

AVR076 : AVR[®] CAN – 4Kバイトローダ

1. 要点

- CAN規約
 - ・ 物理層として使用される制御器域網(CAN)
 - ・ 再設定可能な7つのISP CAN識別子
 - ・ 自動ビット速度
- 実装書き込み
 - ・ フラッシュメモリとEEPROMの読み書き
 - ・ デバイスID読み込み
 - ・ 完全なチップ消去
 - ・ ISP命令からの安全な設定
 - ・ 遠隔応用開始命令
- 応用書き込み
 - ・ 255までの節点(ノード)
 - ・ 再配置可能な16個の予約識別子
- 応用書き込みインターフェース
 - ・ フラッシュ書き込みAPI(応用領域)
- 他の規約を開放
 - ・ LIN
 - ・ RS232
 - ・ SPI
 - ・ TWI
 - ・ その他
- FLIPインターフェース

2. 説明

この資料は“減量”CANポートローダの機能だけでなく、チップ上のフラッシュメモリとEEPROMでの操作を効率的に実行するための規約も記述します。

このポートローダは“実装書き込み(ISP:In-System Programming)”を実装します。ISPはシステムからデバイスを取り去ることなく、そして予め書き込まれた応用の必要なく、マイクロコントローラのチップ上のフラッシュメモリとEEPROMの書き込みや書き換えを使用者に許します。

CANポートローダはCAN網を通してホストとの通信を管理することができます。それはチップ上のフラッシュメモリとEEPROMでのアクセスや要求した操作を実行することもできます。

応用書き込み機能は255までのCAN節点(ノード)の管理が利用可能です。

特殊な入口(フラッシュAPI)が使用者に対して利用可能です。



8ビット **AVR[®]**
マイクロコントローラ

応用記述

AT90CAN32
AT90CAN64
AT90CAN128

ATmega16M1
ATmega32M1
ATmega32C1
ATmega64C1

4Kバイト(減量)
CANポートローダ



本書は一般の方々の便宜のため有志により作成されたもので、ATMEL社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

Rev. 8247A-08/09, 8247AJ1-12/13

3. ブート ロード環境

CANブートローダはチップ上のフラッシュメモリの“ブートローダフラッシュ領域”に格納されます。ブートローダの大きさは4Kバイト未満で、故に物理的な“ブートローダフラッシュ領域”は全体の半分だけです(訳注:4Kバイト領域のデバイスは全部)。応用プログラムの大きさは“応用フラッシュ領域”+4Kバイト(訳注:8Kバイト領域のデバイスのみ)と等しいかまたはそれ以下でなければなりません(表3-1.と表3-2.参照)。

表3-1. AT90CANxxx系列-メモリ割付 (バイトアドレス指定)

メモリ種別		AT90CAN128	AT90CAN64	AT90CAN32
フラッシュメモリ	容量	128Kバイト	64Kバイト	32Kバイト
	アドレス範囲	\$00000~\$1FFFF	\$00000~\$0FFFF	\$00000~\$07FFF
“応用フラッシュ領域”	容量	120Kバイト	56Kバイト	24Kバイト
	アドレス範囲	\$00000~\$1DFFF	\$00000~\$0DFFF	\$00000~\$05FFF
“ブートローダフラッシュ領域”	容量	8Kバイト		
	アドレス範囲	\$1E000~\$1FFFF	\$0E000~\$0FFFF	\$06000~\$07FFF
“ブートローダリセットアドレス” (注1)	最小(第1)ブート	\$1FC00	\$0FC00	\$07C00
	第2ブート	\$1F800	\$0F800	\$07800
	第3ブート	\$1F000 (注2)	\$0F000 (注2)	\$07000 (注2)
	最大(第4)ブート	\$1E000	\$0E000	\$06000
EEPROM	容量	4Kバイト	2Kバイト	1Kバイト
	アドレス範囲	\$0000~\$0FFF	\$0000~\$07FF	\$0000~\$03FF

注1: “ブートローダリセットアドレス”は“BOOTSZ”ヒューズビットに依存します。メモリ(フラッシュメモリ、EEPROMなど)の動きのより多くの詳細についてはデータシートを参照してください。

注2: CANブートローダリセットアドレス

表3-2. ATmegaxxM1/C1系列-メモリ割付 (バイトアドレス指定)

メモリ種別		ATmega64M1/C1	ATmega32M1/C1	ATmega16M1
フラッシュメモリ	容量	64Kバイト	32Kバイト	16Kバイト
	アドレス範囲	\$0000~\$FFFF	\$0000~\$7FFF	\$0000~\$3FFF
“応用フラッシュ領域”	容量	\$0000~\$7FFF	28Kバイト	12Kバイト
	アドレス範囲	\$0000~\$DFFF	\$0000~\$6FFF	\$0000~\$2FFF
“ブートローダフラッシュ領域”	容量	8Kバイト	4Kバイト	
	アドレス範囲	\$E000~\$FFFF	\$7000~\$7FFF	\$3000~\$3FFF
“ブートローダリセットアドレス” (注1)	最小(第1)ブート	\$FC00	\$7E00	\$3E00
	第2ブート	\$F800	\$7C00	\$3C00
	第3ブート	\$F000 (注2)	\$7800	\$3800
	最大(第4)ブート	\$E000	\$7000 (注2)	\$3000 (注2)
EEPROM	容量	2Kバイト	1Kバイト	512バイト
	アドレス範囲	\$0000~\$07FF	\$0000~\$03FF	\$0000~\$01FF

注1: “ブートローダリセットアドレス”は“BOOTSZ”ヒューズビットに依存します。メモリ(フラッシュメモリ、EEPROMなど)の動きのより多くの詳細についてはデータシートを参照してください。

注2: CANブートローダリセットアドレス

3.1. デバイス ヒューズ設定

更なる説明についてはデバイスのデータシートを参照してください。

図3-1. デバイス ヒューズ設定 - 第1部

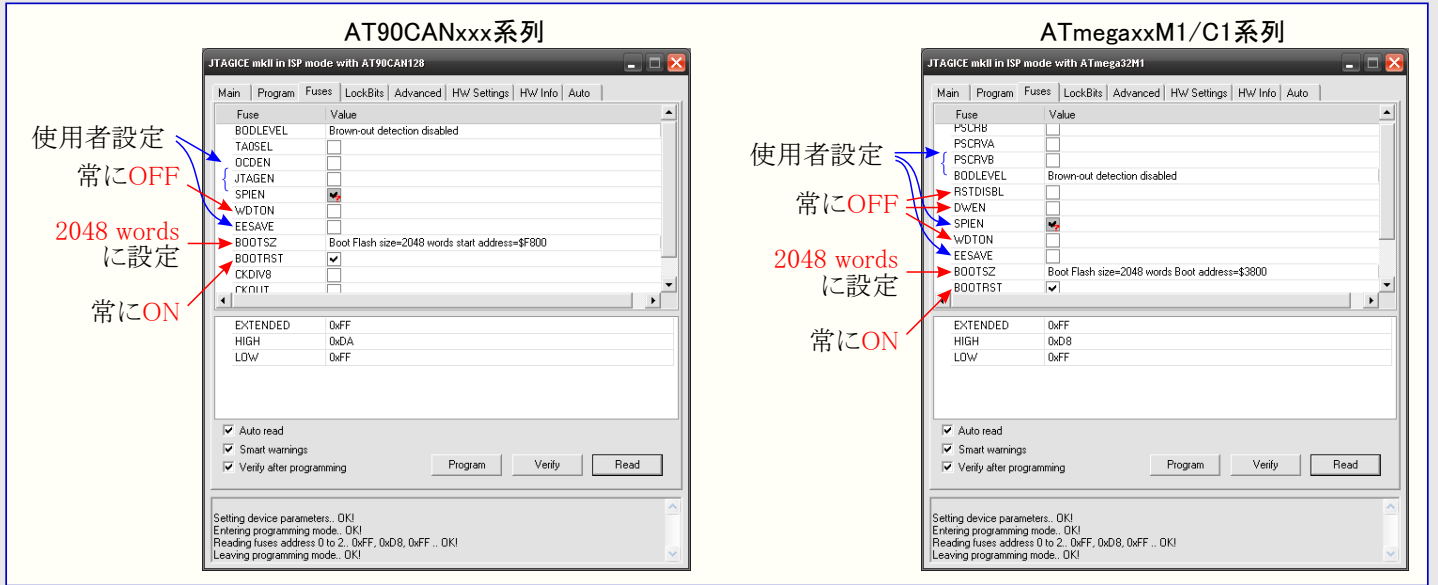
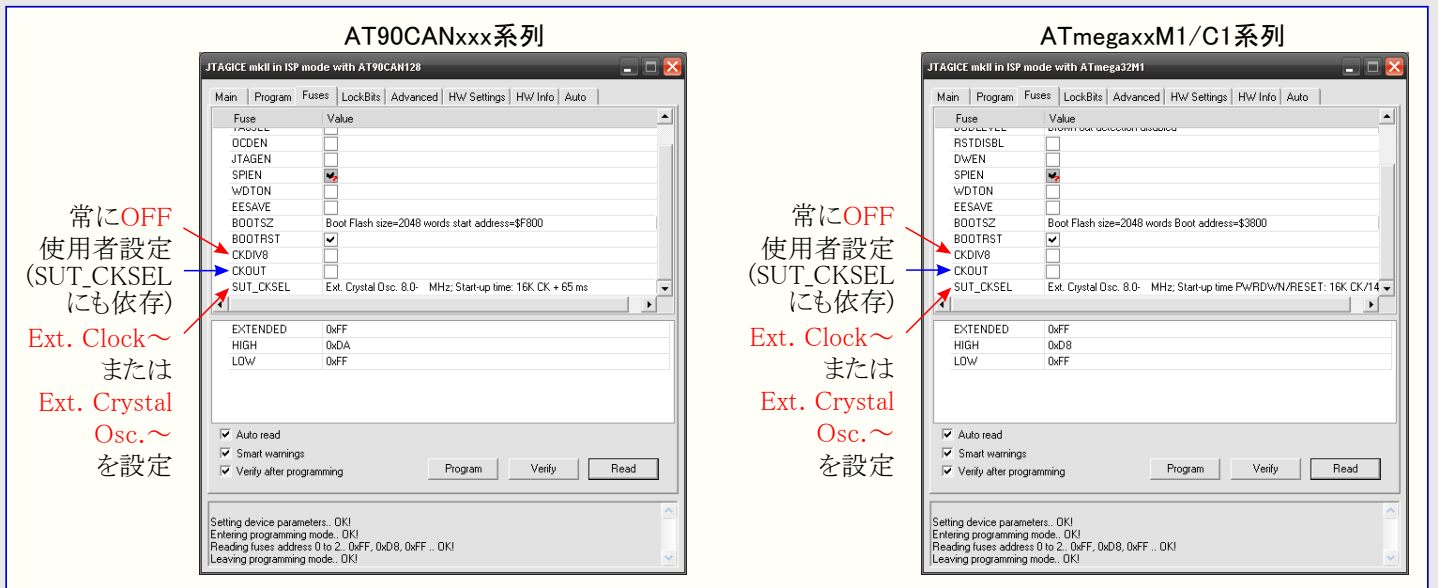


図3-2. デバイス ヒューズ設定 - 第2部



必須ヒューズ設定の要約:

ヒューズ上位バイト: **BOOTRST** プログラム(0)

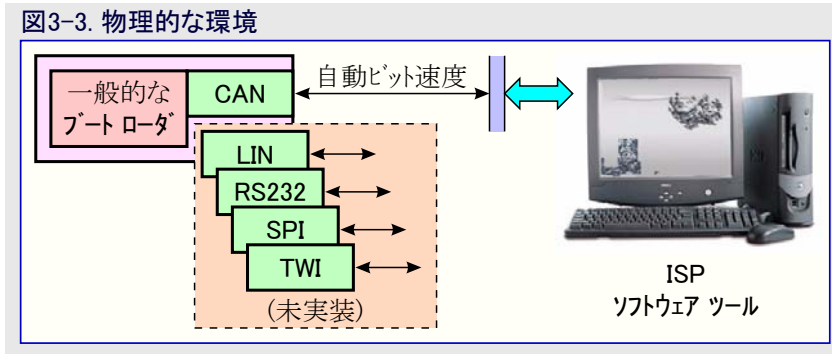
BOOTSZ1,0 2048語に設定

WDTON 非プログラム(1)

ヒューズ下位バイト: **CKSEL3~0** CAN必要条件に合う高い精度を持つクロックを選択するように設定(内蔵RC発振器は合致しません。)

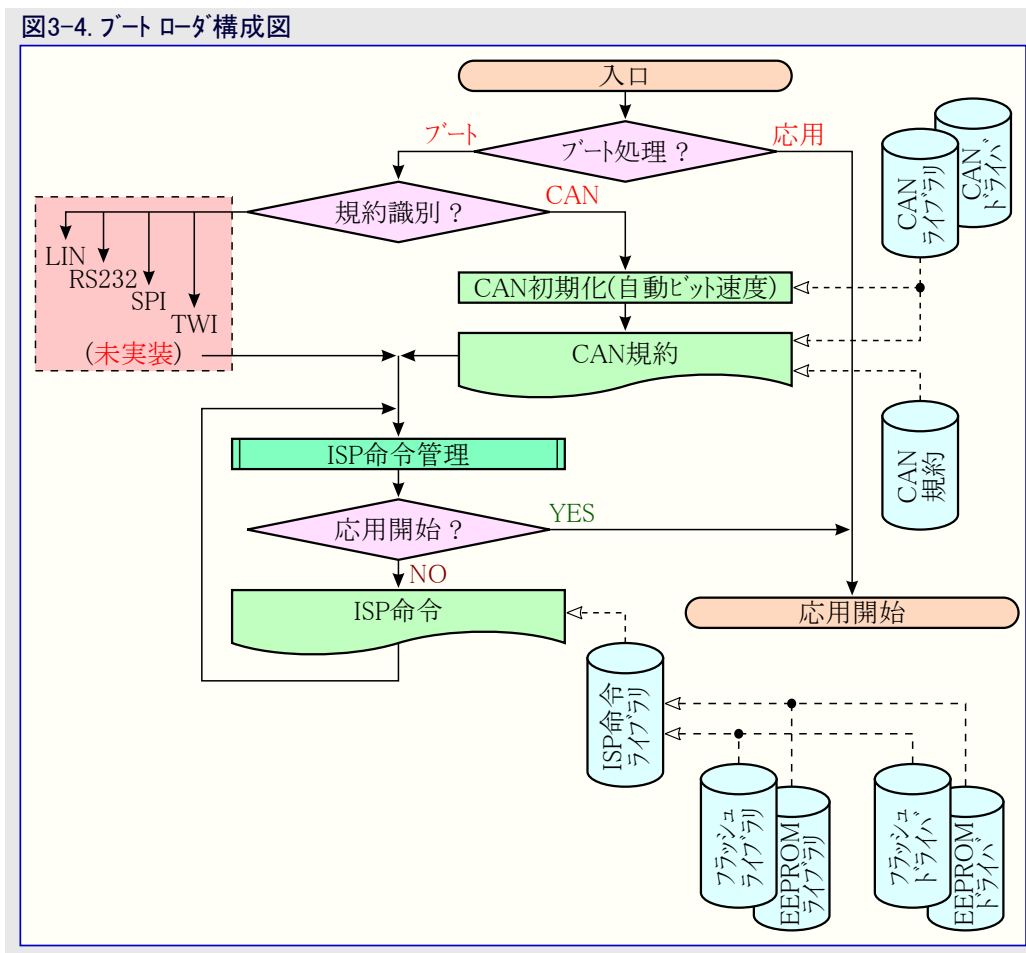
3.2. 物理的な環境

一般的なブートローダはCANインターフェースを通してホスト(またはPC)を処理します。一般的なブートローダは他のインターフェース(LIN,RS232,SPI,TWI,~)に接続することができるサービスです。



3.3. ブートローダ説明

3.3.1. 概要



3.3.2. 入口点

1つの"入口点"だけが利用可能で、それはブートローダへの入口点です。デバイスの"BOOTRST"ヒューズが設定(0)されなければなりません。リセット後、デバイスの"プログラムカウンタ"は"ブートリセットアドレス"に設定されます(2頁の表3-1. AT90CANxxx系列-メモリ割付(バイトアドレス指定)と表3-2. ATmegaxxM1/C1系列-メモリ割付(バイトアドレス指定)参照)。この"入口点"はブートローダの"ブート処理"を初期化します。

3.3.3. ブート処理

ブートローダの“ブート処理”は応用またはブートローダ自身の開始を許します。これは次の2つの変数に依存します。

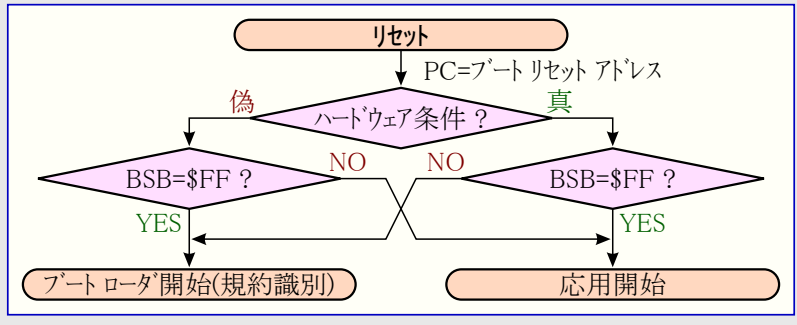
- ・ “ハードウェア条件”

ハードウェア条件はこのブートローダでHWCBと名付けられたデバイスの入力ピンとその活性レベル(例:INT0/PIND.0, active low)によって定義されます。

- ・ “ブート状態バイト”

ブート状態バイト“BSB(Boot Status Byte)”は“ブートローダ形態設定メモリ”に属します(8頁の「4.5.4.1. ブート状態バイト – “BSB”」項参照)。この既定値は\$FFです。ISP命令はこの値の変更を許します。

図3-5. ブート処理構成図



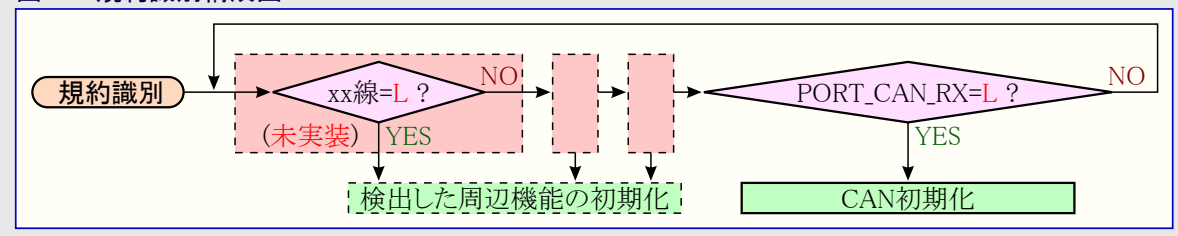
3.3.4. 規約識別

ブートローダの“規約識別”は使用される何かの規約、CANまたは他の規約を選びます。媒体での活動を検知するために物理(信号)線のポーリングが行われます。それらの線は次の通りです。

- ・ PORT_CAN_RX：以下でポーリングが行われます。
 - AT90CANxxx系列についてはRXCAN/PIND.6
 - ATmegaxxM1/C1系列についてはRXCAN/PINC.3
- ・ (使用者定義インターフェース)

PORT_CAN_RX線のLowレベルがCAN周辺機能の初期化を開始します。

図3-6. 規約識別構成図



3.3.5. CAN初期化

ホストとの通信に使用されるCANは以下の形態設定を持ちます。

- ・ 規格：CAN形式2.0A (11ビット識別子)
- ・ フレーム：データフレーム
- ・ ビット速度：特別バイト(EB:Extra Byte)に依存(9頁の「特別バイト – “EB”」をご覧ください。)
 - “EB”=\$FF：ソフトウェア自動ビット速度を使用
 - “EB”≠\$FF：CANビット速度設定にビットタイミング制御1～3を使用(9頁の「ビットタイミング制御1～3 – “BTC1～3”」をご覧ください。)

初期化処理は各デバイスリセット後に実行されなければなりません。ホストは節点(ノード)を選択するためにデータフレームを送ることによって通信を始めます。自動ビット速度の場合、これがCANビット速度を見つけるため、ブートローダを助けます。CAN規格は異常応答を持つフレームは自動的に再送されると言っています。この機能と“聴取”形態に設定されるべきCAN周辺機能の能力が自動ビット速度によって使用されます。一旦同期フレームがどの異常もなく受信されると、“聴取”形態の開放によって応答で劣性レベルが印加されます。

ソフトウェア自動ビット速度はデバイスで設定されたシステムクロック(CKIO)に従って広範囲のポーレートを支援します(“config.h”ファイルで“FOC”定義参照)。この(自動ビット速度)機能は多数のCAN節点(ノード)を持つCAN網で保証されません。この場合に於いては固定ポーレート(“EB”≠\$FF)が推奨されます。

3.3.6. CAN規約概要

“CAN規約”は直列線(CANバス)上の上位規約です。
これは本資料の特別な節で記述されます(11頁の「CAN規約とISP命令」をご覧ください)。

3.3.7. ISP命令概要

“CAN規約”は“ISP命令”を復号します。“ISP命令”の組はどの規約とも明らかに無関係です。
これは本資料の特別な節で記述されます(11頁の「CAN規約とISP命令」をご覧ください)。

3.3.8. ブートローダからの出力

ブートローダからの出力はISP命令:“応用開始”を受信した後で実行されます。(11頁の「CAN規約とISP命令」をご覧ください)。

4. メモリ空間定義

ブートローダは6つまでの独立したメモリ空間を支援します。それらの各々は低位アクセス規約(ドライバ)が異なり得るため、コード番号(対応する規約領域で報告される値)を受け取ります。

メモリ空間のアクセスはバイトアクセス(換言すると、バイトアドレスで与えられたアドレス)です。

表4-1. メモリ空間コード番号

空間 (注)	コード番号	アクセス
フラッシュメモリ	0	読み書き
EEPROMデータメモリ	1	読み書き
識票	2	読み込み専用
ブートローダ情報	3	読み込み専用
ブートローダ形態設定	4	読み書き
デバイスレジスタ	5	読み込み専用

注: 時々、識別は物理的ではありません(例:“識票”は“ブートローダ情報”だけでなく“ブートローダフラッシュ領域”のコードの部分でもあります)。

4.1. フラッシュメモリ空間

ブートローダによって管理されるフラッシュメモリ空間はデバイスのフラッシュメモリの部分です。それは“応用フラッシュ領域”です。

表4-2. フラッシュメモリ空間 (コード番号0)

フラッシュメモリ空間	AT90CAN128	AT90CAN64 ATmega64M1 ATmega64C1	AT90CAN32 ATmega32M1 ATmega32C1	ATmega16M1
容量	124Kバイト	60Kバイト	28Kバイト	12Kバイト
アドレス範囲	\$00000~\$1EFFF	\$0000~\$EFFF	\$0000~\$6FFF	\$0000~\$2FFF
ページ数 (注)	2	1	1	1

注: ページパラメータはブートローダとデバイス自体で異なります。

4.1.1. 読み書き

“ISP読み込み”または“ISP書き込み”命令だけが64Kバイトのページ内にバイトアドレス指定形態でフラッシュメモリ空間にアクセスします(表4-2. フラッシュメモリ空間 (コード番号0)参照)。特定のISP命令が異なるページの選択を許します。

読み書き命令が起きる間にソフトウェア保護バイト“SSB(Software Security Byte)”が設定されている場合、ブートローダは“デバイス保護”異常を返します(9頁の「4.5.4.2. ソフトウェア保護バイト - “SSB”」項参照)。

4.1.2. 消去

“ISP消去”命令はフラッシュメモリ空間の完全消去(全バイト=\$FF)です。この操作は例えソフトウェア保護バイト“SSB”が設定されていても利用可能です。この操作の終わりで、ソフトウェア保護バイト“SSB”は保護レベル0にリセットされます(9頁の「4.5.4.2. ソフトウェア保護バイト - “SSB”」項)。

4.1.3. 制限

フラッシュメモリ空間でのISP命令はブートローダに於いて無効です(“ブートローダフラッシュ領域”で無効)。

ISP命令に対するフラッシュメモリ空間(コード番号0)の大きさは「表4-2. フラッシュメモリ空間 (コード番号0)」で与えられます。

4.2. EEPROMデータメモリ

ブートローダによって管理されるEEPROMデータメモリ空間はデバイスのEEPROMです。

表4-3. EEPROMデータメモリ空間 (コード番号1)

EEPROM データメモリ空間	AT90CAN128	AT90CAN64 ATmega64M1 ATmega64C1	AT90CAN32 ATmega32M1 ATmega32C1	ATmega16M1
容量	4Kバイト	2Kバイト	1Kバイト	512バイト
アドレス範囲	\$0000~\$0FFF	\$0000~\$07FF	\$0000~\$03FF	\$0000~\$01FF
ページ数	(ページなし)			

4.2.1. 読み書き

EEPROMデータメモリ空間は不揮発性データメモリとして使用されます。“ISP読み込み”または“ISP書き込み”命令はこの空間に(ページなしで)バイト単位でアクセスします。

読み書き命令が起きる間にソフトウェア保護バイト“SSB(Software Security Byte)”が設定されている場合、ブートローダは“デバイス保護”異常を返します(9頁の「4.5.4.2. ソフトウェア保護バイト – “SSB”」項参照)。

4.2.2. 消去

“ISP消去”命令はEEPROMデータメモリ空間の完全消去(全バイト=\$FF)です。この操作はソフトウェア保護バイト“SSB”がリセットされている場合にだけ利用可能です(9頁の「4.5.4.2. ソフトウェア保護バイト – “SSB”」項)。

4.2.3. 制限

ISP命令に対するEEPROMデータメモリ空間(コード番号1)の大きさは「表4-3. EEPROMデータメモリ空間 (コード番号1)」で与えられます。

4.3. 識票

ブートローダによって管理される識票空間はブートローダのコードに含まれます。これは“ブートローダフラッシュ領域”内です。

表4-4. 識票空間 (コード番号2)

識票空間		AT90 CAN128	AT90 CAN64	AT90 CAN32	ATmega 64M1	ATmega 32M1	ATmega 16M1	ATmega 64C1	ATmega 32C1
製造者符号	アドレス:\$00 (読み込み専用)	\$1E							
系統符号	アドレス:\$01 (読み込み専用)	\$81			\$84			\$86	
製品名	アドレス:\$02 (読み込み専用)	\$97	\$96	\$95	\$96	\$95	\$94	\$96	\$95
製品改訂	アドレス:\$03 (読み込み専用)	\$00							
ページ数		(ページなし)							

4.3.1. 読み書き

“ISP読み込み”命令はこの空間に(ページなしで)バイト単位でアクセスします。

アクセス保護はこの読み込み専用空間で全く提供されません。

4.3.2. 消去

読み込み専用空間のために当てはまりません。

4.3.3. 制限

ISP命令に対する識票空間(コード番号2)の詳細は「表4-4. 識票空間 (コード番号2)」で与えられます。

4.4. ブートローダ情報

ブートローダによって管理されるブートローダ情報空間はブートローダのコードに含まれます。これは“ブートローダフラッシュ領域”内です。

表4-5. ブートローダ情報空間 (コード番号3)

ブートローダ情報空間		AT90 CAN128	AT90 CAN64	AT90 CAN32	ATmega 64M1	ATmega 32M1	ATmega 16M1	ATmega 64C1	ATmega 32C1
ブートローダ改訂	アドレス:\$00 (読み込み専用)	\$01							
ブート識別1	アドレス:\$01 (読み込み専用)	\$D1							
ブート識別2	アドレス:\$02 (読み込み専用)	\$D2							
ページ数		(ページなし)							

4.4.1. 読み書き

“ISP読み込み”命令はこの空間に(ページなしで)バイト単位でアクセスします。
アクセス保護はこの読み込み専用空間で全く提供されません。

4.4.2. 消去

読み込み専用空間のために当てはまりません。

4.4.3. 制限

ISP命令に対するブートローダ情報空間(コード番号3)の詳細は「表4-5. ブートローダ情報空間(コード番号3)」で与えられます。

4.4.4. ブートローダ情報バイト説明

4.4.4.1. ブート改訂

ブート改訂：読み込み専用アドレス=\$00、値=\$01

4.4.4.2. ブート識別1,2

ブート識別1,2：読み込み専用アドレス=\$01,\$02、値=\$D1,\$D2

4.5. ブートローダ形態設定

ブートローダによって管理されるブートローダ形態設定空間は“ブートローダフラッシュ領域”に含まれます。

表4-6. ブートローダ形態設定空間(コード番号4)

ブートローダ形態設定空間			既定値
ブート状態バイト	“BSB”	アドレス:\$00	\$FF
ソフトウェア保護バイト	“SSB”	アドレス:\$01	\$FF
特別バイト	“EB”	アドレス:\$02	\$FF(注1)
ビットタイミング制御1	“BTC1”	アドレス:\$03	\$FF(注2)
ビットタイミング制御2	“BTC2”	アドレス:\$04	\$FF(注2)
ビットタイミング制御3	“BTC3”	アドレス:\$05	\$FF(注2)
節点(ノード)番号	“NNB”	アドレス:\$06	\$FF(注3)
CAN再配置ID区分	“CRIS”	アドレス:\$07	\$00
開始アドレス下位	“SA_L”	アドレス:\$08	\$00
開始アドレス上位	“SA_H”	アドレス:\$09	\$00
ページ数			(ページなし)

注1: 効力については9頁の「特別バイト – “EB”」をご覧ください。

注2: 効力については9頁の「ビットタイミング制御1~3 – “BTC1~3”」をご覧ください。

注3: 効力については9頁の「(CAN)節点(ノード)番号 – “NNB”」をご覧ください。

4.5.1. 読み書き

“ISP読み込み”命令はこの空間に(ページなしで)バイト単位でアクセスします。

アクセス保護はソフトウェア保護バイトでだけ提供されます(9頁の「4.5.4.2. ソフトウェア保護バイト – “SSB”」項参照)。

4.5.2. 消去

“ISP消去”命令はこの空間に対して**利用できません**。

4.5.3. 制限

ISP命令に対するブートローダ形態設定空間(コード番号4)の詳細は「表4-5. ブートローダ形態設定空間(コード番号4)」で与えられます。

4.5.4. ブートローダ形態設定バイト説明

4.5.4.1. ブート状態バイト – “BSB”

ブートローダのブート状態バイトは応用またはブートローダの開始を制御するために、“ブート処理”(5頁の「3.3.3. ブート処理」項)で使用されます。ハードウェア条件が全く設定されない場合、ブート状態バイトの既定値(\$FF)はブートローダの開始を強制し、さもなければ(ブート状態バイト ≠ \$FF且つハードウェア条件なし)応用が始まります。

4.5.4.2. ソフトウェア保護バイト – “SSB”

ブートローダは使用者アクセスまたはISPアクセスから自身と応用を保護するためにソフトウェア保護バイト“SSB”を持ちます。これはフラッシュメモリとEEPROM空間と自身を保護します。

ソフトウェア保護バイト“SSB”での“ISP書き込み”命令はより高い優先レベルだけを書くことができます。3つの保護レベルがあります。

表4-7. 保護レベル

レベル	保護	“SSB”	注釈
0	NO_SECURITY	\$FF	<ul style="list-style-type: none"> これが既定レベルです。 レベル1またはレベル2だけがレベル0を上書きすることができます。
1	WR_SECURITY	\$FE	<ul style="list-style-type: none"> レベル1ではフラッシュメモリとEEPROMの空間で書くことができません。 ブートローダは異常メッセージを返します。 レベル2だけがレベル1を上書きすることができます。
2	RD_WR_SECURITY	\$FC	<ul style="list-style-type: none"> フラッシュメモリとEEPROMの空間での全ての読み書きアクセスが許されません。 ブートローダは異常メッセージを返します。 フラッシュメモリ空間での“ISP消去”命令だけがソフトウェア保護バイトを(レベル0に)リセットします。

下表はSSBレベルに関して認められた活動を与えます。

表4-8. ソフトウェア保護バイト“SSB”に関して許された活動

ISP命令	NO_SECURITY	WR_SECURITY	RD_WR_SECURITY
フラッシュメモリ空間消去	許可	許可	許可
EEPROM空間消去	許可	-	-
フラッシュメモリ空間書き込み	許可	-	-
EEPROM空間書き込み	許可	-	-
フラッシュメモリ空間読み込み	許可	許可	-
EEPROM空間読み込み	許可	許可	-
(“SSB”を除き)ブートローダ形態設定書き込み	許可	-	-
ブートローダ形態設定読み込み	許可	許可	許可
“SSB”書き込み	許可	より高いレベルのみ	-
ブートローダ情報読み込み	許可	許可	許可
識票読み込み	許可	許可	許可
(何れかのメモリの)空白検査	許可	許可	許可
メモリ空間変更	許可	許可	許可

4.5.4.3. 特別バイト – “EB”

特別バイトはCAN初期化を自動ビット速度または固定のCANビットタイミングに切り替えるのに用いられます。

- “EB”=\$FF：ソフトウェア自動ビット速度使用
- “EB”≠\$FF：(自動ビット速度ではなく)CAN周辺機能のCANビットタイミングレジスタを設定するのにブートローダ形態設定空間のCANBT1～3バイトを使用

4.5.4.4. ビットタイミング制御1～3 – “BTC1～3”

“EB”≠\$FFの時にCAN周辺機能のCANビットタイミングレジスタを設定するのにブートローダ形態設定空間のビットタイミング制御1～3バイト(“BTC1”, “BTC2”, “BTC3”)が使用されます(自動ビット速度ではありません)。

これらのバイトを構成設定する方法は10頁の「4.6.4.1. CANBT1～3レジスタ」項で記述されます。

4.5.4.5. (CAN)節点(ノード)番号 – “NNB”

11頁の「CAN規約とISP命令」をご覧ください。

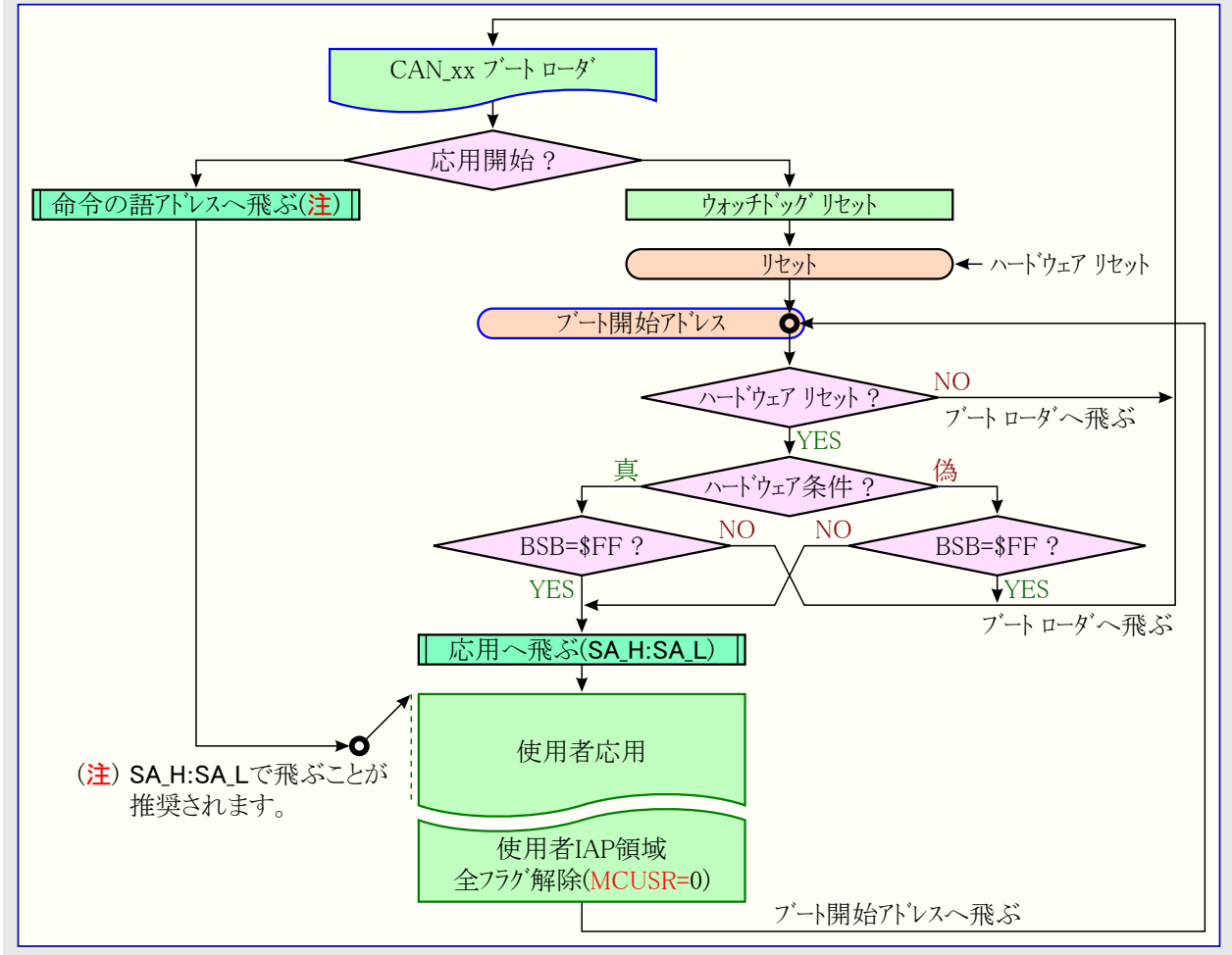
4.5.4.6. CAN再配置ID区分 – “CRIS”

11頁の「CAN規約とISP命令」をご覧ください。

4.5.4.7. 開始(応用)アドレス上位と下位 – “SA_H”と“SA_L”

11頁の「CAN規約とISP命令」をご覧ください。

図4-1. 応用開始とリセットの構成図



4.6. デバイス レジスタ

ブートローダによって管理されるデバイスレジスタ空間(コード番号5)はデバイスの64個の標準I/Oレジスタと160個の拡張I/Oレジスタです。これらは等価なアセンブリ言語命令によってアクセスされます。

LDS Rxx, REG_ADD ここでREG_ADDは次のアドレス範囲です。

- AT90CANxxx系列については\$20 (PINA)～\$FA(CANMSG)
- ATmegaxxM1/C1系列については\$23 (PINB)～\$FA(CANMSG)

4.6.1. 読み書き

“ISP読み込み”命令はこの空間に(ページなしで)バイト単位でアクセスします。アクセス保護はこの読み込み専用空間で全く提供されません。

4.6.2. 消去

読み込み専用空間のために当てはまりません。

4.6.3. 制限

この空間はビットアドレス指定ではなく、未実装のレジスタは\$FFが返ります。

4.6.4. デバイス レジスタ説明

情報については適切なデータシートを参照してください。

4.6.4.1. CANBT1～3レジスタ

CANBT1～3レジスタはアドレス\$E2～\$E4です。

これらは自動ビット速度を禁止する(EB≠\$FF)前に読んでブートローダ形態設定空間の“BTC1”, “BTC2”, “BTC3”内に再複写することが出来ます(9頁の「ビットタイミング制御1～3 - “BTC1～3”」をご覧ください)。その後、(EB≠\$FF中に)常にこのビットタイミングで始まるブートローダはIAPの場合に非常に有用です。

5. CAN規約とISP命令

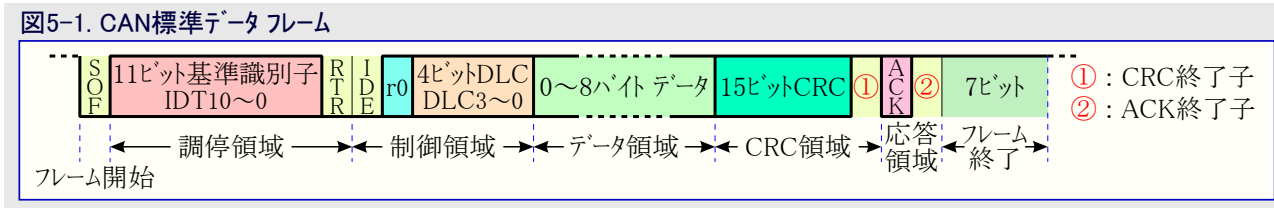
本章はCAN網上のより高位の規約と提携するISP命令の符号化を記述します。

5.1. CANフレーム説明

CAN規約は11ビット識別子を持つCAN 2.0Aとしても知られるCAN標準フレーム(高速用のISO11898と低速用のSIO11519-2参照)を支援します。

“フレーム開始(SOF)”で始まるCAN標準フレーム形式内のメッセージは、データフレームと遠隔フレームと呼ばれるデータ要求フレームとを区別するために使用される“遠隔送信要求(RTR)”ビットと識別子から成る“調停領域”が後続します。後続する“制御領域”は“識別子拡張(IDE)”ビットと、“データ領域”内の後続するデータバイト数を示すのに使用される“データ長符号(DLC)”を含みます。遠隔フレームではDLCが要求するデータバイト数を含みます。後続する“データ領域”は8つまでのデータバイトを保持することができます。フレームの完全性は後続する“巡回冗長検査(CRC)”和によって保証されます。“応答(ACK)領域”はACK間隔とACK終了子で折衷にします。ACK間隔内のビットは劣性ビットとして送られ、この時に正しく受信したデータを持つ受信側によって優性ビットとして上書きされます。

ISP CAN規約はCAN標準データフレームでだけ使用します。



ISP CAN規約を記述するために、識別子に抽象名が使用されますが、既定値は以下の表現内で与えられます。

表5-1. ISP CAN命令の雛形

識別子 (11ビット)	長さ (4ビット)	データ[0] ~ データ[n-1] (1バイト) ~ (1バイト)	説明
SYMBOLIC_NAME ("CRIS"<<4)+X	n (≤8)	値または意味	命令説明

点对点接続のため、送信CANメッセージは受信側によってハードウェア応用が行われるまで繰り返されます。ブートローダは形態設定が見つけられた場合にだけやって来るCANフレームに応答することができます。この機能は多数のCAN節点(ノード)を持つ網で保証されません。

5.2. CAN ISP命令データ列規約

5.2.1. CAN ISP命令説明

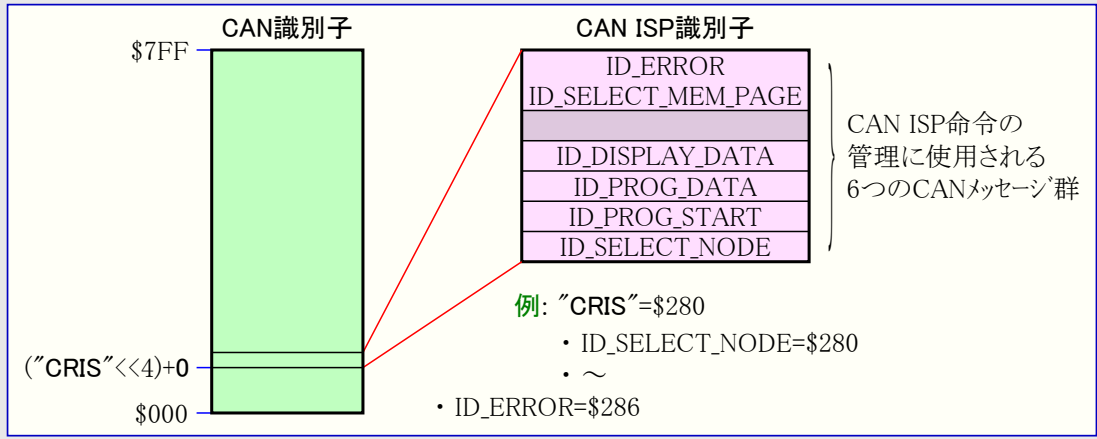
この規約を管理するために多数のCANメッセージ識別子が定義されています。

表5-2. CAN ISP規約用に定義されたCANメッセージ識別子

識別子	ISP命令詳細	値
ID_SELECT_NODE	節点(ノード)との通信路を開く/閉じる	("CRIS"<<4)+0
ID_PROG_START	メモリ空間のプログラミング開始	("CRIS"<<4)+1
ID_PROG_DATA	メモリ空間にデータ書き込み	("CRIS"<<4)+2
ID_DISPLAY_DATA	メモリ空間からデータ読み込み	("CRIS"<<4)+3
ID_START_APPLI	応用開始	("CRIS"<<4)+4
ID_SELECT_MEM_PAGE	メモリ空間またはページを選択	("CRIS"<<4)+6
ID_ERROR	ブートローダからの異常メッセージのみ	

識別子群に関する基礎値を持つ“CRIS”バイトを書くことによってCAN ISP識別子に新しい値を割り当てるのが可能です。最大“CRIS”値は\$7Fで既定値は\$00です。

図5-2. CAN ISP規約に対するCANメッセージ識別子の再配置



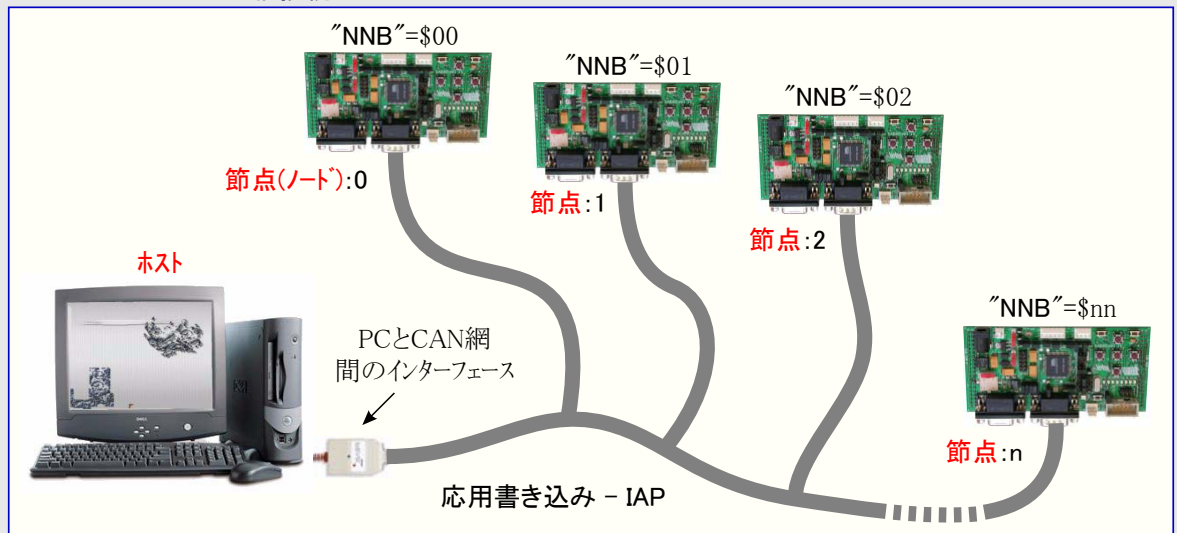
5.2.2. 通信初期化

どんなISP通信を始めるにも先立って装置(CAN節点(ノード))での通信が開かれなければなりません。装置で通信を開くには、ホストがパラメータとして渡される節点番号"NNB"を持つ"接続する"CANメッセージ("ID_SELECT_NODE")を送ります。渡された接点番号が\$FFの場合、CANポートローダが通信を受け入れます(図5-3)。さもなければパラメータで渡された接点番号は局所"NNB"と等しくなければなりません(図5-4)。

図5-3. CANポートローダ初回接続



図5-4. CANポートローダ網接続



別の装置で新しい通信を開く前に、現在の装置の通信がその接続CANメッセージ("ID_SELECT_NODE")で閉じられなければなりません。

5.3. CAN ISP命令

5.3.1. CAN節点選択

CAN節点(ノード)は作業の始めて開かれて終わりで閉じられなければなりません。

5.3.1.1. ホストからのCAN節点選択要求

表5-3. ホストからのCAN節点選択要求

識別子	長さ	データ[0]	説明
ID_SELECT_NODE ("CRIS"<<4)+0	1	節点番号("NNB")	指定節点との通信を開くまたは閉じる

5.3.1.2. フートローダからのCAN節点選択応答

表5-4. フートローダからのCAN節点選択応答

識別子	長さ	データ[0]	データ[1]	説明
ID_SELECT_NODE ("CRIS"<<4)+0	2	"フートローダ改訂"	\$00	通信を閉じました。
			\$01	通信を開きました。

5.3.2. メリ/ページ変更

メリ空間と/またはページの変更は1つの命令だけがあり、切り替えはCANフレームの"データ[0]"によって行われます。

5.3.2.1. ホストからのメリ/ページ変更要求

表5-5. ホストからのメリ/ページ変更要求

識別子	長さ	データ[0]	データ[1]	データ[2]	説明
ID_SELECT_MEM_PAGE ("CRIS"<<4)+6	3	\$00	メリ空間	ページ	活動なし
		\$01			メリ空間選択
		\$02			ページ選択
		\$03			メリ空間とページを選択

5.3.2.2. フートローダからのメリ/ページ変更応答

表5-6. フートローダからのメリ/ページ変更応答

識別子	長さ	データ[0]	説明
ID_SELECT_MEM_PAGE ("CRIS"<<4)+6	1	\$00	選択OK (例え要求フレームで"データ[0]"=\$00でも)

5.3.3. メリの読み込み/空白検査

これらの操作は通信に於いて直前に開いた装置でだけ実行されます。この命令は直前に定義されたメリ空間とページで利用可能です。

読み込みまたは空白検査を開始するには、ホストが"データ[0]"で望む操作、パラメータとして開始アドレスと終了アドレスを持つCANメッセージ("ID_DISPLAY_DATA")を送ります。

5.3.3.1. ホストからのメリの読み込み/空白検査要求

表5-7. ホストからのメリの読み込み/空白検査要求

識別子	長さ	データ[0]	データ[1]	データ[2]	データ[3]	データ[4]	説明
ID_DISPLAY_DATA ("CRIS"<<4)+3	5	\$00	開始アドレス (上位, 下位)		終了アドレス (上位, 下位)		選択したメリ/ページのデータ表示
		\$80					選択したメリ/ページの空白検査

5.3.3.2. フートローダからの読み込み/空白検査応答

表5-8. フートローダからのメモリの読み込み/空白検査応答

識別子	長さ	データ[0]	データ[1]	～	データ[7]	説明
ID_DISPLAY_DATA ("CRIS"<<4)+3	最大8	8バイトまでのデータバイト				データ読み込み
	0	-	-	-	-	空白検査OK
	2	非空白の最初のアドレス		-	-	空白検査で誤り
ID_ERROR ("CRIS"<<4)+6	1	\$00	-	-	-	ソフトウェア保護設定異常("データ表示"のみ)

5.3.4. メモリの書き込み/消去

これらの操作は通信に於いて直前に開いた装置でだけ実行されます。これらは次の2つの段階が必要です。

- ・最初の段階は書き込みまたは消去の命令に関するアドレス範囲を指示することです。
- ・2つ目の段階は書き込みだけに関してデータを送ることです。

書き込み操作を開始するには、ホストが"データ[0]"で望む操作、パラメータとして開始アドレスと終了アドレスを持つ"書き込み開始"CANメッセージ("ID_PROG_START")を送ります。

5.3.4.1. ホストからのメモリの書き込み/消去要求

表5-9. ホストからのメモリの書き込み/消去要求

識別子	長さ	データ[0]	データ[1]	データ[2]	データ[3]	データ[4]	データ[5~7]	説明
ID_PROG_START ("CRIS"<<4)+1	5	\$00	開始アドレス (上位, 下位)		終了アドレス (上位, 下位)		-	選択したメモリ空間/ページ 書き込み開始
	3	\$80	\$FF	\$FF	-	-	-	選択したメモリ空間/ページ消去
ID_PROG_DATA ("CRIS"<<4)+2	最大8	8バイトまでのデータバイト						書き込むデータ

5.3.4.2. フートローダからのメモリの書き込み/消去応答

表5-10. フートローダからのメモリの書き込み/消去応答

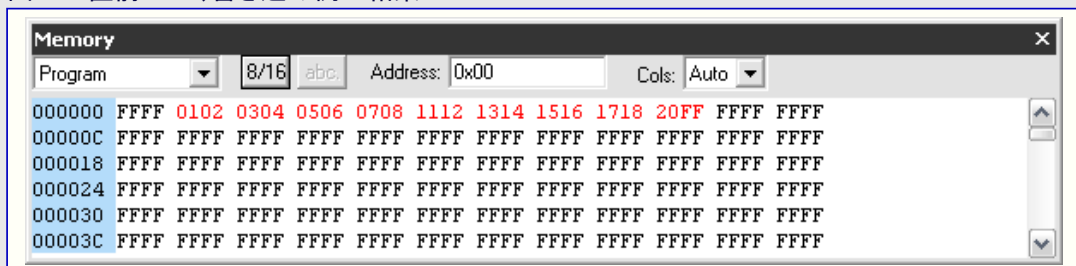
識別子	長さ	データ[0]	説明
ID_PROG_START ("CRIS"<<4)+1	0	-	命令OK
ID_PROG_DATA ("CRIS"<<4)+2	1	\$00	命令OK、転送終了
		\$02	命令OK、けれども新しい(他の)データを期待
ID_ERROR ("CRIS"<<4)+6	1	\$00	異常 - ソフトウェア保護設定 ("書き込み開始"のみ)

5.3.4.3. メモリ書き込み例

表5-11. メモリ書き込み例

要求/応用	CANメッセージ (16進)			説明
	識別子	長さ	データ[0~7]	
要求(→)	000	1	FF	CAN節点(ノード)選択
応答(←)	000	2	03 01	通信を開く
既定メモリ空間=フラッシュメモリ、既定ページ=0ページ				
要求(→)	001	5	00 00 02 00 12	\$0002~\$0012番地に書き込み
応答(←)	001	0	-	命令OK
要求(→)	002	8	01 02 03 04 05 06 07 08	第1データ列送信
応答(←)	002	1	02	命令OK、新しいデータを期待
要求(→)	002	8	11 12 13 14 15 16 17 18	第2データ列送信
応答(←)	002	1	02	命令OK、新しいデータを期待
要求(→)	002	1	20	第3データ列送信
応答(←)	002	1	00	命令OK、転送終了

図5-5. 直前のメモリ書き込み例の結果



注: AVR Studio®のプログラム用メモリ表示

5.3.5. 応用開始

この操作は通信で直前に開いた装置でだけ実行することができます。

5.3.5.1. ホストからの応用開始要求

応用を開始するには、ホストが“データ[1]”で選択する“方法”を持つ応用開始CANメッセージを送ります。応用はウォッチドッグリセットまたは定義された語アドレスに飛ぶことによって開始することができます。飛ぶ語アドレスはSA_H:SA_L(ブートローダ形態設定空間)から参照することができます。

表5-12. ホストからの応用開始要求

識別子	長さ	データ[0]	データ[1]	データ[2]	データ[3]	説明
ID_START_APPLI ("CRIS"<<4)+4	2		\$00	-	-	ウォッチドッグリセットで応用開始
	4	\$03	\$01	飛ぶ語アドレス (上位, 下位)		リセットなしで語アドレス(上位: 下位)から応用開始

5.3.5.2. ブートローダからの応用開始応答

ブートローダによって応答は全く返されません。

6. API - 応用プログラム インターフェース

6.1. API定義

応用プログラム インターフェース(API)はコンピュータプログラムによってそれを行うサービスに対する要求を支援するために、コンピュータシステムまたはプログラムが提供するソースコード インターフェースです。

6.2. API実装

ATMEL®のAVR 8ビット マイクロプロセッサの特異性はフラッシュメモリを書くことを提供するコードがブートローダ領域に配置されることを必要とすることです。デバイスに“減量”CANブートローダが書かれている場合、ブートローダ領域は既に占有され、使用者用に利用できません。フラッシュメモリでの書き込みを使用者に許す解決策はこの“減量”CANブートローダに含まれる“flash_wr_block()”ルーチンを開くことです。その後には使用者は自身の応用からそれを呼び出す(そしてよく吟味された部分コードを使用する)ことができます。

6.3. API - 使用の制限

CANブートローダはIARTMで開発、コンパイルされました。IARでコンパイルされた使用者プログラムだけが“減量”CANブートローダのAPIにアクセスすることができます。

6.4. API詳細

6.4.1. 関数名

flash_wr_block()

注: この関数は“flash_boot_lib.c”ファイルに配置されています。

6.4.2. 特徴

この関数はフラッシュメモリ内で65535バイト(64Kバイト-1バイト)までの書き込みを許します。この関数は整列の問題(バイトとフラッシュページの整列)を管理します。

注: 1. この関数は最大64Kバイトの供給元緩衝部をデバイスで得ることができないため、1度で65535バイト全部をアドレス指定することができません。

2. 64Kより大きなフラッシュメモリ量に関しては、先にページ設定が行われなければなりません(例: RAMPZレジスタの設定または解除)。既定の設定は0です。

6.4.3. 警告

異なったページで書かれるバイトはその"flash_wr_block()"関数でなければなりません。

6.4.4. 関数パラメータ

1. *src : unsigned charでの位置指示子 - (SRAM内の)供給元緩衝部
2. dest : unsigned short - 目的地、データが書かれなければならないフラッシュメモリ内の開始アドレス値
3. byte_nb : unsigned short - 書くバイト数

6.4.5. 関数戻り値

(なし)

6.5. 入口点

使用者応用と"減量"CANポートローダと一緒にコンパイルされないため、入口点が使われます。デバイス部品番号に拘らず、入口点は以下のように固定化されます。

API_ENTRY_POINT = (FLASH_SIZE - FLASH_BOOT_SIZE(注)) + 4バイト

注: FLASH_BOOT_SIZE=4Kバイト

下表は使用デバイスに対するAPI入口点アドレスを与えます。

表6-1. API入口点 対 デバイス部品番号

部品番号	API_ENTRY_POINT - 語アドレス	API_ENTRY_POINT - バイトアドレス
ATmega16M1	\$1802	\$3004
AT90CAN32, ATmega32M1, ATmega32C1	\$3802	\$7004
AT90CAN64, ATmega64M1, ATmega64C1	\$7802	\$F004
AT90CAN128	\$0F802	\$1F004

6.6. IARでのAPI呼び出し例

使用者プログラムは以下のコード(注)を用いてAPIを呼び出すことができます。(注: IAR Cコンパイラ用にだけ利用可能)

- ドライバ(*.c)ファイルでの - 関数

```
void (*flash_write) (unsigned char* src, unsigned short dest, unsigned short byte_nb) ¥
    = (void (*)(unsigned char*, unsigned short, unsigned short)) (API_ENTRY_POINT);
```

注: ここでENTRY_POINTはバイトアドレスです。

- ヘッド(*.h)ファイルでの - 関数原型

```
extern void (*flash_write) (unsigned char* src, unsigned short dest, unsigned short byte_nb);
```

6.7. 他のコンパイラでのAPI呼び出し例

6.7.1. IAR コンパイラと他のコンパイラまたはアセンブリ言語間の変数渡し

AVRに対してIAR Cコンパイラが使用される時にレジスタファイルは図6-1.で示されるように区分されます。

- 雑用レジスタは関数呼び出しを渡って保護されません。
- 局所レジスタは関数呼び出しを渡って保護されます。
- Yレジスタ(R29:R28)はSRAMへのデータスタックポインタとして使用されます。
- 関数間のパラメータ渡しと戻り値に雑用レジスタが使用されます。

関数が呼ばれる時に関数に渡されるべきパラメータはレジスタファイルのR16~R23に置かれます。関数が値を返す時に、パラメータと返される値の大きさに依存して、その値はレジスタファイルのR16~R23に置かれます。

表6-2.は関数呼び出し時のパラメータの配置例を示します。

表6-2. C関数へのパラメータと配置

関数	パラメータ1 レジスタ	パラメータ2 レジスタ
func(char,char)	R16	R20
func(char,short)	R16	R20,R21
func(short,long)	R16,R17	R20,R21,R22,R23
func(long,long)	R16,R17,18,19	R20,R21,R22,R23

支援されるデータ形式と対応する大きさの完全な参照についてはIAR SystemsからのAVR® IARコンパイラ手引きのデータ表現章をご覧ください。

図6-1. レジスタファイルの区分

雑用レジスタ	R30~R31
データスタックポインタ(Y)	R28~R29
局所レジスタ	R24~R27
雑用レジスタ	R16~R23
局所レジスタ	R4~R15
雑用レジスタ	R0~R3

6.7.2. "flash_wr_block()" APIで使われるレジスタ

関数パラメータ

- *src : R17:R16その後(⇒R9:R8、そしてZ(R31:R30))に転送されます。
- dest : R19:R18その後(⇒R25:R24⇒R27:R26(アドレス))
- nb_bytes : R21:R20その後(⇒R5:R4)

他の資源

- Y(R29:R28)は"データ スタック ポインタ"として使用されます。
- Z(R31:R30)は"LPM"と"SPM"で使用されます。
- "LPM"を使用するR0解除と(R0:R1)は"fill_temp_buffer"で使用されます。

使用レジスタの要約

R0, R1, R4, R5, R6, R7, R8, R9, R10, R11, R16, R17, R18, R19, R20, R21, R24, R25, R26, R27, R28, R29, R30, R31

"flash_boot_lib.lst"ファイル抜粋

```

138 //-----
139 #pragma location = "API_FLASH"
¥      In segment API_FLASH, align 2, keep-with-next
140 void flash_wr_block(U8* src, U16 dest, U16 byte_nb)
¥      flash_wr_block:
141 {
¥ 00000000 92BA          ST      -Y, R11
¥ 00000002 92AA          ST      -Y, R10
¥ 00000004 929A          ST      -Y, R9
¥ 00000006 928A          ST      -Y, R8
¥ 00000008 927A          ST      -Y, R7
¥ 0000000A 926A          ST      -Y, R6
¥ 0000000C 925A          ST      -Y, R5
¥ 0000000E 924A          ST      -Y, R4
¥ 00000010 93BA          ST      -Y, R27
¥ 00000012 93AA          ST      -Y, R26
¥ 00000014 939A          ST      -Y, R25
¥ 00000016 938A          ST      -Y, R24
¥ 00000018          REQUIRE ?Register_R4_is_cg_reg
¥ 00000018          REQUIRE ?Register_R5_is_cg_reg
¥ 00000018          REQUIRE ?Register_R6_is_cg_reg
¥ 00000018          REQUIRE ?Register_R7_is_cg_reg
¥ 00000018          REQUIRE ?Register_R8_is_cg_reg
¥ 00000018          REQUIRE ?Register_R9_is_cg_reg
¥ 00000018          REQUIRE ?Register_R10_is_cg_reg
¥ 00000018          REQUIRE ?Register_R11_is_cg_reg
¥ 00000018 0148          MOVW   R9:R8, R17:R16
¥ 0000001A 01C9          MOVW   R25:R24, R19:R18
¥ 0000001C 012A          MOVW   R5:R4, R21:R20
142 U8      save_i_flag;
143 U16     u16_temp, nb_word;
144 U16     address;
145 U16     save_page_addr;
146
147 //--- Special for API's -----
148 //-- First of all, disabling the Global Interrupt
149 save_i_flag = SREG;
¥ 0000001E B6AF          IN      R10, 0x3F
150 Disable_interrupt();
¥ 00000020 94F8          CLI

```

注: より多くの詳細については(生成されているなら(コンパイル任意選択)、)以下の*.lstファイルを参照してください。

- "flash_boot_lib.lst"ファイル
- "flash_boot_drv.lst"ファイル

7. 追補A : #define

7.1. プロセッサ定義

```
//----- ブートローダ定義 -----
#define     BOOT_LOADER_SIZE           0x1000    // バイトでの量:4KB
#define     MAX_FLASH_SIZE_TO_ERASE   ( FLASH_SIZE - ((U32) (BOOT_LOADER_SIZE)) )
//----- プロセッサ定義 -----
#define     XRAM_END                   XRAMEND    // "ioxxx.h"で定義
#define     RAM_END                     RAMEND    // "ioxxx.h"で定義
#define     E2_END                       E2END    // "ioxxx.h"で定義
#define     FLASH_END                   FLASHEND  // "ioxxx.h"でバイトで定義
#define     FLASH_SIZE                  ((U32) (FLASH_END)) + 1 // バイトでの量

// 特別な定義用の切り替え
#if defined(__AT90CAN128__)              // IARによって__HAS_ELPM__定義
#  define   MANUF_ID                     0x1E     // ATMEL(製造者)
#  define   FAMILY_CODE                  0x81     // AT90CANxxx系列
#  define   PRODUCT_NAME                  0x97     // 128Kバイトのフラッシュメモリ
#  define   PRODUCT_REV                   0x00    // 改訂0
#  define   FLASH_PAGE_SIZE              256     // バイトでの量
#  define   _BOOT_CONF_TYPE_             __farflash // 64K以上に割り当てられるブートローダ
#  define   _RAMPZ_IS_USED_               // 使用フラッシュメモリが64Kバイト以上の場合にRAMPZレジスタを使用

#elif defined(__AT90CAN64__)             // IARによって__HAS_ELPM__未定義
#  define   MANUF_ID                     0x1E     // ATMEL(製造者)
#  define   FAMILY_CODE                  0x81     // AT90CANxxx系列
#  define   PRODUCT_NAME                  0x96     // 64Kバイトのフラッシュメモリ
#  define   PRODUCT_REV                   0x00    // 改訂0
#  define   FLASH_PAGE_SIZE              256     // バイトでの量
#  define   _BOOT_CONF_TYPE_             __flash  // 64K以下に割り当てられるブートローダ

#elif defined(__AT90CAN32__)             // IARによって__HAS_ELPM__未定義
#  define   MANUF_ID                     0x1E     // ATMEL(製造者)
#  define   FAMILY_CODE                  0x81     // AT90CANxxx系列
#  define   PRODUCT_NAME                  0x95     // 32Kバイトのフラッシュメモリ
#  define   PRODUCT_REV                   0x00    // 改訂0
#  define   FLASH_PAGE_SIZE              256     // バイトでの量
#  define   _BOOT_CONF_TYPE_             __flash  // 64K以下に割り当てられるブートローダ

#elif defined(__ATmega16M1__)            // IARによって__HAS_ELPM__未定義
#  define   MANUF_ID                     0x1E     // ATMEL(製造者)
#  define   FAMILY_CODE                  0x84     // ATmegaxxM1/C1系列
#  define   PRODUCT_NAME                  0x94     // 16Kバイトのフラッシュメモリ
#  define   PRODUCT_REV                   0x00    // 改訂0
#  define   FLASH_PAGE_SIZE              128     // バイトでの量
#  define   _BOOT_CONF_TYPE_             __flash  // 64K以下に割り当てられるブートローダ

#elif defined(__ATmega32M1__)            // IARによって__HAS_ELPM__未定義
#  define   MANUF_ID                     0x1E     // ATMEL(製造者)
#  define   FAMILY_CODE                  0x84     // ATmegaxxM1/C1系列
#  define   PRODUCT_NAME                  0x95     // 32Kバイトのフラッシュメモリ
#  define   PRODUCT_REV                   0x00    // 改訂0
#  define   FLASH_PAGE_SIZE              128     // バイトでの量
#  define   _BOOT_CONF_TYPE_             __flash  // 64K以下に割り当てられるブートローダ

#elif defined(__ATmega32C1__)            // IARによって__HAS_ELPM__未定義
#  define   MANUF_ID                     0x1E     // ATMEL(製造者)
#  define   FAMILY_CODE                  0x86     // ATmegaxxM1/C1系列
#  define   PRODUCT_NAME                  0x95     // 32Kバイトのフラッシュメモリ
#  define   PRODUCT_REV                   0x00    // 改訂0
#  define   FLASH_PAGE_SIZE              128     // バイトでの量
#  define   _BOOT_CONF_TYPE_             __flash  // 64K以下に割り当てられるブートローダ
```

```

#elif defined(__ATmega32C1__) // IARによって__HAS_ELPM__未定義
#   define MANUF_ID          0x1E // ATMEL(製造者)
#   define FAMILY_CODE      0x86 // ATmegaxxM1/C1系列
#   define PRODUCT_NAME     0x95 // 32Kバイトのフラッシュメモリ
#   define PRODUCT_REV      0x00 // 改訂0
#   define FLASH_PAGE_SIZE  128 // バイトでの量
#   define __BOOT_CONF_TYPE_ __flash // 64K以下に割り当てられるブートローダ

#elif defined(__ATmega64M1__) // IARによって__HAS_ELPM__未定義
#   define MANUF_ID          0x1E // ATMEL(製造者)
#   define FAMILY_CODE      0x84 // ATmegaxxM1/C1系列
#   define PRODUCT_NAME     0x96 // 64Kバイトのフラッシュメモリ
#   define PRODUCT_REV      0x00 // 改訂0
#   define FLASH_PAGE_SIZE  256 // バイトでの量
#   define __BOOT_CONF_TYPE_ __flash // 64K以下に割り当てられるブートローダ

#elif defined(__ATmega64C1__) // IARによって__HAS_ELPM__未定義
#   define MANUF_ID          0x1E // ATMEL(製造者)
#   define FAMILY_CODE      0x86 // ATmegaxxM1/C1系列
#   define PRODUCT_NAME     0x96 // 64Kバイトのフラッシュメモリ
#   define PRODUCT_REV      0x00 // 改訂0
#   define FLASH_PAGE_SIZE  256 // バイトでの量
#   define __BOOT_CONF_TYPE_ __flash // 64K以下に割り当てられるブートローダ

#else
#   error IAR Embedded Workbench IDEでの不正なデバイス選択
#endif

#define FOSC          8000 // 8MHz外部クリスタル

```

7.2. CAN接続定義

```

//----- CAN定義 -----
#define CAN_BAUDRATE  CAN_AUTOBAUD

#if defined(__AT90CAN128__) || defined(__AT90CAN64__) || defined(__AT90CAN32__)
#   define CAN_PORT_IN    PIND
#   define CAN_PORT_DIR   DDRD
#   define CAN_PORT_OUT   PORTD
#   define CAN_INPUT_PIN  6
#   define CAN_OUTPUT_PIN 5
#   define NB_MOB         15
#elif defined(__ATmega16M1__) || defined(__ATmega32M1__) || defined(__ATmega32C1__) || ¥
    defined(__ATmega64M1__) || defined(__ATmega64C1__)
#   define CAN_PORT_IN    PINC
#   define CAN_PORT_DIR   DDRC
#   define CAN_PORT_OUT   PORTC
#   define CAN_INPUT_PIN  3
#   define CAN_OUTPUT_PIN 2
#   define NB_MOB         6
#else
#   error IAR Embedded Workbench IDEでの不正なデバイス選択
#endif

```

7.3. ブートローダ定義

```
//----- ブートローダ形態設定 -----
#define BOOT_LOADER_SIZE      0x1000    // バイトでの量:4KB
#define MAX_FLASH_SIZE_TO_ERASE  ( FLASH_SIZE - ((U32) (BOOT_LOADER_SIZE)) )

#define BOOT_VERSION          0x03
#define BOOT_ID1              0xD1
#define BOOT_ID2              0xD2

#define BSB_DEFAULT           0xFF
#define SSB_DEFAULT           0xFF
#define EB_DEFAULT            0xFF
#define BTC1_DEFAULT          0xFF
#define BTC2_DEFAULT          0xFF
#define BTC3_DEFAULT          0xFF
#define NNB_DEFAULT           0xFF
#define CRIS_DEFAULT          0x00
#define SA_H_DEFAULT          0x00
#define SA_L_DEFAULT          0x00

#define BSB ((U16) &boot_conf[0])
#define SSB ((U16) &boot_conf[1])
#define EB  ((U16) &boot_conf[2])
#define BTC1 ((U16) &boot_conf[3])
#define BTC2 ((U16) &boot_conf[4])
#define BTC3 ((U16) &boot_conf[5])
#define NNB ((U16) &boot_conf[6])
#define CRIS ((U16) &boot_conf[7])
#define SA_H ((U16) &boot_conf[8])
#define SA_L ((U16) &boot_conf[9])

#define BOOT_CONF_SIZE        10
#define SSB_INDEX              0x01
#define SSB_NO_SECURITY        0xFF
#define SSB_WR_PROTECTION      0xFE
#define SSB_RD_WR_PROTECTION  0xFC
```

7.4. メモリ定義

```
//----- メモリ定義 -----
#define MEM_FLASH              0
#define MEM_EEPROM             1
#define MEM_SIGNATURE          2
#define MEM_BOOT_INF           3        // ブートローダ情報
#define MEM_BOOT_CONF          4        // ブートローダ形態設定
#define MEM_HW_REG              5
#define MEM_DEF_MAX             MEM_HW_REG
#define MEM_DEFAULT             MEM_FLASH

#define PAGE_DEFAULT            0x00
#define ADD_DEFAULT             0x0000
#define N_DEFAULT               0x0001
```

7.5. CAN規約定義

```
//----- IAPデータ -----
#define MAX_BASE_ISP_IAP_ID      0x7F0
#define MIN_BASE_ISP_IAP_ID      0x000

//----- 規約命令 -----
#define CAN_ID_SELECT_NODE       0x00

#define CAN_ID_PROG_START         0x01
# define CAN_INIT_PROG           0x00
# define CAN_FULL_ERASE_1        0x80
# define CAN_FULL_ERASE_2        0xFF
# define CAN_FULL_ERASE_3        0xFF

#define CAN_ID_PROG_DATA          0x02

#define CAN_ID_DISPLAY_DATA       0x03
# define CAN_READ_DATA           0x00
# define CAN_BLANK_CHECK         0x80

#define CAN_ID_START_APPLI        0x04
# define CAN_START_APPLI         0x03
# define CAN_RESET_APPLI         0x00
# define CAN_JUMP_APPLI          0x01

#define CAN_ID_SELECT_MEM_PAGE    0x06
# define CAN_NO_ACTION           0x00
# define CAN_SEL_MEM             0x01
# define CAN_SEL_PAGE            0x02
# define CAN_SEL_MEM_N_PAGE      0x03

#define CAN_ID_ERROR              0x06

#define COMMAND_OK                0x00
#define OK_END_OF_DATA            0x00
#define OK_NEW_DATA               0x02

#define LOCAL_BUFFER_SIZE         0x100
```

8. 追補B : CAN規約要約

表8-1. CAN規約要約 - ホストからの要求

ISP命令要求識別子	長さ	データ								説明	
		[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]		
ID_SELECT_NODE ("CRIS"<<4)+0	1	節点	-	-	-	-	-	-	-	通信を開くまたは閉じる	
ID_PROG_START ("CRIS"<<4)+1	5	\$00	開始アドレス (上位, 下位)		終了アドレス (上位, 下位)		-	-	-	書き込みの初期化	
	3	\$80	\$FF	\$FF	-	-	-	-	消去		
ID_PROG_DATA ("CRIS"<<4)+2	最大8	8バイトまでのデータ バイト								書き込むデータ	
ID_DISPLAY_DATA ("CRIS"<<4)+3	5	\$00	開始アドレス (上位, 下位)		終了アドレス (上位, 下位)		-	-	-	データ表示(読み込み)	
		\$80	空白検査								
ID_START_APPLI ("CRIS"<<4)+4	2	\$03	\$00	-	-	-	-	-	-	リセットで応用開始	
	4	\$03	\$01	\$0000		-	-	-	-	応用開始, 0番地へ飛ぶ	
ID_SELECT_MEM_PAGE ("CRIS"<<4)+6	3	\$00	活動なし								
		\$01	メモリ 空間	ページ							メモリ空間選択
		\$02									ページ選択
		\$03									メモリ空間とページを選択

表8-2. CAN規約要約 - ブートローダからの応答

ISP命令応答識別子	長さ	データ								説明	
		[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]		
ID_SELECT_NODE ("CRIS"<<4)+0	2	ブート ローダ 改訂	\$00								通信を閉じる
			\$01								通信を開く
ID_PROG_START ("CRIS"<<4)+1	0	-	-	-	-	-	-	-	-	命令OK	
ID_PROG_DATA ("CRIS"<<4)+2	1	\$00	命令OK, 転送の最後								
		\$02	命令OK, 新しいデータを期待								
ID_DISPLAY_DATA ("CRIS"<<4)+3	最大8	8バイトまでのデータ バイト								データ読み込み	
	0	-	-	-	-	-	-	-	-	リセットで応用開始	
	2	非空白の 最初のアドレス		-	-	-	-	-	-	空白検査での異常	
ID_SELECT_MEM_PAGE またはID_ERROR ("CRIS"<<4)+6	1	\$00	-	-	-	-	-	-	-	選択OKまたはソフトウェア保護設定異常	

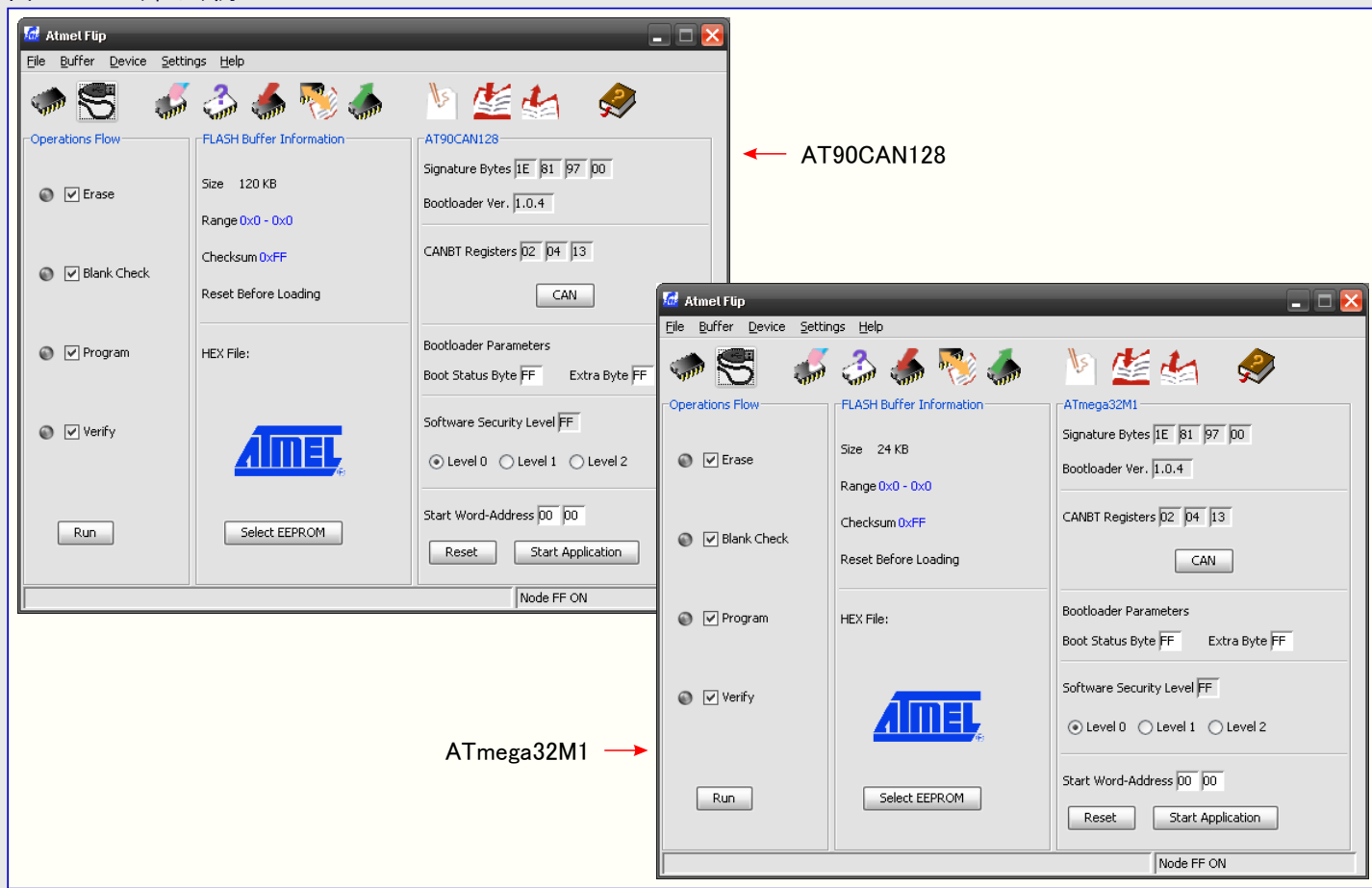
9. 追補C : FLIPインターフェース

FLIPはATMELのマイクロ コントローラ デバイスを実装書き込みする柔軟なPC応用です。このFLIPの主新版は以下の能力を提供します。

- RS232、USB、CANのインターフェースを通しての実装書き込みの実行
- その直感的な画像ユーザーインターフェースまたは、DOS®窓(batchisp手引書をご覧ください)やKEIL's uVision2のような組み込みソフトウェアIDE、更にあなた自身の応用(ISP関数ライブラリ手引書をご覧ください)からの起動を通して使用できます。
- Windows® 9x/Me/NT®/2000/XP®下で走行
- データ ファイルの読み込みと保存にIntel® MCS-86 16進オブジェクト、コード88ファイル形式を支援
- 緩衝部編集能力：塗り、検索、複写、書き込み、照合、読み込み、保護レベルと特殊バイトの読み出しと設定
- デバイス逐次化能力(batchispからのみ)
- ISPハードウェア条件がソフトウェアによって設定可能
- どんな目的対象ハードウェアもなしに実演動作形態がISP操作を擬態

FLIPリンク：http://www.atmel.com/dyn/resources/prod_documents/Flip_Installer_3_3_4.exe

図9-1. FLIPウィンドウ例



10. 追補D：目的対象デバイスに対するコンパイルの注意

参照：・ AVR用IAR C/C++コンパイラ 5.20.1 (5.20.50092)
 ・ ".¥\$PROJ_DIR\$¥config.h"ファイルでの基盤選択

10.1. 目的対象選択

Project⇒Options⇒General Options⇒Target⇒Processor Configuration (以下の何れか)

- ・ --cpu=can128, AT90CAN128
- ・ --cpu=can64, AT90CAN64
- ・ --cpu=can32, AT90CAN32
- ・ --cpu=32m1, ATmega32M1
- ・ --cpu=32c1, ATmega32C1

(以下は2009年4月13日現在、未実装)

- ・ --cpu=64m1, ATmega64M1
- ・ --cpu=64c1, ATmega64C1
- ・ --cpu=16m1, ATmega16M1

10.2. 最適化

Project⇒Options⇒C/C++ Compiler⇒Optimizations

- ・ Speed : High(最大最適化)
- ・ Number of cross-call passes : Unlimited
- ・ Always do cross call optimization : ON

10.3. 対応するリカ命令ファイルの選択

Project⇒Options⇒Linker⇒Config⇒Linker Command File (以下の何れか)

- Override defaultをチェック
- ..¥\$PROJ_DIR\$¥can128_iar_can_bootloader_link.xcl
- ..¥\$PROJ_DIR\$¥can64_iar_can_bootloader_link.xcl
- ..¥\$PROJ_DIR\$¥can32_iar_can_bootloader_link.xcl
- ..¥\$PROJ_DIR\$¥m64m1c1_iar_can_bootloader_link.xcl
- ..¥\$PROJ_DIR\$¥m32m1c1_iar_can_bootloader_link.xcl
- ..¥\$PROJ_DIR\$¥m16m1c1_iar_can_bootloader_link.xcl

10.4. コンパイル

1. Project⇒Clean
2. Project⇒Rebuilt All

生成物 : "IAR_can_boot_loader.a90" ("*.hex"と等価)と"IAR_can_boot_loader.dbg"

位置 : ..¥\$PROJ_DIR\$¥output_iar¥debug¥exe¥

10.5. 予めコンパイルされたHEXファイル

(使用される基盤に依存して)いくつかの予めコンパイルされたHEXファイルが以下に作られています。

..¥\$PROJ_DIR\$¥output_iar¥debug¥exe¥pre_compiled_hex_file¥

- AT90CAN128用 : IAR_can_boot_loader_dvk90can1_at90can128.hexまたはIAR_can_boot_loader_stk600_at90can128.hex
- AT90CAN64用 : IAR_can_boot_loader_stk600_at90can64.hex
- AT90CAN32用 : IAR_can_boot_loader_stk600_at90can32.hex
- ATmega32M1/C1用 : IAR_can_boot_loader_stk600_atmega32m1c1.hexまたはIAR_can_boot_loader_mc310_atmega32m1c1.hex



本社

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131
USA
TEL 1(408) 441-0311
FAX 1(408) 487-2600

国外営業拠点

Atmel Asia

Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
TEL (852) 2245-6100
FAX (852) 2722-1369

Atmel Europe

Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
TEL (33) 1-30-60-70-00
FAX (33) 1-30-60-71-11

Atmel Japan

104-0033 東京都中央区
新川1-24-8
東熱新川ビル 9F
アトメル ジャパン株式会社
TEL (81) 03-3523-3551
FAX (81) 03-3523-7581

製品窓口

ウェブサイト

www.atmel.com

技術支援

avr@atmel.com

販売窓口

www.atmel.com/contacts

文献請求

www.atmel.com/literature

お断り: 本資料内の情報はATMEL製品と関連して提供されています。本資料またはATMEL製品の販売と関連して承諾される何れの知的所有権も禁反言あるいはその逆によって明示的または暗示的に承諾されるものではありません。ATMELのウェブサイトに位置する販売の条件とATMELの定義での詳しい説明を除いて、商品性、特定目的に関する適合性、または適法性の暗黙保証に制限せず、ATMELはそれらを含むその製品に関連する暗示的、明示的または法令による如何なる保証も否認し、何ら責任がないと認識します。たとえATMELがそのような損害賠償の可能性を進言されたとしても、本資料を使用できない、または使用以外で発生する(情報の損失、事業中断、または利益の損失に関する制限なしの損害賠償を含み)直接、間接、必然、偶然、特別、または付随して起こる如何なる損害賠償に対しても決してATMELに責任がないでしょう。ATMELは本資料の内容の正確さまたは完全性に関して断言または保証を行わず、予告なしでいつでも製品内容と仕様の変更を行う権利を保留します。ATMELはここに含まれた情報を更新することに対してどんな公約も行いません。特に別の方法で提供されなければ、ATMEL製品は車載応用に対して適当ではなく、使用されるべきではありません。ATMEL製品は延命または生命維持を意図した応用での部品としての使用に対して意図、認定、または保証されません。

© Atmel Corporation 2009. 全権利予約済 ATMEL®、ロゴとそれらの組み合わせ、AVR®とその他はATMEL Corporationの登録商標または商標またはその付属物です。他の用語と製品名は一般的に他の商標です。

© HERO 2013.

本応用記述はATMELのAVR076応用記述(doc8247.pdf Rev.8247A-08/09)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意識されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には()内に英語表記や略称などを残す形で表記しています。

青字の部分はリンクとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。