec une exception. Au lieu de cela, l'esclave ignore les requêtes incomplètes ou corrompues et commence à attendre un nouveau message entrant.

Les exceptions sont rapportées dans un format de paquet bien défini. Tout d'abord, un code de fonction est renvoyé au maître demandeur équivalent à un code de fonction initial, à l'exception de son ensemble de bits le plus significatif. Cela équivaut à ajouter 0x80 à la valeur du code de fonction initial. Au lieu des données normales associées à une réponse de fonction données, les réponses d'exception incluent un code d'exception simple.

Au sein du standard, les quatre codes d'exception les plus courants sont 01, 02, 03 et 04. Le Tableau 7 en est l'illustration avec les significations standard ajoutées à côté de chaque fonction.

Code d'exception	Signification
01	Le code de fonction reçu n'est pas supporté. Pour confirmer le code de fonction initial, soustrayez 0x80 à la valeur renvoyée.
02	La requête a tenté d'accéder à une adresse non valide. Dans le standard, cela peut se produire que si l'adresse de départ et le nombre requis de valeurs excèdent 2^{16}. Cependant, certains appareils peuvent restreindre cet espace d'adresse dans leur modèle de données.
03	La requête contenait des données incorrectes. Dans certains cas, cela signifie qu'il y a eu une mauvaise correspondance de paramètres, entre le nombre de registres envoyé et le champ "comptage d'octets", par exemple. Plus communément, le maître a demandé plus de données que ne le permet l'esclave ou le protocole. Par exemple, un maître peut ne lire que 125 registres de maintien à la fois, et les appareils limités en ressources peuvent restreindre cette valeur à encore moins de registres.
04	Une erreur irrécupérable s'est produite lors de la tentative de traitement de la requête. Il s'agit d'un code d'exception "catchall" qui indique que la requête était valide, mais que l'esclave n'a pas pu l'exécuter.

Table 7. Codes d'exception Modbus courants

Le diagramme d'états de chaque code de fonction devrait couvrir au moins le code d'exception 01 et inclut généralement les codes d'exception 04, 02 et 03, et tout autre code d'exception définit est facultatif.

3. Unité de données d'application (ADU)

En plus des fonctionnalités définies au sein du noyau PDU du protocole Modbus, vous pouvez utiliser de multiples protocoles réseau. Les protocoles les plus courants sont les protocoles série et TCP/IP, mais vous pouvez en utiliser d'autres tels que l'UDP. Pour transmettre des données nécessaires au protocole Modbus à travers ces couches, Modbus inclut un ensemble de variants ADU personnalisés pour chaque protocole réseau.

Fonctionnalités courantes

Modbus exige que certaines fonctionnalités fournissent une communication fiable. L'identifiant ou l'adresse de l'unité est utilisé dans chaque format ADU pour fournir des informations de routage à la couche d'application. Chaque ADU contient un PDU complet, qui inclut le code de fonction et les données associées pour une requête donnée. Pour des raisons de fiabilité, chaque message inclut des informations pour vérifier les erreurs. Enfin, tous les ADU disposent d'un mécanisme de détermination de début et de fin d'une requête, mais les implémentent différemment.

Formats standard

Les trois formats ADU standard sont le TCP, les terminaux distants (RTU) et l'ASCII. Les ADU RTU et ASCII sont traditionnellement utilisés à travers une ligne série, alors que le TCP est utilisé à travers des réseaux TCP/IP ou UDP/IP.

TCP/IP

Les ADU TCP comprennent le Modbus Application Protocol (MBAP) Header lié au PDU Modbus. Le MBAP est un en-tête polyvalent qui dépend d'une couche en réseau fiable. Le format de cet ADU, y compris l'en-tête, est illustré à la Figure 6.

Figure 6. L'ADU TCP/IP

Les champs de données de l'en-tête indiquent son utilisation. Tout d'abord, l'en-tête comprend un identifiant de transaction. Cela est important sur un réseau dans lequel de multiples requêtes peuvent être en suspens simultanément. C'est-à-dire qu'un maître peut envoyer les requêtes 1, 2 et 3. Par la suite, un esclave peut répondre dans l'ordre 2, 1, 3 et le maître peut faire correspondre les requêtes aux réponses et analyser les données de manière précise. Cela est utile pour les réseaux Ethernet.

L'identifiant de protocole est normalement zéro, mais vous pouvez l'utiliser pour étendre le comportement du protocole. Le champ longueur est utilisé par le protocole pour délimiter la longueur du paquet. L'emplacement de cet élément indique également la dépendance de ce format d'en-tête sur une couche en réseau fiable. Étant donné que les paquets TCP disposent de la vérification des erreurs intégrée et assurent la cohérence et la livraison des données, la longueur de paquet peut se trouver n'importe où dans l'en-tête. Sur un réseau moins fiable tel qu'un réseau série, un paquet peut se perdre ; par conséquent, même si le flux de données lu par l'application inclut des informations de transaction et de protocole valides, les informations de longueur corrompues rendraient l'en-tête invalide. Le TCP apporte une protection raisonnable contre cette situation.

L'identifiant d'unité est généralement non utilisé pour les appareils TCP/IP. Cependant, Modbus est un protocole tellement courant que de nombreuses passerelles sont développées et convertissent le protocole Modbus en un autre protocole. Dans le cas d'une utilisation initialement prévue, un TCP/IP Modbus à une passerelle série pourrait être utilisé pour autoriser la connexion entre de nouveaux réseaux TCP/IP et des réseaux série plus anciens. Dans un tel environnement, l'identifiant d'unité est utilisé pour déterminer l'adresse de l'appareil esclave pour lequel le PDU a été prévu.

Enfin, l'ADU inclut un PDU. La longueur de ce PDU est toujours limitée à 253 octets pour le protocole standard.

RTU

L'ADU RTU paraît beaucoup plus simple, comme le montre la Figure 7.

Figure 7. L'ADU RTU

Contrairement à l'ADU TCP/IP plus complexe, cet ADU ne comprend que deux informations en plus du PDU principal. Tout d'abord, une adresse est utilisée pour déterminer l'esclave pour lequel un PDU est prévu. Sur la plupart des réseaux, une adresse de 0 définit l'adresse de "diffusion". Cela signifie qu'un maître peut envoyer une commande de sortie à l'adresse 0 et tous les esclaves devraient traiter la requête mais aucun d'entre eux ne devrait répondre. Mis à part cette adresse, un CRC est utilisé pour garantir l'intégrité des données.

Cependant, la réalité de la situation dans des mises en œuvre plus modernes est bien plus simple. Des silences (des pauses sur le bus) entourent le paquet, soit des périodes pendant lesquelles il n'y a aucune communication dans le bus. Pour un débit en bauds de 9 600, ce temps est d'environ 4 ms. Le standard définit une longueur de silence minimale, indépendamment du débit en bauds, d'un peu moins de 2 ms.

Tout d'abord, cela représente un désavantage de performance dans la mesure où l'appareil doit attendre que le temps d'inactivité soit terminé avant que le paquet ne soit envoyé. Cependant, il existe un plus grand danger : l'introduction de différentes technologies utilisées pour un transfert série et des débits en bauds bien plus hauts que lors de l'introduction du standard. Avec un câble de conversion USB-série, par exemple, vous n'avez aucun contrôle sur la mise en paquets et le transfert de données. Les tests montrent que l'utilisation d'un câble USB-série avec un driver NI-VISA introduit de grands espacements de tailles variables dans le flux de données, et ces espacements (périodes de silence) trompent le code conforme aux spécifications en lui faisant croire qu'un message est complet. Comme le message n'est pas complet, cela engendre généralement un CRC invalide et une interprétation de l'appareil indiquant que l'ADU est corrompu.

En plus de problèmes de transmission, les technologies de driver modernes extraient la communication série et exigent en général un mécanisme d'interrogation de la part du code d'application. Par exemple, ni la Classe .NET Framework 4.5 SerialPort ni le driver NI-VISA ne fournit un mécanisme de détection des silences sur une ligne en série, sauf en interrogeant les octets sur le port. Cela résulte en une dégradation des performances (si l'interrogation se fait trop lentement) ou en une utilisation élevée du CPU (si l'interrogation se fait trop rapidement).

Une méthode courante pour résoudre ces problèmes est de briser la couche d'abstraction entre le PDU Modbus et la couche en réseau. C'est-à-dire que le code série interroge le paquet du PDU Modbus pour déterminer le code de fonction. Combinée à d'autres données du paquet, la longueur du paquet restant peut être découverte et utilisée pour déterminer la fin du paquet. Avec ces informations, un délai bien plus long peut être utilisé, permettant des espacements de transmission, ce qui fait que l'interrogation au niveau de l'application peut avoir lieu bien plus lentement. Ce mécanisme est recommandé pour les nouveaux développements. Le code qui n'emploie pas cette méthode peut faire face à un nombre bien plus élevé que prévu de paquets "corrompus".

ASCII

L'ADU ASCII est plus complexe que le RTU, comme le montre la Figure 8, mais il évite aussi de nombreux problèmes du paquet RTU. Cependant, il comprend aussi certains désavantages.

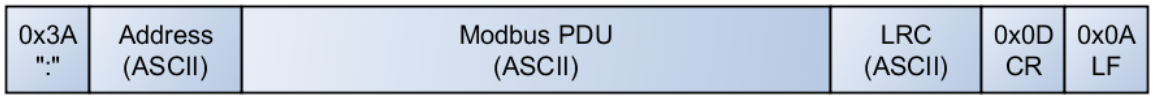

Figure 8. L'ADU ASCII

Résolvant le problème de détermination de la taille du paquet, l'ADU ASCII a un début et une fin bien définis et uniques pour chaque paquet. C'est-à-dire que chaque paquet commence par ":" et se termine par un retour chariot (CR) et un retour à la ligne (LF). De plus, les API série telles que NI-VISA et la Classe .NET Framework SerialPort peuvent facilement lire les données d'un buffer jusqu'à ce qu'un caractère spécifique (comme CR/LF) soit reçu. Ces fonctionnalités facilitent le traitement efficace du flux de données sur la ligne série au sein du code d'application moderne.

L'inconvénient de l'ADU ASCII est que toutes les données sont transférées comme caractères hexadécimaux encodés en ASCII. C'est-à-dire qu'au lieu d'envoyer un seul octet pour le code de fonction 3, 0x03, il envoie les caractères ASCII "0" et "3," ou 0x30/0x33. Cela rend le protocole plus lisible pour l'œil humain, mais signifie également que deux fois plus de données doivent être transférées à travers le réseau série et que la fonction d'envoi et de réception d'applications doit pouvoir analyser les valeurs ASCII.

Extensibilité du protocole Modbus

Modbus, standard relativement simple et ouvert, peut être modifié pour répondre aux besoins d'une application donnée. Cela est plus courant pour les communications entre une IHM et un PLC ou PAC, étant donné qu'il s'agit d'une situation dans laquelle une organisation unique contrôle les deux points d'extrémité du protocole. Les développeurs de capteurs, par exemple, sont plus à même d'adhérer au standard écrit car ils ne contrôlent généralement que la mise en œuvre de leur esclave, et l'interopérabilité est souhaitable.

En général, modifier le protocole n'est pas recommandé. Cette section est simplement considérée comme la reconnaissance des mécanismes que d'autres ont utilisé pour ajuster le comportement du protocole.

4. Nouveaux codes de fonction

Certains codes de fonction sont définis, mais le standard Modbus vous permet de développer des codes de fonctions supplémentaires. Plus particulièrement les codes de fonction 1 à 64, 73 à 99 et 111 à 127 sont des codes publics réservés et garantis comme étant uniques. Les codes restants, 65 à 72 et 100 à 110, sont réservés pour une utilisation définie par l'utilisateur. Avec ces codes définis par l'utilisateur, vous pouvez utiliser n'importe quelle structure de données. Les données peuvent même excéder la limite standard de 253 octets pour le PDU Modbus, mais l'application entière devrait être validée pour garantir que d'autres couches fonctionnent comme prévu quand le PDU excède la limite standard. Les codes de fonction supérieurs à 127 sont réservés aux réponses d'exception.

5. Couches réseau

Modbus peut s'exécuter sur de nombreuses couches réseau en plus des couches série et TCP. Une mise en œuvre potentielle est l'UDP car il correspond bien au style de communication Modbus. En son noyau, Modbus est un protocole basé message, donc la capacité de l'UDP à envoyer un paquet bien défini d'informations sans autre information d'application supplémentaire, comme un caractère de début ou une longueur, rend Modbus extrêmement simple à mettre en œuvre. Plutôt que de nécessiter un ADU supplémentaire ou de réutiliser un ADU existant, les paquets PDU Modbus peuvent être envoyés en utilisant une API UDP standard et peuvent être reçus entièrement formés à l'autre extrémité. Bien que le TCP soit avantageux pour certains protocoles grâce au système de "handshaking" intégré, Modbus effectue de la reconnaissance au niveau de la couche d'application. Cependant, l'utilisation de l'UDP de cette façon élimine le champ de l'identifiant de transaction de l'ADU TCP, ce qui empêche d'effectuer simultanément de multiples transactions laissées en suspens. Par conséquent, le maître doit être un maître synchrone ou le paquet UDP doit avoir un identifiant afin d'aider le maître à organiser les requêtes et réponses. Une mise en œuvre suggérée serait d'utiliser l'ADU TCP/IP sur une couche réseau UDP.

6. Modifications d'un ADU

Enfin, une application pourrait choisir de modifier un ADU ou d'utiliser des portions inutilisées d'un ADU existant tel qu'un TCP. Par exemple, TCP définit un champ de longueur de 16 bits, un protocole de 16 bits et un identifiant d'unité de 8 bits. Étant donné que le PDU Modbus le plus large est de 253 octets, l'octet élevé du champ de longueur est toujours zéro. Pour Modbus/TCP, le champ de protocole et l'identifiant d'unité sont toujours zéro. Une extension simple du protocole peut envoyer trois paquets simultanément en changeant le champ de protocole à un nombre autre que zéro et en utilisant les deux octets non utilisés (l'identifiant d'unité et l'octet élevé du champ de longueur) pour envoyer les longueurs des deux PDU supplémentaires (voir Figure 9).

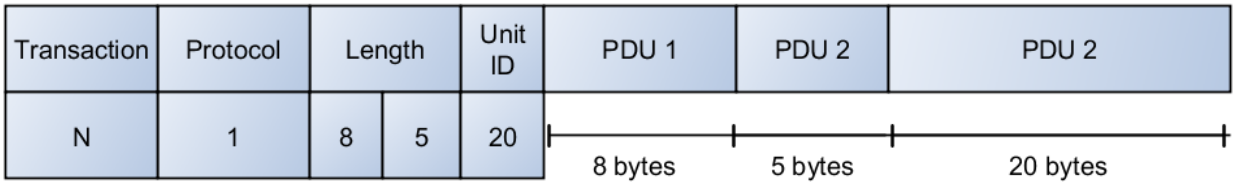

Figure 9. Exemple de modification de l'ADU TCP

Ressources supplémentaires

Spécifications du protocole d'application Modbus

Pourquoi l'OPC UA a de l'importance ?

À propos de l'OPC

EtherNet/IP — CIP sur la technologie Ethernet

Support Digi

Bibliothèque Modbus Java