Armadillo 標準ガイド ハードウェア拡張編

～組み込み Linux の導入から製品化まで～

Version 1.0.0
2019/04/09

株式会社アットマークテクノ [http://www.atmark-techno.com]

Armadillo サイト [http://armadillo.atmark-techno.com]

Armadillo 標準ガイド ハードウェア拡張編: 〜組み込み Linux の導入から製品化まで〜

株式会社アットマークテクノ

Version 1.0.0
2019/04/09

目次

1. はじめに 9
1.1. 対象読者 9
1.2. 表記について 9
1.2.1. コマンド入力例 9
1.2.2. アイコン 9
1.3. サンプルソースコード 10
1.4. 困った時は 10
1.5. お問い合わせ先 11
1.6. 商標 11
1.7. ライセンス 11
1.8. 謝辞 11
2. 注意事項 12
3. ハードウェア機能をカスタマイズする 13
3.1. Device Tree のカスタマイズ 13
3.1.1. Device Tree とは 13
3.1.2. Device Tree の記述 14
3.2. カーネルイメージのカスタマイズ 19
3.3. DTB・イメージをビルドする 21
4. UART(シリアルインターフェース)の活用 22
4.1. UART とは 22
4.2. コンソールに使用する UART を変更する 23
4.2.1. ブートローダーのコンソールに使用する UART を変更する 23
4.2.2. Linux のコンソールに使用する UART を変更する 24
5. I2C デバイスの活用 26
5.1. I2C とは 26
5.2. A/D コンバーター(PCF8591)を使用する 28
5.2.1. 接続方法 28
5.2.2. 対応カーネルイメージの作成 29
5.2.3. 使用例 30
5.2.4. プロトコル 31
5.3. I/O エクスパンダー(PCF8574)を使用する 32
5.3.1. 接続方法 33
5.3.2. 対応カーネルイメージの作成 33
5.3.3. 使用例 35
5.3.4. プロトコル 36
5.4. 温度湿度センサー(HDC1080)を使用する 36
5.4.1. 接続方法 37
5.4.2. 対応カーネルイメージの作成 37
5.4.3. 使用例 39
5.4.4. プロトコル 39
5.5. VOCs センサー(CCS811)を使用する 41
5.5.1. 接続方法 41
5.5.2. 対応カーネルイメージの作成 41
5.5.3. 使用例 43
5.5.4. プロトコル 43
6. SPI デバイスの活用 45
6.1. SPI とは 45
6.2. A/D コンバーター(MCP3204)を使用する 46
6.2.1. 接続方法 47
6.2.2. 対応カーネルイメージの作成 47

Armadillo 標準ガイド ハードウェア拡張編

6.2.3. 使用例 49
6.2.4. プロトコル 49
7. 1-Wire デバイスの活用 51
7.1. 1-Wire とは 51
7.2. 温度センサー(DS18B20)を使用する 53
7.2.1. 接続方法 53
7.2.2. 対応カーネルイメージの作成 54
7.2.3. 使用例 55
7.2.4. プロトコル 56
8. CAN の活用 58
8.1. CAN とは 58
8.2. CAN を使用する 61
8.2.1. 接続方法 61
8.2.2. 対応カーネルイメージの作成 62
8.2.3. CAN 通信プログラムの準備 63
8.2.4. 使用例 64
A. Linux カーネルサポートデバイス情報 66
A.1. 加速度センサー 66
A.2. A/D コンバーター 66
A.3. 化学センサー 69
A.4. D/A コンバーター 69
A.5. 周波数シンセサイザ 70
A.6. ジャイロスコープ 70
A.7. ヘルスセンサー 70
A.8. 湿度センサー 70
A.9. 慣性測定センサー 71
A.10. 照度センサー 71
A.11. 磁力計センサー 72
A.12. デジタルポテンショメーター 72
A.13. ポテンショスタット 73
A.14. 気圧センサー 73
A.15. 距離センサー 74
A.16. 温度センサー 74
A.17. ファン速度コントローラ 75
A.18. 電源モニタ 76

図目次

1.1. クリエイティブコモンズライセンス 11
3.1. armadillo-640-i2c4.dtsi (I2C 使用ピン設定部分) 14
3.2. armadillo-640_multiplex 抜粋 14
3.3. MUX の内部構成 15
3.4. PAD の内部構成 16
3.5. PAD 設定値のフォーマット 16
3.6. armadillo-640-i2c4.dtsi (I2C バスノード部分) 18
3.7. armadillo-640-i2c4.dtsi (I2C スレーブデバイスノード部分) 18
3.8. armadillo-640-i2c4.dtsi (全体) 18
3.9. armadillo-640.dts 19
4.1. UART の接続方法 22
4.2. ハードウェアフロー制御の接続方法 22
4.3. RS232C への変換 23
4.4. UART プロトコル(データビット数 8、ストップビット数 1、パリティビットなし) 23
4.5. デフォルトコンフィギュレーション適用 24
4.6. menuconfig の実行 24
4.7. menuconfig 24
4.8. ブートローダーをビルド 24
4.9. ブートローダーイメージ書き換え 24
4.10. 再起動 24
4.11. ブートパラメータの設定 25
5.1. I2C の接続方法 26
5.2. 複数のスレーブを接続する場合 26
5.3. I2C の通信フォーマット 27
5.4. 複数バイトの連続送受信 27
5.5. アドレスバイト 28
5.6. I2C の波形 28
5.7. I2C 接続 A/D コンバーター回路図 29
5.8. armadillo-640-i2c4.dtsi 29
5.9. armadillo-640.dts 30
5.10. menuconfig の実行 30
5.11. menuconfig 30
5.12. コマンド実行例 31
5.13. PCF8591 通信フォーマット(書き込み) 31
5.14. PCF8591 通信フォーマット(D/A 出力値書き込み) 31
5.15. PCF8591 通信フォーマット(読み出し) 31
5.16. PCF8591 アドレスバイト 32
5.17. PCF8591 コントロールバイト 32
5.18. PCF8591 データバイト 32
5.19. I2C 接続 I/O エクスパンダー回路図 33
5.20. armadillo-640-i2c4.dtsi 34
5.21. armadillo-640.dts 34
5.22. menuconfig の実行 34
5.23. menuconfig 34
5.24. GPIO クラスディレクトリ作成コマンド実行例 35
5.25. 出力コマンド実行例 35
5.26. 入力コマンド実行例 35
5.27. PCF8574 通信フォーマット(書き込み) 36
5.28. PCF8574 通信フォーマット(読み出し) 36
5.29. PCF8574 アドレスバイト 36

5.30. PCF8574 データバイト 36
5.31. I2C 接続温度湿度センサー回路図 37
5.32. armadillo-640-i2c4.dtsi 38
5.33. armadillo-640.dts 38
5.34. menuconfig の実行 38
5.35. menuconfig 38
5.36. コマンド実行例 39
5.37. 湿度の計算 39
5.38. 温度の計算 39
5.39. HDC1080 アドレスバイト 40
5.40. HDC1080 通信フォーマット(読み出し) 40
5.41. HDC1080 通信フォーマット(書き込み) 40
5.42. HDC1080 コンフィグレーションレジスタ 40
5.43. I2C 接続 VOCs センサー回路図 41
5.44. armadillo-640-i2c4.dtsi 42
5.45. armadillo-640.dts 42
5.46. menuconfig の実行 42
5.47. menuconfig 43
5.48. コマンド実行例 43
5.49. CCS811 アドレスバイト 43
5.50. CCS811 通信フォーマット(STATUS 読み出し) 44
5.51. CCS811 通信フォーマット(ALG_RESULT_DATA 読み出し) 44
5.52. CCS811 STATUS レジスタ 44
6.1. SPI の接続方法 45
6.2. 複数のスレーブを接続する場合 45
6.3. SPI モードと波形 46
6.4. SPI 接続 A/D コンバーター回路図 47
6.5. armadillo-640-ecspi1.dtsi 48
6.6. armadillo-640.dts 48
6.7. menuconfig の実行 49
6.8. menuconfig 49
6.9. コマンド実行例 49
6.10. MCP3204 通信フォーマット 50
7.1. 1-Wire の接続方法 51
7.2. 複数のスレーブを接続する場合 51
7.3. 1-Wire プロトコル(書き込みスロット) 52
7.4. 1-Wire プロトコル(読み出しスロット) 52
7.5. 1-Wire プロトコル 53
7.6. 1-Wire 接続温度センサー回路図 54
7.7. armadillo-640-w1.dtsi 54
7.8. armadillo-640.dts 55
7.9. menuconfig の実行 55
7.10. menuconfig 55
7.11. コマンド実行例 55
7.12. コマンド実行例(Hardware Monitoring) 56
7.13. DS18B20 Temperature Register フォーマット 57
8.1. CAN の接続方法 58
8.2. 3 つ以上のノードの接続方法 58
8.3. CAN プロトコル(データフレーム) 59
8.4. CAN プロトコル(リモートフレーム) 60
8.5. CAN トランシーバー回路図 62
8.6. CAN 接続図 62
8.7. armadillo-640-can2.dtsi 62

8.8. armadillo-640.dts に include を追記 63
8.9. menuconfig の実行 63
8.10. menuconfig 63
8.11. can-utils のインストール 64
8.12. CAN 信速度設定 64
8.13. CAN を有効化 64
8.14. candump 実行開始 64
8.15. cansend でのメッセージ送信 64
8.16. candump でのメッセージ受信結果 64
8.17. cangen での連続メッセージ送信 65
8.18. candump での連続メッセージ受信結果 65

Armadillo 標準ガイド ハードウェア拡張編

表目次

1.1. 表示プロンプトと実行環境の関係 9
1.2. コマンド入力例での省略表記 9
3.1. PAD 設定値の詳細 17
5.1. I²C モード一覧 26
5.2. PCF8591 のデータバイト 31
5.3. PCF8574 のデータバイト 36
5.4. HDC1080 のレジスタ 40
5.5. CCS811 のレジスタ 43
6.1. SPI モード 46
6.2. MCP3204 チャンネル指定 50
7.1. DS18B20 内蔵レジスタ 56
7.2. DS18B20 温度センサー分解能 56
8.1. CAN プロトコルフレーム 59
A.1. Linux カーネルがサポートする加速度センサー 66
A.2. Linux カーネルがサポートする A/D コンバーター 66
A.3. Linux カーネルがサポートする化学センサー 69
A.4. Linux カーネルがサポートする D/A コンバーター 70
A.5. Linux カーネルがサポートする周波数シンセサイザ 70
A.6. Linux カーネルがサポートするジャイロスコープ 70
A.7. Linux カーネルがサポートするヘルスセンサー 70
A.8. Linux カーネルがサポートする湿度センサー 71
A.9. Linux カーネルがサポートする慣性測定センサー 71
A.10. Linux カーネルがサポートする照度センサー 71
A.11. Linux カーネルがサポートする磁力計センサー 72
A.12. Linux カーネルがサポートするデジタルポテンショメーター 72
A.13. Linux カーネルがサポートするポテンショスタット 73
A.14. Linux カーネルがサポートする気圧センサー 73
A.15. Linux カーネルがサポートする距離センサー 74
A.16. Linux カーネルがサポートする温度センサー 74
A.17. Linux カーネルがサポートするファン速度コントローラ 76
A.18. Linux カーネルがサポートする電源モニタ 76

1. はじめに

「Armadillo 入門編」では、Armadillo を使った組み込みシステムを構築する方法の全体像について説明しました。本書「ハードウェア拡張編」では、デバイス追加などの具体的な事例を取り上げ、Howto 形式で紹介します。

1.1. 対象読者

本書が主な対象読者としているのは、Armadillo を使って組み込みシステムを開発したいと考えているソフトウェア開発者です。ソフトウェア開発者は、少なくとも C 言語での開発経験が必要です。Linux や Armadillo を使用した開発の経験が少ない場合や開発の全体像を把握していない場合は、「Armadillo 入門編」から読むことをお勧めします。

1.2. 表記について

1.2.1. コマンド入力例

本書に記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。「 / 」の部分はカレントディレクトリによって異なります。各ユーザのホームディレクトリは「 ~ 」で表します。

表 1.1 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[PC /]#	作業用 PC(Linux)の root ユーザで実行
[PC /]$	作業用 PC(Linux)の一般ユーザで実行
[ATDE /]#	ATDE 上の root ユーザで実行
[ATDE /]$	ATDE 上の一般ユーザで実行
[armadillo /]#	Armadillo 上 Linux の root ユーザで実行
[armadillo /]$	Armadillo 上 Linux の一般ユーザで実行
⇒	Armadillo 上 U-Boot の保守モードで実行

コマンド中で、変更の可能性のあるものや、環境により異なるものに関しては以下のように表記します。適宜読み替えて入力してください。

表 1.2 コマンド入力例での省略表記

表記	説明
[version]	ファイルのバージョン番号

1.2.2. アイコン

本書では以下のようにアイコンを使用しています。

注意事項を記載します。

役に立つ情報を記載します。

用語の説明や補足的な説明を記載します。

1.3. サンプルソースコード

本書で紹介するサンプルソースコードは、 https://download.atmark-techno.com/armadillo-guide-std/sample/ からダウンロードできます。サンプルソースコードは、MIT ライセンス[1]の下に公開します。

1.4. 困った時は

本書を読んでわからなかったり困ったことがあった際は、ぜひ Armadillo サイト[2]で情報を探してみてください。本書には記載しきれていない FAQ や Howto が掲載されています。

Armadillo サイトでも知りたい情報が見つからない場合は、「Armadillo フォーラム」[3]で質問してみてください。Armadillo フォーラムは、アットマークテクノユーザーズサイト内に設けられた、Armadillo ブランド製品での開発や周辺技術に関する話題を扱うユーザー向けコミュニティです。Armadillo に関する技術的な話題なら何でも投稿できます。多くのユーザーや開発者が参加しているので、知識のある人や同じ問題で困ったことがある人から情報を集めることができます。

フォーラムに参加するときの心構え

Armadillo フォーラムには、その前身となったメーリングリストから引き続き、数百人のユーザーが参加しています。また、フォーラムへ投稿した内容は Web 上で誰でも閲覧・検索可能になるほか、通知を希望しているユーザーにメールで送信されます。

フォーラムには多くの人が参加しており、投稿内容は多くの人の目に触れますので、そこにはマナーが存在します。一般的な対人関係と同様に、受け取り手に対して失礼にならないよう一定の配慮はすべきです。技術系コミュニティに不慣れな方は、投稿する前に「技術系メーリングリストで質問するときのパターン・ランゲージ」[4] をご一読されることをお勧めします。メーリングリストに投稿するときの心構えや、適切な回答を得るために有用なテクニックが分かりやすく紹介されています。メーリングリストとフォーラムの違いはあれど、基本的な考え方は共通しており、とても参考になります。

[1]http://opensource.org/licenses/mit-license.php
[2]https://armadillo.atmark-techno.com
[3]https://users.atmark-techno.com/forum/armadillo
[4]結城浩氏によるサイトより http://www.hyuki.com/writing/techask.html

> とはいえ、技術的に簡単なものであるとか、ちょっとした疑問だからという理由で、投稿をためらう必要はありません。Armadillo に関係のある内容であれば、難しく考えることなく気軽にお使いください。

1.5. お問い合わせ先

本書に関するご意見やご質問は、Armadillo フォーラム[3]にご連絡ください。

1.6. 商標

Armadillo は、株式会社アットマークテクノの登録商標です。その他の記載の商品名および会社名は、各社・各団体の商標または登録商標です。™、®マークは省略しています。

1.7. ライセンス

図 1.1 クリエイティブコモンズライセンス

1.8. 謝辞

Armadillo で使用しているソフトウェアの多くは Free Software / Open Source Software で構成されています。Free Software / Open Source Software は世界中の多くの開発者の成果によってなりたっています。この場を借りて感謝の意を表します。

2. 注意事項

注意: 本書の内容を実践する前に

ご使用になる製品のマニュアル(ハードウェアマニュアル、ソフトウェアマニュアル、その他関連資料)をよく読み、それらに記述されている注意事項に従って正しく安全にお使いください。

3. ハードウェア機能をカスタマイズする

本章では、Armadillo-640 を例として、ハードウェア機能をカスタマイズするために必要な DeviceTree の記述方法と、カーネルへのデバイスドライバの有効化方法について述べます。

Linux カーネルに含まれているデバイスドライバの一覧は付録 A Linux カーネルサポートデバイス情報をご覧ください。

次章からは、いくつかのデバイスを例に、具体的なデバイスの追加方法を示します。

使用するソフトウェアのバージョンは以下のとおりです。

- ブートローダー: U-Boot 2018.03-at4 以降
- Linux カーネル: linux-4.14-at9 以降
- ユーザーランド: Debian GNU/Linux 9（stretch）v20181128 以降

3.1. Device Tree のカスタマイズ

3.1.1. Device Tree とは

Device Tree とは、ハードウェア情報を記述したデータ構造体です。ハードウェアの差分を Device Tree に記述することによって、1 つの Linux カーネルイメージを複数のハードウェアで利用することができるようになります。

Device Tree に対応しているメリットの 1 つは、ハードウェアの変更に対するソフトウェアの変更が容易になることです。例えば、CON9(拡張インターフェース)に接続する拡張基板を作成した場合、主に C 言語で記述された Linux カーネルのソースコードを変更する必要はなく、やりたいことをより直感的に記述できる DTS(Device Tree Source)の変更で対応できます。

ただし、Device Tree は「データ構造体」であるため、ハードウェアの制御方法などの「処理」を記述することができない点に注意してください。Device Tree には、CPU アーキテクチャ、RAM の容量、各種デバイスのベースアドレスや割り込み番号などのハードウェアの構成情報のみが記述されます。

Device Tree のより詳細な情報については、Linux カーネルのソースコードに含まれているドキュメント(`Documentation/devicetree/`)、devicetree.org で公開されている「Device Tree Specification」を参照してください。

> **DeviceTree: The Devicetree Specification**
>
> https://www.devicetree.org/

Linux カーネルのソースコードに含まれている初期出荷状態での DTS、及び関連するファイルを次に示します。

初期出荷状態での DTS、及び関連するファイル	・ `arch/arm/boot/dts/armadillo-640.dts` ・ `arch/arm/boot/dts/imx6ull.dtsi`

- arch/arm/boot/dts/imx6ul.dtsi
- armadillo-640-default-console.dtsi
- armadillo-640-uart5.dtsi
- armadillo-640-lcd70ext-l00.dtsi

3.1.2. Device Tree の記述

ここでは I2C を使用するための設定を例に説明します。新規に作成するファイルは".armadillo-640-i2c4.dtsi"とし、次の 3 つのノードを作成します。

- iomuxc ノード
- I2C バスノード
- I2C スレーブデバイスノード

1. iomuxc ノードの記述

   I2C4 の SCL を例に、iomuxc の記述方法を説明します。

```
&iomuxc {
        pinctrl_i2c4: i2c4grp {                  // 他の iomuxc ノードと重複しない名前
                fsl,pins= <
                //      MX6UL_PAD_AAAAAAAAAAAAA__BBBBBBBB 0xCCCCCCCC  // 説明のため
                        MX6UL_PAD_UART2_TX_DATA__I2C4_SCL 0x40010808  // SCL ピンの設定
                        MX6UL_PAD_UART2_RX_DATA__I2C4_SDA 0x40010808  // SDA ピンの設定
                >;
        };
};
```

図 3.1 armadillo-640-i2c4.dtsi (I2C 使用ピン設定部分)

AAA と BBB の部分はピンのマルチプレクサ(接続先を切り替える機能)の設定です。

a. 「マルチプレクス表」を参照し、I2C4 の SCL で使用できるピンを調べます。

Armadillo-640 マルチプレクス表

https://users.atmark-techno.com/files/downloads/armadillo-640/document/armadillo-640_multiplex-v1.0.0.zip

部品番号	ピン番号	信号名	ピン名	マルチプレクス機能(i.MX6ULLの信号名で表記)				
				I2C			CAN	
				I2C2	I2C3	I2C4	CAN1	CAN2
CON14	1	VCC_3.3V						
	2	GND						
	3	GPIO1_IO20	UART2_TX_DATA			I2C4_SCL		
	4	GPIO1_IO21	UART2_RX_DATA			I2C4_SDA		
CON11	15	LCD_DATA02	LCD_DATA02			I2C4_SDA		
	16	LCD_DATA03	LCD_DATA03			I2C4_SCL		

図 3.2 armadillo-640_multiplex 抜粋

列「I2C4」を見ると、CON14-3 と CON11-16 が使用できることがわかります。今回は CON14-3 を使用することにします。CON14-3 の行の「ピン名」を見ると UART2_TX_DATA と書かれています。これが AAA の部分に入ります。_(アンダースコア)を二つ挟み、列「マルチプレクス機能-I2C4」の I2C4_SCL が BBB の部分に入ります。AAA と BBB の記述により、下図のように入出力の MUX が選択され、太線部分が接続されます。

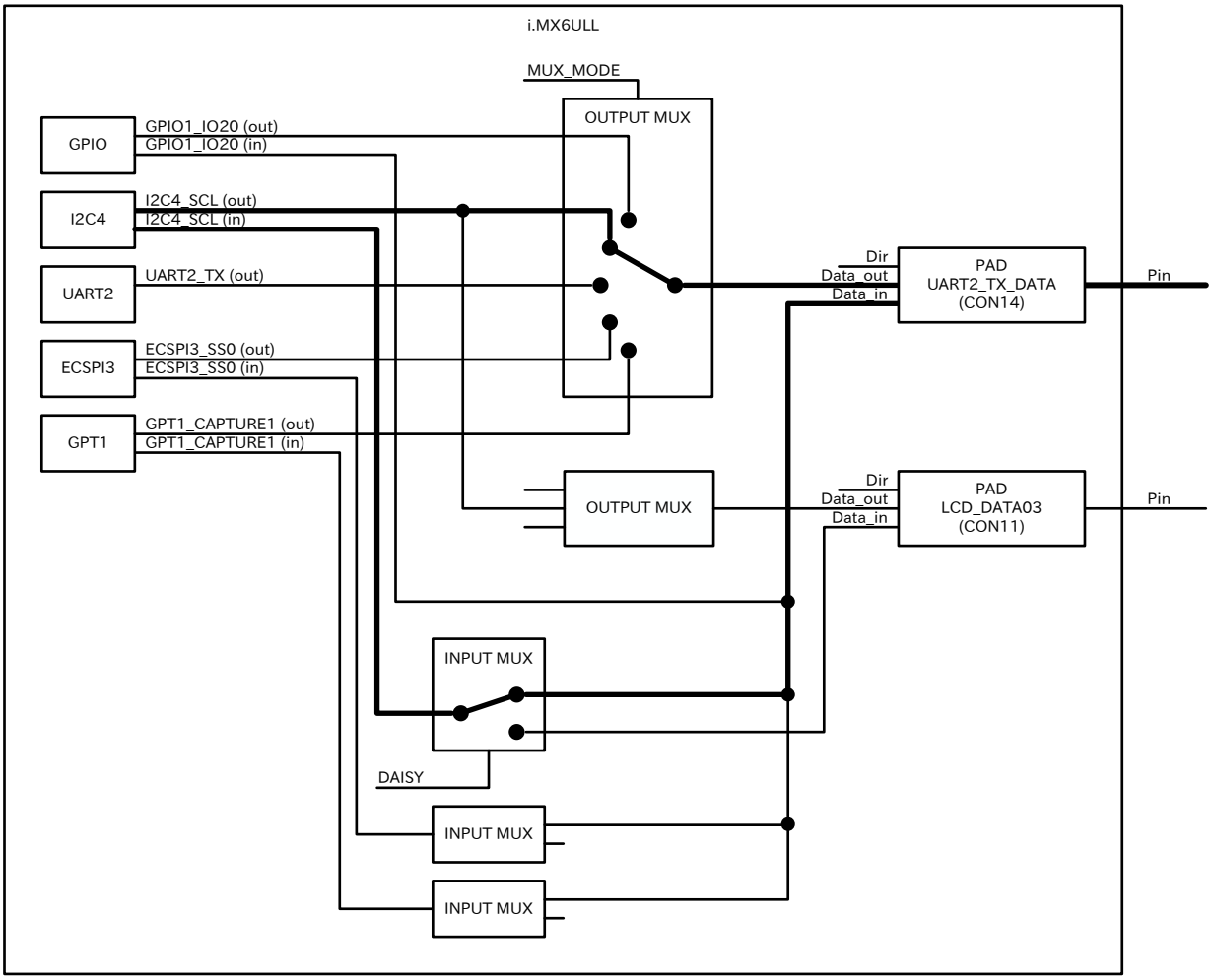

図 3.3 MUX の内部構成

CCC の部分は PAD(各ピンの入出力特性等の設定)の設定値です。PAD の内部は下図のようになっています。

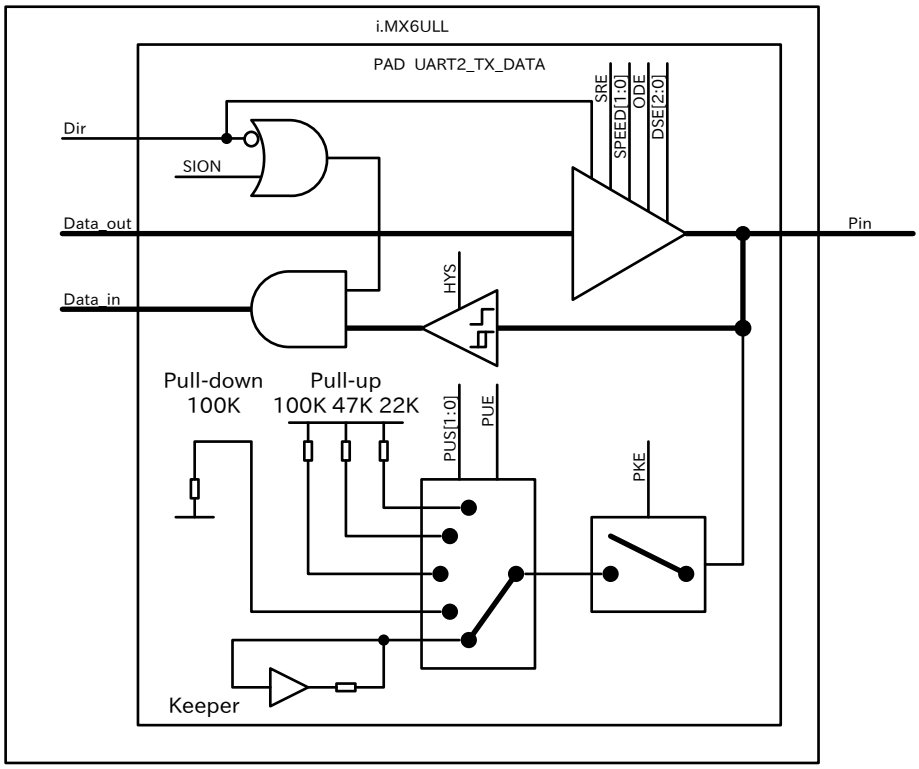

図 3.4 PAD の内部構成

出力バッファーの設定は、信号の周波数等にあわせて調整できます。入力バッファーのヒステリシス(シュミットトリガー)や、内部プルアップ等の設定も可能です。

I²C の場合は下記の点を考慮します。

・ 最高 400kHz 程度と低速 → ノイズ低減のため、低速な設定を選択

・ ピンは双方向で使用 → 入力バッファーを強制的に有効化

・ オープンドレイン出力 → オープンドレインを選択

・ 遅い信号の立ち上がり → ヒステリシス有りを選択

32 ビットで表される値のフォーマットは「図 3.5. PAD 設定値のフォーマット」のようになっており、各ビットの動作は「表 3.1. PAD 設定値の詳細」のようになります。

31							24	23							16
NO_PAD_CTL	SION	-	-	-	-	-	-	-	-	-	-	-	-	-	HYS

15							8	7							0
PUS [1]	PUS [0]	PUE	PKE	ODE	-	-	-	SPEED [1]	SPEED [0]	DSE [2]	DSE [1]	DSE [0]	-	-	SRE

図 3.5 PAD 設定値のフォーマット

よって、下表で * が付いている設定にします。

表 3.1 PAD 設定値の詳細

設定	値	動作
NO_PAD_CTL PAD 設定の要否	0 *	PAD 設定を行う
	1	PAD 設定を行わない
SION 入力バッファーを強制的に有効化 ・ I2C や 1-Wire 等の双方向ピンでは要有効化	0	出力ピンの場合は入力バッファー無効
	1 *	強制的に有効
HYS 入力バッファーのヒステリシス設定 ・ 有りにすると耐ノイズ性向上	0	ヒステリシス無し
	1 *	ヒステリシス有り
PUS プルダウン / プルアップ選択	00 *	プルダウン 100KΩ
	01	プルアップ 47KΩ
	10	プルアップ 100KΩ
	11	プルアップ 22KΩ
PUE キーパー / プル選択	0 *	キーパー
	1	プル
PKE キーパー / プル設定	0 *	キーパー/プル無効
	1	キーパー/プル有効
ODE 出力バッファーのオープンドレイン選択	0	プッシュプル
	1 *	オープンドレイン
SPEED 出力電流設定 ・ 高周波設定にすると出力電流を増加 ・ 低周波設定にするとスイッチングノイズを低減	00 *	最高出力周波数 50MHz
	01	最高出力周波数 100MHz
	10	最高出力周波数 100MHz
	11	最高出力周波数 200MHz
DSE ドライブストレングス調整 ・ 出力インピーダンスを上げるとオーバーシュート小 ・ 出力インピーダンスを下げると立ち上がり・立ち下がりが高速	000	出力バッファーがオフ
	001 *	出力インピーダンス最大
	010	
	011	
	100	
	101	
	110	
	111	出力インピーダンス最小
SRE スルーレート設定 ・ 低速にするとノイズ低減	0 *	低速
	1	高速

SDA も同様に UART2_TX_DATA が AAA に、 I2C4_SDA が BBB に入り、CCC は同じ値を設定します。

詳しくは以下の資料をご覧ください。

> i.MX 6ULL Applications Processor Reference Manual(リファレンスマニュアル)
>
> https://www.nxp.com/webapp/Download?colCode=IMX6ULLRM※登録が必要です。

カーネルのソースコードにも関連するドキュメントが付属しています。

- Documentation/devicetree/bindings/pinctrl/fsl,imx-pinctrl.txt
- Documentation/devicetree/bindings/pinctrl/fsl,imx6ul-pinctrl.txt

2. I2C バスノードの記述

次章以降や、 arch/arm/boot/dts/ にある .dtsi .dts ファイルが参考になります。

関連するドキュメントは Documentation/devicetree/bindings/ 以下にあります。I²C の場合は、次に示すファイルとなります。

- Documentation/devicetree/bindings/i2c/i2c.txt
- Documentation/devicetree/bindings/i2c/i2c-imx.txt

100kHz の設定で記述した場合、次のようになります。このノードの中に、スレーブデバイスのノードも書きます。

```
&i2c4 {
        status = "okay";                // okay で上書きする
        clock-frequency = <100000>;     // クロック周波数(100kHz)
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c4>;    // 先ほど作成した iomuxc ノード

        // ここにスレーブデバイスのノードを記述
};
```

図 3.6 armadillo-640-i2c4.dtsi (I²C バスノード部分)

3. I²C スレーブデバイスノードの記述

ここでは PCF8591 を例に説明します。記述方法はデバイスによって異なります。

```
pcf8591@48 {                            // スレーブのデバイス名@デバイスアドレス
        #address-cells = <1>;
        #size-cells = <0>;
        compatible = "nxp,pcf8591";     // 文字列が一致するデバイスドライバが使用される
        reg = <0x48>;                   // スレーブのデバイスアドレス

        input_mode = <0>;               // デバイス固有のプロパティー(あれば)
};
```

図 3.7 armadillo-640-i2c4.dtsi (I²C スレーブデバイスノード部分)

3 つのノードを合わせると以下のようになります。これを"arch/arm/boot/dts/armadillo-640-i2c4.dtsi"として保存します(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_i2c4: i2c4grp {                 // 他の iomuxc ノードと重複しない名前
                fsl,pins= <
                        MX6UL_PAD_UART2_TX_DATA__I2C4_SCL 0x40010808  // SCL ピンの設定
                        MX6UL_PAD_UART2_RX_DATA__I2C4_SDA 0x40010808  // SDA ピンの設定
                >;
        };
};

&i2c4 {
        status = "okay";                // okay で上書きする
        clock-frequency = <100000>;     // クロック周波数(100kHz)
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c4>;    // 先ほど作成した iomuxc ノード
```

```
        pcf8591@48 {                            // スレーブのデバイス名@デバイスアドレス
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "nxp,pcf8591";     // 文字列が一致するデバイスドライバが使用される
                reg = <0x48>;                   // スレーブのデバイスアドレス

                input_mode = <0>;               // デバイス固有のプロパティー(あれば)
        };
};
```

図 3.8 armadillo-640-i2c4.dtsi (全体)

さきほどのファイルへの include を"arch/arm/boot/dts/armadillo-640.dts"に追加します。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-i2c4.dtsi"              //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 3.9 armadillo-640.dts

このように、新たにデバイス等を追加する場合は、dtsi ファイルを追加する形式にすると管理しやすいでしょう。

3.2. カーネルイメージのカスタマイズ

menuconfig を使用してカーネルコンフィギュレーションを変更します。なお、すでにカスタマイズしたソースコードを元に行う場合、手順 5 から行います。

1. Linux カーネルのソースコードアーカイブを準備カレントディレクトリにソースコードアーカイブがあることを確認します。

```
[ATDE ~]$ ls
initramfs_a600-[version].cpio.gz linux-v4.14-at[version].tar.gz
```

2. Linux カーネルのソースコードアーカイブを展開

```
[ATDE ~]$ tar xf linux-v4.14-at[version].tar.gz
[ATDE ~]$ ls
initramfs_a600-[version].cpio.gz linux-v4.14-at[version]  linux-v4.14-at[version].tar.gz
```

3. initramfs アーカイブへのシンボリックリンク作成

```
[ATDE ~]$ cd linux-v4.14-at[version]
[ATDE ~/linux-v4.14-at[version]]$ \
> ln -s ../initramfs_a600-[version].cpio.gz initramfs_a600.cpio.gz
```

4. コンフィギュレーションの初期化

```
[ATDE ~/linux-v4.14-at[version]]$ make ARCH=arm armadillo-640_defconfig
```

5. menuconfig の実行

```
[ATDE ~/linux-v4.14-at[version]]$ make ARCH=arm menuconfig
```

6. カーネルコンフィギュレーションを変更

変更後、"Exit"を選択して"Do you wish to save your new configuration? (Press <ESC><ESC> to continue kernel configuration.)"で"Yes"を選択し、カーネルコンフィギュレーションを確定します。

```
.config - Linux/arm 4.14-at11 Kernel Configuration
--------------------------------------------------------------------------------
 ---------------- Linux/arm 4.14-at11 Kernel Configuration -----------------
    Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty
    submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
    includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to
    exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
   ------------------------------------------------------------------------
            General setup  --->
        [ ] Enable loadable module support  ----
        [*] Enable the block layer  --->
            System Type  --->
            Bus support  --->
            Kernel Features  --->
            Boot options  --->
            CPU Power Management  --->
            Floating point emulation  --->
            Userspace binary formats  --->
            Power management options  --->
        [*] Networking support  --->
            Device Drivers  --->
            Firmware Drivers  --->
            File systems  --->
            Kernel hacking  --->
            Security options  --->
        -*- Cryptographic API  --->
            Library routines  --->
        [ ] Virtualization  ----
   ------------------------------------------------------------------------
 --------------------------------------------------------------------------------
          <Select>    < Exit >    < Help >    < Save >    < Load >
 --------------------------------------------------------------------------------
```

Linux Kernel Configuration メニューで"/"キーを押下すると、カーネルコンフィギュレーションの検索を行うことができます。カーネルコンフィギュレーションのシンボル名(の一部)を入力して"Ok"を選択すると、部分

一致するシンボル名を持つカーネルコンフィギュレーションの情報が一覧されます。

3.3. DTB・イメージをビルドする

1. ビルド

```
[ATDE ~/linux-v4.14-at[version]]$ \
> make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0x82000000 uImage
[ATDE ~/linux-v4.14-at[version]]$ \
> make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
```

2. イメージファイルの生成確認

   ビルドが終了すると、arch/arm/boot/ディレクトリと、arch/arm/boot/dts/以下にイメージファイル(Linux カーネルと DTB)が作成されています。

```
[ATDE ~/linux-v4.14-at[version]]$ ls arch/arm/boot/uImage
arch/arm/boot/uImage
[ATDE ~/linux-v4.14-at[version]]$ ls arch/arm/boot/dts/armadillo-640.dtb
arch/arm/boot/dts/armadillo-640.dtb
```

4. UART(シリアルインターフェース)の活用

4.1. UART とは

UART は「Universal Asynchronous Receiver/Transmitter」の略で、主に機器間の通信に用いられる通信方式です。[1]

一般的な接続方法を下図に示します。なお、Armadillo-640 では、CON3/4 の RS232C、CON9/CON11 の UART バスを使用して PC や別のデバイスと通信できます。

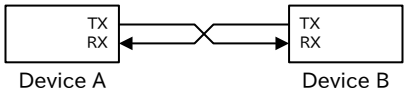

図 4.1 UART の接続方法

信号線は TX(送信)を相手の RX(受信)に接続します。

UART の主な特徴を以下に示します。

- 各デバイスは対等
- 1 対 1 の通信
- 信号線は送受信の 2 本
- 9600bps と 115200bps が標準(最高 1Mbps 程度)

信頼性の高い通信のため、通信相手の受信可否に合わせて送信できる構成にすることがあります。これをフロー制御といい、ハードウェアフロー制御では主に RTS/CTS を使用してフロー制御を行います。

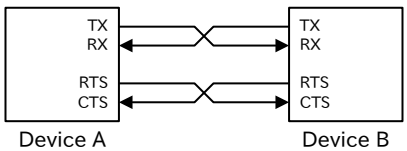

図 4.2 ハードウェアフロー制御の接続方法

簡単な回路で RS-232C に変換し PC 等と通信することもできます。

[1]正確には「UART」は調歩同期方式通信を行うための回路を指しますが、その通信方式を指している場合が多いです。

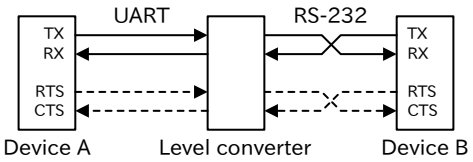

図 4.3 RS232C への変換

UART ではボーレート[2]等に様々なバリエーションがあり、主に以下の設定を通信相手と合わせる必要があります。よく選択される設定を強調で示します。

- ボーレート : 4800, **9600**, 19200, 38400, **115200** 等
- データビット数 : 7, **8**, 9
- ストップビット数 : **1**, 1.5, 2
- パリティビット : **なし**, Odd, Even
- フロー制御 : **なし**, ハードウェア, ソフトウェア

UART の波形は以下のようになっています。I^{2}C や SPI と異なり、LSB から送信されることに注意してください。ハードウェアフロー制御を行う場合、送信側は CTS が H のときに送信を開始しません。

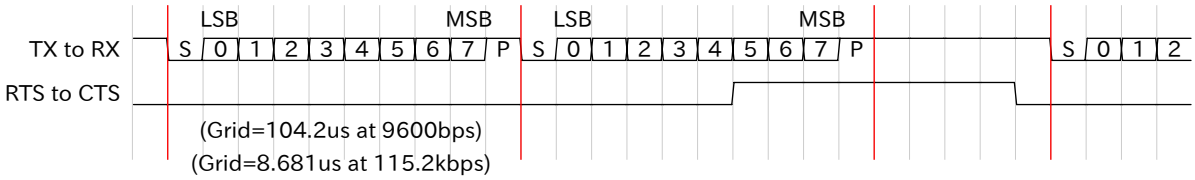

図 4.4 UART プロトコル(データビット数 8、ストップビット数 1、パリティビットなし)

4.2. コンソールに使用する UART を変更する

Armadillo-640 は、標準状態で UART1(CON9)をコンソールとして使用します。コンソールには、起動ログやカーネルメッセージなどが出力されるため、標準の設定では UART1 で使用しているピンに外部機器を接続して使用するといったことはできません。ここでは、コンソールとして別のシリアルインターフェースを使用する方法について説明します。例として、コンソールを UART3(CON3 もしくは CON4)に変更します。

シリアルインターフェースとデバイスファイルの対応は、「Armadillo-640 製品マニュアル」の「Linux カーネル仕様」章「UART」項を参照してください。

「Armadillo 入門編」の「起動の仕組み」でも説明したように、コンソールに文字を表示するプログラムには、ブートローダー(u-boot)、Linux カーネル、ユーザーランドアプリケーションプログラムの 3 種類があります。

4.2.1. ブートローダーのコンソールに使用する UART を変更する

ブートローダーで UART3 をコンソールとして使うためにブートローダーコンフィギュレーションを変更してビルドを行います。

[2]デジタルデータを変調・復調する速さ

まず、Armadillo-640 のデフォルトコンフィギュレーションを適用します。以下のコマンドを実行してください。

```
[ATDE ~]$ cd u-boot-a600-v2018.03-at[version]
[ATDE ~/u-boot-a600-v2018.03-at[version]]$ make ARCH=arm armadillo-640_defconfig
```

図 4.5 デフォルトコンフィギュレーション適用

次に、コンソールとして UART3 を利用するようブートローダーのコンフィギュレーションを変更します。

以下のコマンドを実行し、「ARM architecture」→「Console UART select」で「UART3」を選択してください。

```
$ make ARCH=arm menuconfig
```

図 4.6 menuconfig の実行

```
ARM architecture ->
  Console UART select (UART1) --->
  () UART1
  (X) UART3 <-を選択
```

図 4.7 menuconfig

コンフィギュレーションの変更後、ブートローダーをビルドしてください。

```
make CROSS_COMPILE=arm-linux-gnueabihf-
```

図 4.8 ブートローダーをビルド

```
[armadillo ~]# ls u-boot.imx
u-boot.imx
[armadillo ~]# dd if=u-boot.imx of=/dev/mmcblk0 bs=1k seek=1 conv=fsync
```

図 4.9 ブートローダーイメージ書き換え

```
[armadillo ~]# reboot
```

図 4.10 再起動

4.2.2. Linux のコンソールに使用する UART を変更する

u-boot-a600-v2018.03-at3 以降では、環境変数 optargs に設定されたパラメータがブートパラメータとして Linux カーネルに渡されます。

Linux で UART3 をコンソールとして使用するには、optargs に console=ttymxc2,115200 を設定します。

```
=> editenv optargs
edit: console=ttymxc2,115200
=> printenv optargs
optargs=console=ttymxc2,115200
=> saveenv
Saving Environment to MMC... Writing to MMC(0)... OK  ←OK が出る
```

図 4.11 ブートパラメータの設定

5. I2C デバイスの活用

5.1. I2C とは

I2C は「Inter Integrated Circuit」の略で、機器内、機器間のどちらの通信にも用いられる通信方式です[1]。機器内のセンサー等のインターフェースとして最も普及しています。また、類似の規格として、DDC(Display Data Channel)や SMBus があり、I2C と一部互換性があります。

I2C には以下のモード[2]があります。本書では、普及している「スタンダードモード」と「ファストモード」の説明を行います。

表 5.1 I2C モード一覧

モード	クロック	Armadillo-640 対応	スタンダードモードとの互換性
スタンダードモード	～100kHz	○	○
ファストモード	～400kHz	○	○
ファストモードプラス	～1MHz	×	○
ハイスピードモード	～3.4MHz	×	一部互換
ウルトラファストモード	～5MHz	×	互換性なし

一般的な接続方法を下図に示します。なお、Armadillo-640 はマスターとして別のデバイスと通信できます。

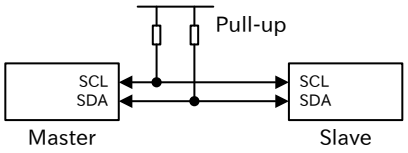

図 5.1 I2C の接続方法

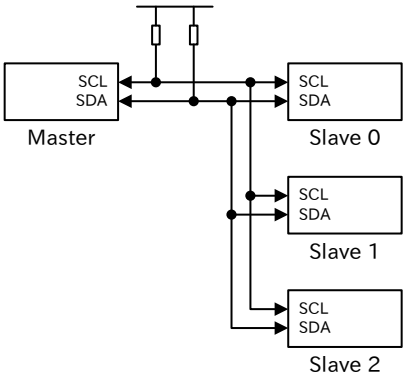

図 5.2 複数のスレーブを接続する場合

I2C の主な特徴を以下に示します。

- デバイスはマスターとスレーブが存在
- 1 つのバス上にマスターとスレーブを複数接続可能[3]
- 信号線は SDA(データ)と SCL(クロック)の 2 本のみ
- 信号線はオープンドレイン(プルアップが必要)
- データはクロックに合わせて変化

[1] I2C や IIC と表記される場合もあります。アイ・スクエア(ド)・シーやアイ・ツー・シーと読みます。
[2] I2C の「モード」は、デバイスが対応している通信速度を表します。
[3] マスターは 1 つのみにする場合が多いです。

・スレーブは 7 ビットもしくは 10 ビットのアドレスを持つ[4]

・通信は常にマスターが開始し、アドレスでスレーブを選択

・クロックは最高 400kHz(ファストモード)、最高 100kHz(スタンダードモード)

I²C の詳しい仕様は、NXP Semiconductors から公開されている資料をご覧ください。

I²C バス仕様およびユーザーマニュアル

https://www.nxp.com/docs/ja/user-guide/UM10204.pdf

I²C のデータフォーマットは以下のようになっています。

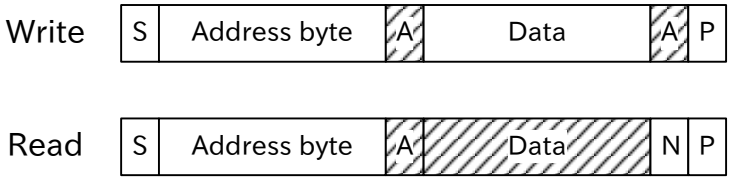

図 5.3 I²C の通信フォーマット

S	スタートコンディション
P	ストップコンディション
A	ACK(ACK=0)
N	NACK(ACK=1)

通信はスタートコンディションで開始し、ストップコンディションで終わります。斜線部分がスレーブの応答です。1 バイトごとに受信側が ACK を返すようになっています。なお、マスターが最後のデータを受信したときは NACK を返します。

最初のバイトはアドレスバイトといいます。2 バイト目以降はデータバイトとなり、内容やフォーマットはスレーブのデバイスごとに異なります。

デバイスによっては、以下のように複数バイトの読み出しを連続で行うことも可能です。

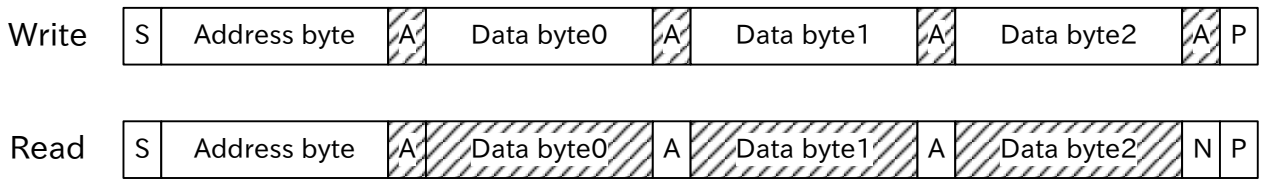

図 5.4 複数バイトの連続送受信

アドレスバイトは、スレーブ指定のための 7 ビットのアドレスとマスターが読み出すか書き込むかをスレーブに伝えるための R/W で構成されています。

[4]アドレスはデバイスの型番により異なります。アドレスを設定ピンによって変更し、複数の同じデバイスで別のアドレスを持つことができるデバイスもあります。

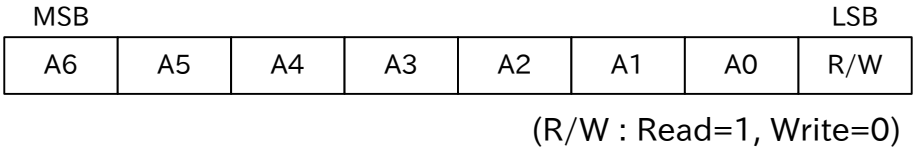

図 5.5 アドレスバイト

なお、リスタートやクロックストレッチについては複雑なため、ここでは触れていません。

I²C の波形は以下のようになっています。

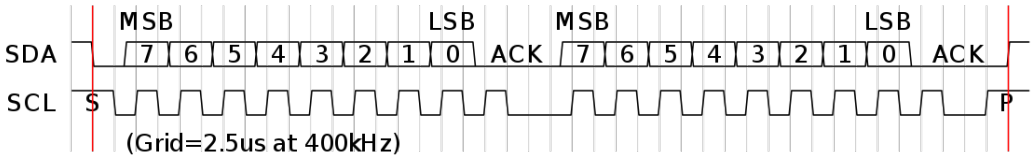

図 5.6 I²C の波形

5.2. A/D コンバーター(PCF8591)を使用する

ここでは、I²C バスに A/D コンバーターを接続する方法を紹介します。

使用するデバイスは以下のとおりです。

- PCF8591(NXP Semiconductors 製)

今回使用する PCF8591 は、以下の特長を持ちます。

- 単電源動作(2.5〜6V)
- I²C 接続(スタンダードモード)
- アドレス 0x48〜0x4F(同一バスに 8 つ接続可能)
- 分解能 8 ビット
- 逐次比較型
- 4 入力
- D/A 出力あり(8 ビット 1 出力)

5.2.1. 接続方法

Armadillo-640 との接続を示します。Armadillo-640 の CON14 から出ている I2C4 に PCF8591 を接続します。アドレスを指定する A0〜A2 は全て GND に接続しておきます[5]。AIN0〜AIN3 がアナログ入力ピンです。AIN0 にかかる電圧を、10kΩ の可変抵抗で変えられるようにしています。AIN1〜AIN3 はそれぞれ固定電圧としています。リファレンス電圧 VREF に電源電圧と同じ 3.3V を入力しているため、0V〜3.3V の範囲のアナログ入力を 8 ビット(256 段階)のデジタル値に変換します。

[5]複数のスレーブをバスに接続する場合は、A0〜A2 の設定を変えてアドレスが重複しないようにしてください。

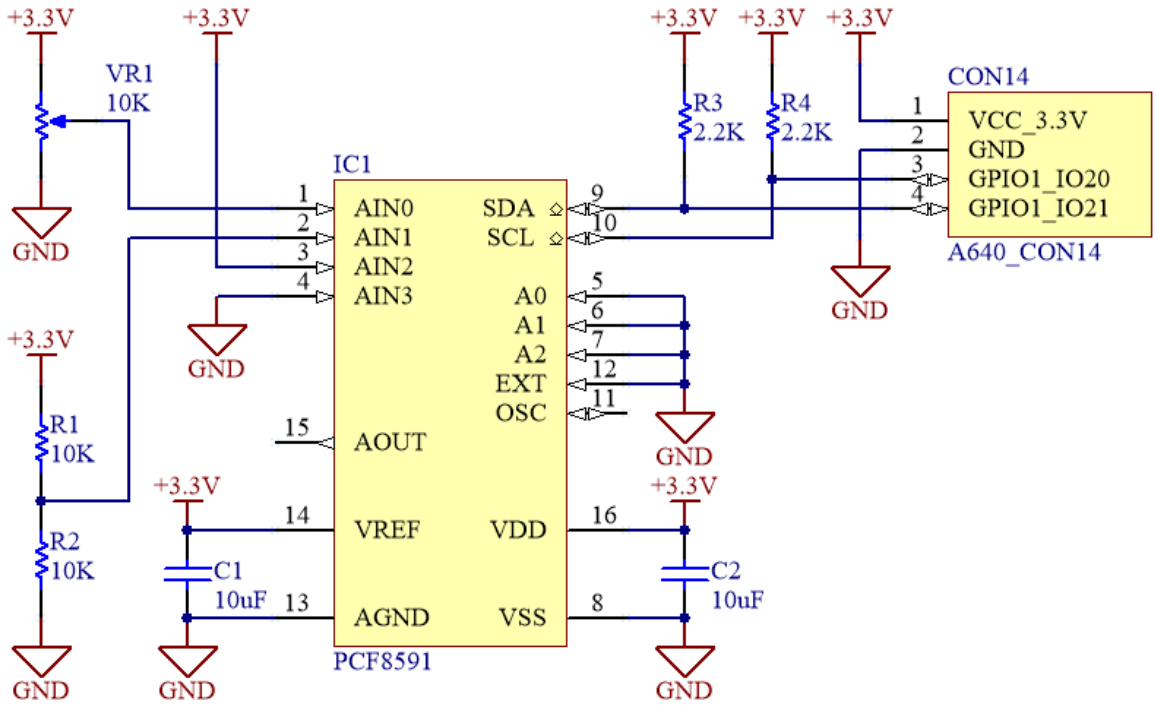

図 5.7 I²C 接続 A/D コンバーター回路図

5.2.2. 対応カーネルイメージの作成

PCF8591 のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の編集
2. カーネルコンフィギュレーションでのデバイスドライバの有効化
3. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-i2c4.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_i2c4: i2c4grp {
                fsl,pins= <
                        MX6UL_PAD_UART2_TX_DATA__I2C4_SCL 0x40010808
                        MX6UL_PAD_UART2_RX_DATA__I2C4_SDA 0x40010808
                >;
        };
};

&i2c4 {
        status = "okay";
        clock-frequency = <50000>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c4>;
```

```
        pcf8591@48 {                            // ❶
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "nxp,pcf8591";
                reg = <0x48>;                   // ❷
                input_mode = <0>;
        };
};
```

図 5.8 armadillo-640-i2c4.dtsi

❶ デバイス名@スレーブアドレス(16 進表記)

❷ スレーブアドレス(16 進表記)

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-i2c4.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 5.9 armadillo-640.dts

続いて、「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 5.10 menuconfig の実行

```
Device Drivers  --->
  [*] Hardware Monitoring support  --->   ← 有効にする
    [*]   Philips PCF8591 ADC/DAC         ← 有効にする
```

図 5.11 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

5.2.3. 使用例

実際に、PCF8591 から値の取得をおこなう手順を説明します。変換された値の末尾に'0'が追加された文字列が返ります。

```
[armadillo ~]# cat /sys/bus/i2c/devices/3-0048/in0_input
0120                                          ←VR1 を回すと値が変化
[armadillo ~]# cat /sys/bus/i2c/devices/3-0048/in0_input
1230                                          ←VR1 を回すと値が変化
[armadillo ~]# cat /sys/bus/i2c/devices/3-0048/in0_input
2340                                          ←VR1 を回すと値が変化
[armadillo ~]# cat /sys/bus/i2c/devices/3-0048/in1_input
1280
[armadillo ~]# cat /sys/bus/i2c/devices/3-0048/in2_input
2550
[armadillo ~]# cat /sys/bus/i2c/devices/3-0048/in3_input
0
```

図 5.12 コマンド実行例

5.2.4. プロトコル

PCF8591 のデータには 3 種類あります。

表 5.2 PCF8591 のデータバイト

データ名	データ方向
コントロールバイト	W
D/A 出力値	W
A/D 入力値	R

コントロールバイトの書き込みは以下のように行います。

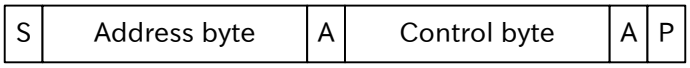

図 5.13 PCF8591 通信フォーマット(書き込み)

D/A 出力値を書き込む場合は、コントロールバイトに続けて書き込みます。

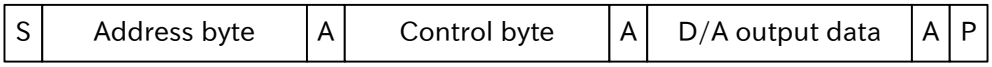

図 5.14 PCF8591 通信フォーマット(D/A 出力値書き込み)

A/D 入力値の読み出しは以下のように行います。

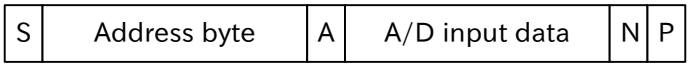

図 5.15 PCF8591 通信フォーマット(読み出し)

PCF8591 のアドレスバイトのフォーマットを示します。7 ビットアドレスの上位 4 ビットは固定で 1001 です。下位 3 ビットは対応するピン(A2〜A0)で設定可能です。今回の例では全て GND に接続したので、A2〜A0 は全て 0 になります。

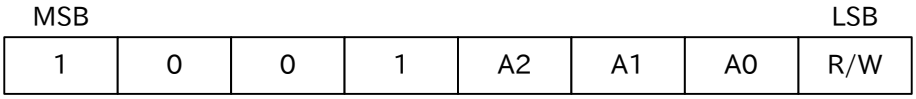

図 5.16 PCF8591 アドレスバイト

コントロールバイトのフォーマットを示します。AOE を 1 にすると D/A 出力が有効になります。AISEL1〜0 は A/D 入力のモードを指定します。両方 0 でシングルエンド入力となります。AINC は 1 にするとオートインクリメントが有効になります。CH1〜0 は A/D 入力のチャンネル 0〜3 を 2 進数で指定します。

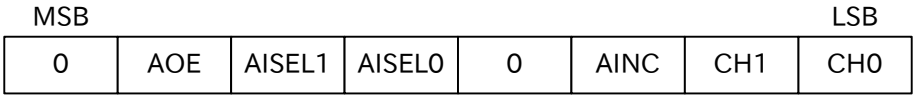

図 5.17 PCF8591 コントロールバイト

A/D 入力値、D/A 出力値のフォーマットを示します。どちらも 1 バイトに 8 ビットのデータがそのまま入っています。

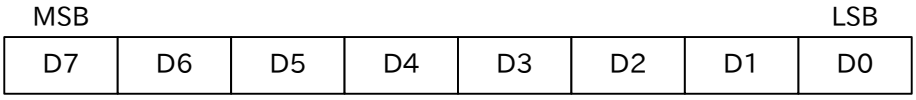

図 5.18 PCF8591 データバイト

サンプリング・A/D 変換が行われるタイミング

PCF8591 では、ACK の後にサンプリングが行われ、データバイトの読み出し中に A/D 変換がおこなわれます。そのため、通信の最初のデータバイトで転送される値は、前回の通信中に変換された値となります。また、電源投入後の最初のデータバイトで転送される値は 0x80 となります。今回使用するドライバでは、対策として電源投入後やコントロールバイト変更後は 2 回読み出しています。

5.3. I/O エクスパンダー(PCF8574)を使用する

ここでは、I²C バスに I/O エクスパンダーを接続する方法を紹介します。

使用するデバイスは以下のとおりです。

- PCF8574(NXP Semiconductors 製)

今回使用する PCF8574 は、以下の特長を持ちます。

- 単電源動作(2.5〜6V)
- I²C 接続(スタンダードモード)
- アドレス 0x20〜0x27(同一バスに 8 つ接続可能)
- GPIO 数 8

・入出力方向の設定不要

・入力変化割り込み機能あり

5.3.1. 接続方法

Armadillo-640 との接続を示します。Armadillo-640 の CON14 から出ている I2C4 に PCF8574 を接続します。アドレスを指定する A0～A2 は全て GND に接続しておきます[6]。例として、スイッチや LED を接続しています。スイッチはオンで Low 入力になるようにし、LED は Low 出力で点灯するようにします。

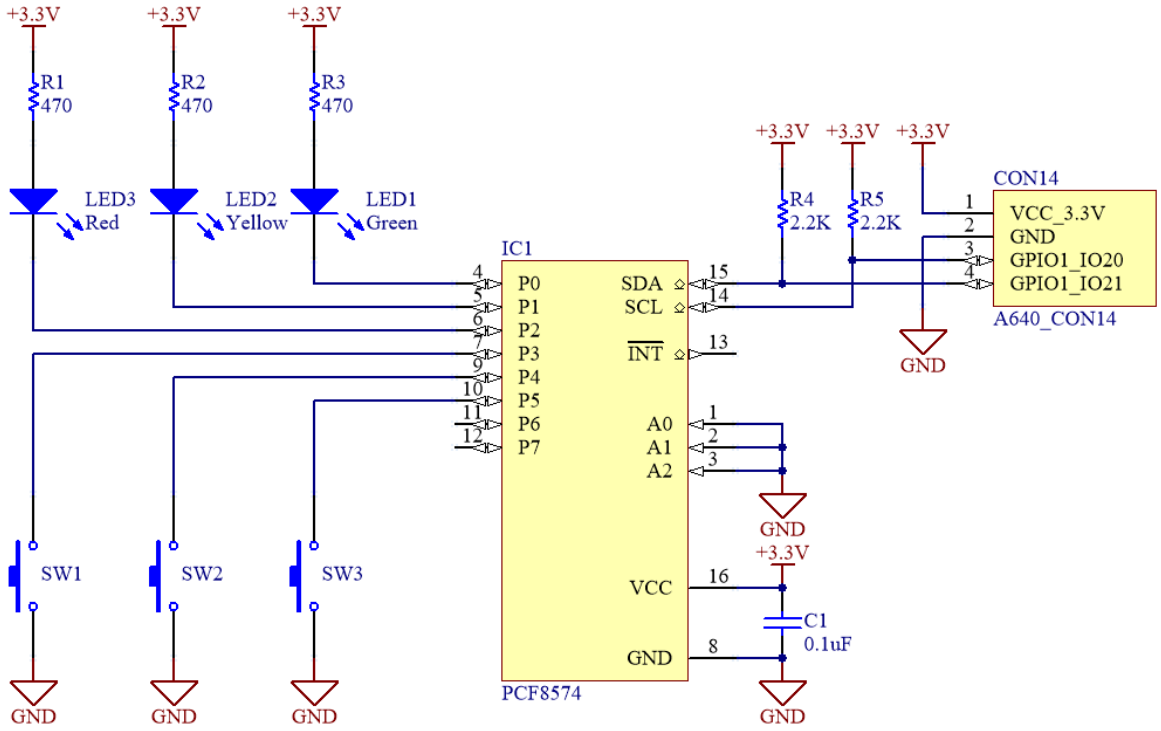

図 5.19 I2C 接続 I/O エクスパンダー回路図

5.3.2. 対応カーネルイメージの作成

PCF8574 のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の編集
2. カーネルコンフィギュレーションでのデバイスドライバの有効化
3. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-i2c4.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

[6]複数のスレーブをバスに接続する場合は、A0～A2 の設定を変えてアドレスが重複しないようにしてください。

```
&iomuxc {
        pinctrl_i2c4: i2c4grp {
                fsl,pins= <
                        MX6UL_PAD_UART2_TX_DATA__I2C4_SCL 0x40010808
                        MX6UL_PAD_UART2_RX_DATA__I2C4_SDA 0x40010808
                >;
        };
};

&i2c4 {
        status = "okay";
        clock-frequency = <50000>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c4>;

        pcf8574@20 {                            // ❶
                #address-cells = <2>;
                #size-cells = <0>;
                compatible = "nxp,pcf8574";
                reg = <0x20>;                   // ❷
                gpio-controller;
                #gpio-cells = <2>;
        };
};
```

図 5.20 armadillo-640-i2c4.dtsi

❶ デバイス名@スレーブアドレス(16 進表記)

❷ スレーブアドレス(16 進表記)

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-i2c4.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 5.21 armadillo-640.dts

続いて「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 5.22 menuconfig の実行

```
Device Drivers  --->
  -*- GPIO Support  --->
```

```
    I2C GPIO expanders  --->
        [*] PCF857x, PCA{85,96}7x, and MAX732[89] I2C GPIO expanders    ← 有効にする
```

図 5.23 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

5.3.3. 使用例

まず、GPIO クラスディレクトリを作成します。

```
[armadillo ~]# ls /sys/class/gpio/
export     gpiochip128  gpiochip504  gpiochip96
gpiochip0  gpiochip32   gpiochip64   unexport
[armadillo ~]# echo 504 > /sys/class/gpio/export
[armadillo ~]# echo 505 > /sys/class/gpio/export
[armadillo ~]# echo 506 > /sys/class/gpio/export
[armadillo ~]# echo 507 > /sys/class/gpio/export
[armadillo ~]# echo 508 > /sys/class/gpio/export
[armadillo ~]# echo 509 > /sys/class/gpio/export
[armadillo ~]# echo 510 > /sys/class/gpio/export
[armadillo ~]# echo 511 > /sys/class/gpio/export
[armadillo ~]# ls /sys/class/gpio/
export   gpio506  gpio509  gpiochip0    gpiochip504  unexport    ←gpio504～gpio511 が作成された
gpio504  gpio507  gpio510  gpiochip128  gpiochip64
gpio505  gpio508  gpio511  gpiochip32   gpiochip96
[armadillo ~]# cat /sys/class/gpio/gpio504/direction
in                                                              ←初期状態では入力
```

図 5.24 GPIO クラスディレクトリ作成コマンド実行例

次に、実際に PCF8574 の GPIO にアクセスし、LED を点灯させてみます。

```
[armadillo ~]# cat /sys/class/gpio/gpio504/direction
in                                                          ←初期状態では入力
[armadillo ~]# echo high > /sys/class/gpio/gpio504/direction    ←high に設定しても LED1 は消灯のまま
[armadillo ~]# echo low > /sys/class/gpio/gpio504/direction     ←low に設定すると LED1 が点灯
[armadillo ~]# echo 1 > /sys/class/gpio/gpio504/value           ←1 に設定すると LED1 が消灯
[armadillo ~]# echo 0 > /sys/class/gpio/gpio504/value           ←0 に設定すると LED1 が点灯
```

図 5.25 出力コマンド実行例

最後に、スイッチの状態を取得してみます。

```
[armadillo ~]# cat /sys/class/gpio/gpio505/value                ←スイッチが ON の時に実行
0
```

```
[armadillo ~]# cat /sys/class/gpio/gpio505/value                ←スイッチが OFF の時に実行
1
```

図 5.26 入力コマンド実行例

5.3.4. プロトコル

PCF8574 のデータには 2 種類あります。

表 5.3 PCF8574 のデータバイト

データ名	データ方向
GPIO 出力値	W
GPIO 入力値	R

GPIO 出力値の書き込みは以下のように行います。

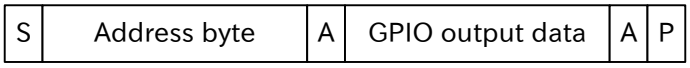

図 5.27 PCF8574 通信フォーマット(書き込み)

GPIO 入力値の読み出しは以下のように行います。

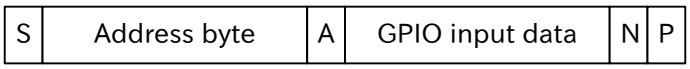

図 5.28 PCF8574 通信フォーマット(読み出し)

PCF8574 のアドレスバイトのフォーマットを示します。7 ビットアドレスの上位 4 ビットは固定で 0100 です。下位 3 ビットは対応するピン(A2〜A0)で設定可能です。今回の例では全て GND に接続したので、A2〜A0 は全て 0 になります。

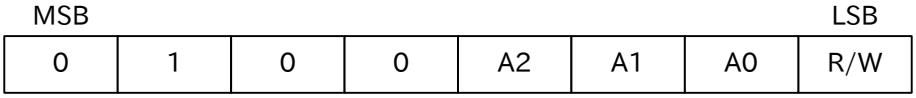

図 5.29 PCF8574 アドレスバイト

GPIO 出力値、GPIO 入力値のフォーマットを示します。各ビットはピン P7〜P0 にそのまま対応します。GPIO 出力値で 0 を書き込むと、Low 出力になり、1 を書き込むと、High 出力になります。High 出力は弱いプルアップとなっており、入力も兼ねています。そのため、「図 5.19. I²C 接続 I/O エクスパンダー回路図」のようにスイッチはオンで Low 入力になるようにし、LED は Low 出力で点灯するようにします。

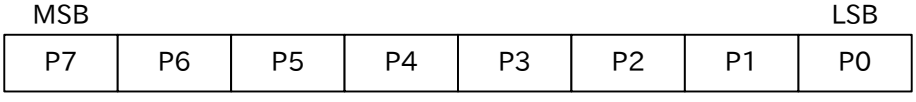

図 5.30 PCF8574 データバイト

5.4. 温度湿度センサー(HDC1080)を使用する

ここでは、I²C バスに温度湿度センサーを接続する方法を紹介します。

使用するデバイスは以下のとおりです。

- HDC1080(Texas Instruments 製)

今回使用する HDC1080 は、以下の特長を持ちます。

- 単電源動作(2.7～5.5V)
- I2C 接続(ファストモード)
- アドレス 0x40
- 測定範囲 温度: -40～125℃ 湿度: 0～100%RH
- 精度 温度: 0.2℃ 湿度: 2%RH
- 分解能 温度: 最大 14 ビット 湿度: 最大 14 ビット

5.4.1. 接続方法

Armadillo-640 との接続を示します。Armadillo-640 の CON14 から出ている I2C4 に HDC1080 を接続します。

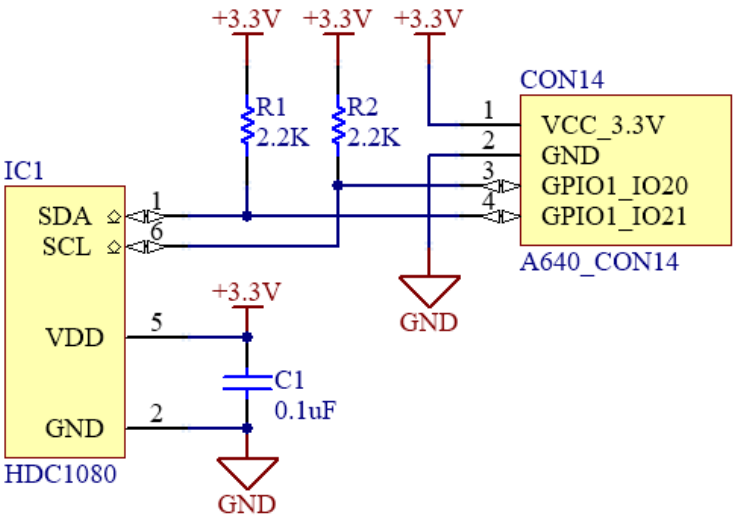

図 5.31 I2C 接続温度湿度センサー回路図

5.4.2. 対応カーネルイメージの作成

HDC1080 のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の編集
2. カーネルコンフィギュレーションでのデバイスドライバの有効化
3. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-i2c4.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_i2c4: i2c4grp {
                fsl,pins= <
                        MX6UL_PAD_UART2_TX_DATA__I2C4_SCL 0x40010808
                        MX6UL_PAD_UART2_RX_DATA__I2C4_SDA 0x40010808
                >;
        };
};

&i2c4 {
        status = "okay";
        clock-frequency = <50000>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c4>;

        hdc1080@40 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "ti,hdc1080";
                reg = <0x40>;
        };
};
```

図 5.32 armadillo-640-i2c4.dtsi

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-i2c4.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 5.33 armadillo-640.dts

続いて「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 5.34 menuconfig の実行

```
Device Drivers  --->
  [*] Industrial I/O support  --->                          ← 有効にする
```

```
    Humidity sensors  --->
      [*] TI HDC100x relative humidity and temperature sensor   ← 有効にする
```

図 5.35 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

5.4.3. 使用例

実際に、HDC1080 から値の取得をおこなう手順を説明します。

```
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_humidityrelative_scale
0.001525878
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_humidityrelative_raw
25296                                                   ←湿度:38.6%
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_humidityrelative_raw
48136                                                   ←湿度:73.4%
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_temp_offset
-15887.515151
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_temp_scale
2.517700195
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_temp_raw
25296                                                   ←温度:23.687℃
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_temp_raw
28448                                                   ←温度:31.623℃
```

図 5.36 コマンド実行例

湿度(%)は以下の式で求められます。

```
in_humidityrelative_raw × in_humidityrelative_scale
= 25296 × 0.001525878
= 38.598609888
```

図 5.37 湿度の計算

温度(1/1000℃)は以下の式で求められます。

```
(in_temp_raw + in_temp_offset) × in_temp_scale
= (25296 + (-15887.515151) ) × 2.517700195
= 23687.744138981845555
```

図 5.38 温度の計算

5.4.4. プロトコル

HDC1080 のアドレスバイトのフォーマットを示します。7 ビットアドレスは固定で 1000000 です。

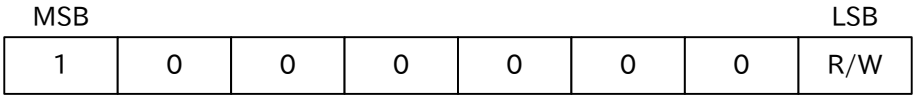

図 5.39 HDC1080 アドレスバイト

HDC1080 には 16 ビットのレジスタが 8 個あります。ポインターの書き込みによってレジスタを指定してアクセスします。

表 5.4 HDC1080 のレジスタ

ポインター	レジスタ名	データ方向
0x00	Temperature(温度)	R
0x01	Humidity(湿度)	R
0x02	Configuration	R/W
0xFB～FD	Serial ID(40 ビット)	R
0xFE	Manufacturer ID	R
0xFF	Device ID	R

温度の読み出しは以下のように行います。ポインター(0x00)の書き込みで温度の取得が開始されるため、6.5ms 以上待ってから読み出しを行います。データは 14 ビットなので、下位 2 ビットが 0 埋めになっています。湿度も同様に、ポインター(0x01)の書き込み後に読み出します。

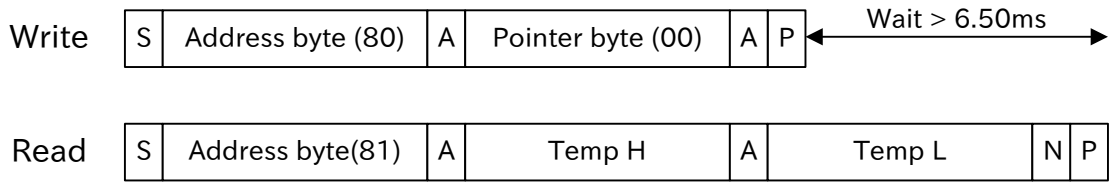

図 5.40 HDC1080 通信フォーマット(読み出し)

コンフィグレーションの書き込みは以下のように行います。

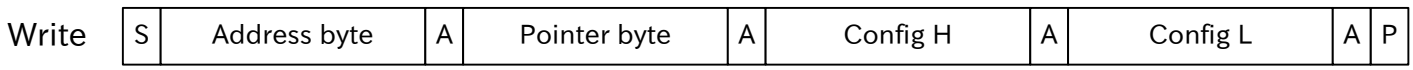

図 5.41 HDC1080 通信フォーマット(書き込み)

コンフィグレーションレジスタのフォーマットを示します。デバッグを行う上であまり重要ではないため、各ビットの説明は省きます。

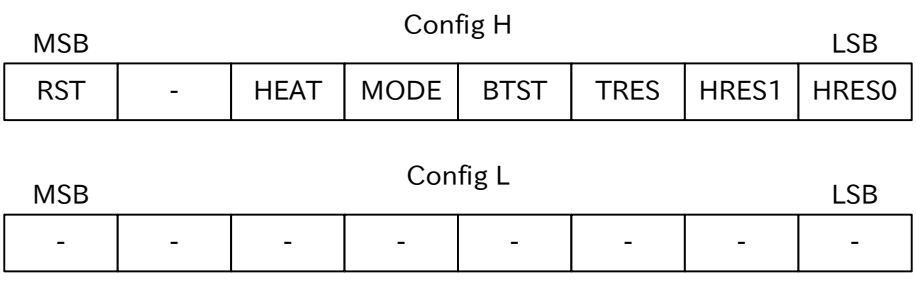

図 5.42 HDC1080 コンフィグレーションレジスタ

5.5. VOCs センサー(CCS811)を使用する

ここでは、I2C バスに VOCs センサーを接続する方法を紹介します。

使用するデバイスは以下のとおりです。

- CCS811(ams AG 製)

今回使用する CCS811 は、以下の特長を持ちます。

- 単電源動作(1.8～3.3V)
- I2C 接続(ファストモード)
- アドレス 0x5A～0x5B(同一バスに 2 つ接続可能)
- VOCs(揮発性有機化合物)、CO_2 の濃度を測定
- VOCs: 0～1156ppb[7]
- CO_2: 400～7992ppm[7]
- 屋内用

5.5.1. 接続方法

Armadillo-640 との接続を示します。Armadillo-640 の CON14 から出ている I2C4 に CCS811 を接続します。アドレスを指定する ADDR は GND に接続しておきます[8]。

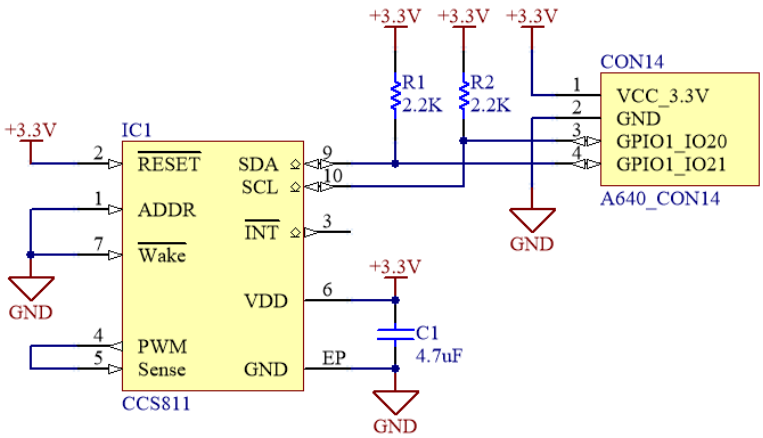

図 5.43 I2C 接続 VOCs センサー回路図

5.5.2. 対応カーネルイメージの作成

CCS811 のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の編集

[7]動作確認時に表示された値
[8]複数のスレーブをバスに接続する場合は、ADDR の設定を変えてアドレスが重複しないようにしてください。

2. カーネルコンフィギュレーションでのデバイスドライバの有効化
3. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-i2c4.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_i2c4: i2c4grp {
                fsl,pins= <
                        MX6UL_PAD_UART2_TX_DATA__I2C4_SCL 0x40010808
                        MX6UL_PAD_UART2_RX_DATA__I2C4_SDA 0x40010808
                >;
        };
};

&i2c4 {
        status = "okay";
        clock-frequency = <50000>;
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_i2c4>;

        ccs811@5A {                             // ❶
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "ccs811";
                reg = <0x5A>;                   // ❷
        };
};
```

図 5.44 armadillo-640-i2c4.dtsi

❶ デバイス名@スレーブアドレス(16 進表記)

❷ スレーブアドレス(16 進表記)

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-i2c4.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 5.45 armadillo-640.dts

続いて「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 5.46 menuconfig の実行

```
Device Drivers  --->
  [*] Industrial I/O support  --->       ← 有効にする
    Humidity sensors  --->
      [*] AMS CCS811 VOC sensor          ← 有効にする
```

図 5.47 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

5.5.3. 使用例

実際に、CCS811 から値の取得をおこなう手順を説明します。VOCs 濃度(ppb)、CO2 濃度(ppm)が出力されます。

```
[armadillo ~]# cat /sys/class/i2c-dev/i2c-3/device/3-005a/iio\:device0/in_concentration_voc_raw
53
[armadillo ~]# cat /sys/class/i2c-dev/i2c-3/device/3-005a/iio\:device0/in_concentration_voc_raw
1156                                                    ←息を吹きかけると VOCs 濃度が上がる
[armadillo ~]# cat /sys/class/i2c-dev/i2c-3/device/3-005a/iio\:device0/in_concentration_co2_raw
751
[armadillo ~]# cat /sys/class/i2c-dev/i2c-3/device/3-005a/iio\:device0/in_concentration_co2_raw
7992                                                    ←息を吹きかけると CO2 濃度が上がる
```

図 5.48 コマンド実行例

5.5.4. プロトコル

CCS811 のアドレスバイトのフォーマットを示します。アドレスの上位 6 ビットは固定で 101101 です。

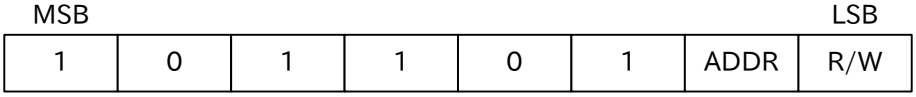

図 5.49 CCS811 アドレスバイト

CCS811 にはレジスタが 14 個あります。ポインターの書き込みによってレジスタを指定してアクセスします。レジスタのバイト数はそれぞれ異なります。ここでは重要なレジスタのみ紹介します。

表 5.5 CCS811 のレジスタ

ポインター	レジスタ名	バイト数	データ方向
0x00	STATUS	1	R
0x02	ALG_RESULT_DATA	8	R

STATUS レジスタの読み出しは以下のように行います。

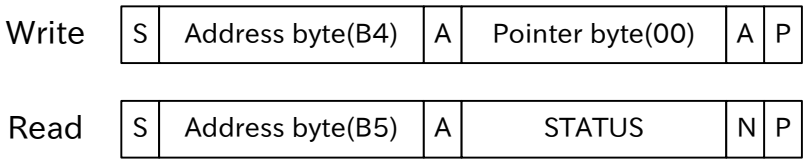

図 5.50 CCS811 通信フォーマット(STATUS 読み出し)

CO_2 や VOCs の濃度の読み出しは以下のように行います。

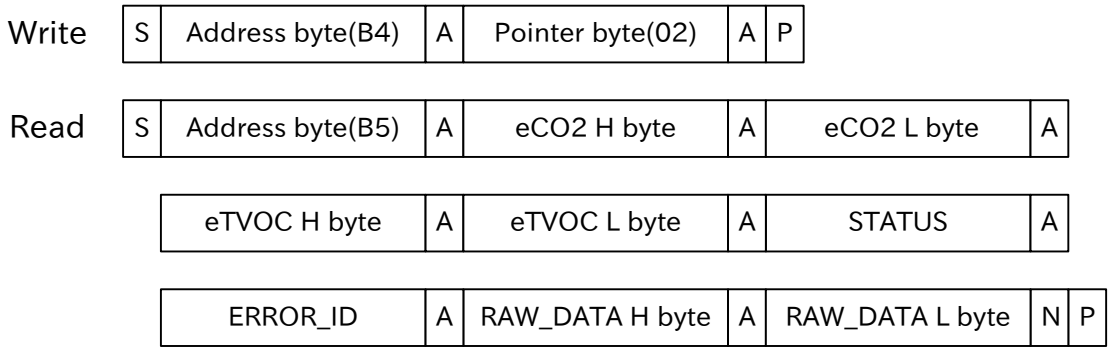

図 5.51 CCS811 通信フォーマット(ALG_RESULT_DATA 読み出し)

STATUS レジスタのフォーマットを示します。

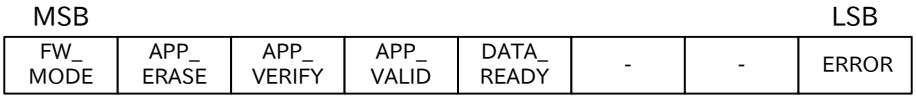

図 5.52 CCS811 STATUS レジスタ

読み出されていないデータがあるとき、DATA_READY が 1 になります。今回使用したドライバでは、DATA_READY が 1 になるまで待ち、新しいデータが用意された後に濃度を読み出します。

6. SPI デバイスの活用

6.1. SPI とは

SPI は「Serial Peripheral Interface」の略で、主に機器内の通信に用いられる通信方式です。I²C と同様に機器内のセンサーやメモリー等のインターフェースとして普及していますが、より近距離で高速な通信に用いられる場合が多いです。

一般的な接続方法を下図に示します。なお、Armadillo-640 はマスターとして別のデバイスと通信できます。

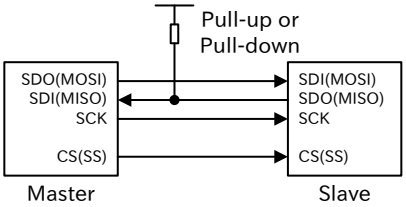

図 6.1 SPI の接続方法

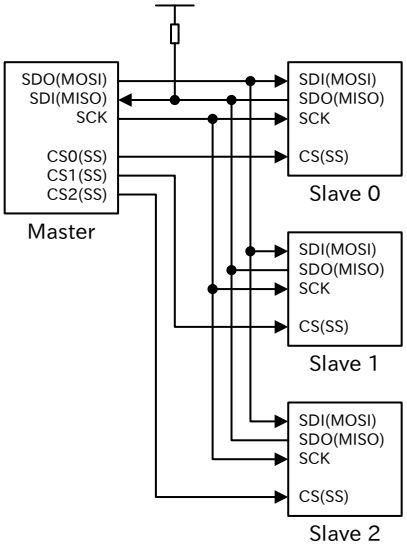

図 6.2 複数のスレーブを接続する場合

SPI の主な特徴を以下に示します。

- デバイスはマスターとスレーブが存在
- 1 つのバス上にマスターは 1 つ、スレーブは複数接続可能
- データはクロックに合わせて変化
- 信号線はクロック、送受信のデータ、スレーブセレクト(SS)
- 通信は常にマスターが開始し、SS=Low でスレーブを選択
- クロックは最高 100MHz 程度

SPI では信号名・ピン名は統一されておらず、主に 2 通りの呼び方があります。

- シリアルクロック(SCLK) = SCK
- マスターアウト/スレーブイン(MOSI) = [マスターの SDO、スレーブの SDI]
- マスターイン/スレーブアウト(MISO) = [マスターの SDI、スレーブの SDO]

・ スレーブセレクト(SS) = チップセレクト(CS)

SDO は相手の SDI に接続します。双方向通信が不要の場合、MOSI もしくは MISO を接続しないことがあります。

SPI ではクロックの極性(Polarity)と位相(Phase)も統一されておらず、通信相手に合わせて設定する必要があります。極性は CPOL、位相は CPHA で設定します。

・ CPOL=0: クロックを出力していないとき SCLK を Low に保ちます。

・ CPOL=1: クロックを出力していないとき SCLK を High に保ちます。

・ CPHA=0: クロックの奇数番目の変化でデータ取り込み、偶数番目の変化でシフト[1]

・ CPHA=1: クロックの奇数番目の変化でシフトし、偶数番目の変化でデータ取り込む

CPOL と CPHA の組み合わせを、SPI モードで表現する場合もあります。

表 6.1 SPI モード

モード	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

各モードの波形は以下のようになっています。

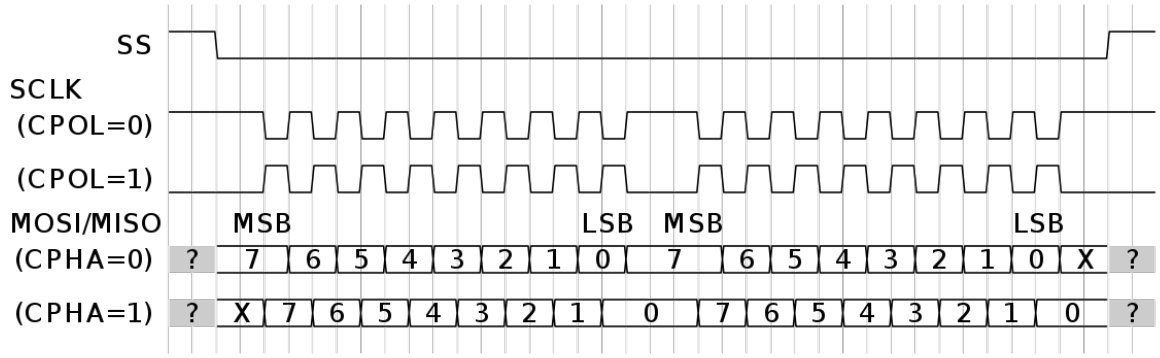

図 6.3 SPI モードと波形

6.2. A/D コンバーター(MCP3204)を使用する

ここでは、SPI バスに A/D コンバーターを接続する方法を紹介します。

使用するデバイスは以下のとおりです。

・ MCP3204(Microchip Technology 製)

今回使用する MCP3204 は、以下の特長を持ちます。

・ 単電源動作(2.7～5.5V)

[1]最初のデータは SS が Low に変化した時に出力されます。

- SPI 接続 最大 2MHz(Vdd=5V 時)、最大 1MHz(Vdd=2.7V 時)
- サンプリング速度 最大 100ksps(Vdd=5V 時)、50ksps(Vdd=2.7V 時)
- 分解能 12 ビット
- 逐次比較型
- 4 入力
- 疑似差動入力によるコモンモードノイズ除去が可能

6.2.1. 接続方法

Armadillo-640 との接続を示します。Armadillo-640 の CON9 から出ている ECSPI1 に MCP3204 を接続します。SS 信号には、GPIO を使用します。AIN0～AIN3 がアナログ入力ピンです。AIN0 にかかる電圧を、10kΩ の可変抵抗で変えられるようにしています。AIN1～AIN3 はそれぞれ固定電圧としています。リファレンス電圧 VREF に電源電圧と同じ 3.3V を入力しているため、0V から 3.3V の範囲のアナログ入力を 12 ビット(4096 段階)のデジタル値に変換します。

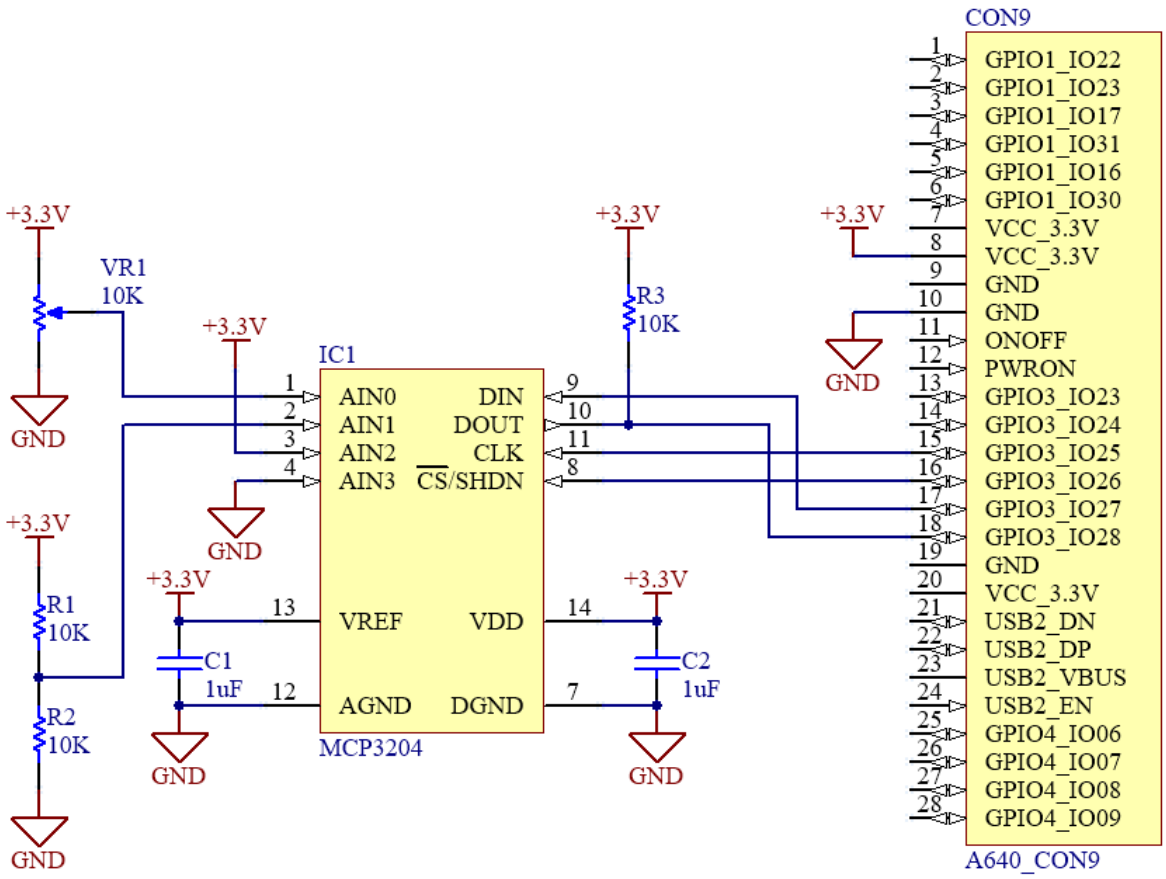

図 6.4 SPI 接続 A/D コンバーター回路図

6.2.2. 対応カーネルイメージの作成

SPI ドライバ・MCP3204 のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の編集
2. カーネルコンフィギュレーションでの SPI の有効化
3. カーネルコンフィギュレーションでのデバイスドライバの有効化
4. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-ecspi1.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_ecspi1: ecspi1grp {
                fsl,pins = <
                        MX6UL_PAD_LCD_DATA20__ECSPI1_SCLK       0x1b0b0 // CON9_15
                        MX6UL_PAD_LCD_DATA21__GPIO3_IO26        0x1b0b0 // CON9_16
                        MX6UL_PAD_LCD_DATA22__ECSPI1_MOSI       0x1b0b0 // CON9_17
                        MX6UL_PAD_LCD_DATA23__ECSPI1_MISO       0x1b0b0 // CON9_18
                >;
        };
};

&ecspi1 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_ecspi1>;
        cs-gpios = <&gpio3 26 GPIO_ACTIVE_HIGH>;
        status = "okay";

        mcp3204@0 {
                #address-cells = <1>;
                #size-cells = <0>;
                compatible = "mcp3204";
                spi-max-frequency = <1000000>;
                reg = <0>;
        };
};
```

図 6.5 armadillo-640-ecspi1.dtsi

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-ecspi1.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 6.6 armadillo-640.dts

続いて「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 6.7 menuconfig の実行

```
Device Drivers  --->
  [*] SPI support  --->                                 ← 有効にする
    [*]   Freescale i.MX SPI controllers                ← 有効にする
  [*] Industrial I/O support  --->                      ← 有効にする
    Analog to digital converters  --->
      [*] Microchip Technology MCP3x01/02/04/08         ← 有効にする
```

図 6.8 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

6.2.3. 使用例

実際に、MCP3204 から値の取得をおこなう手順を説明します。

```
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw
0123                                                    ←VR1 を回すと値が変化
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw
1234                                                    ←VR1 を回すと値が変化
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage0_raw
2345                                                    ←VR1 を回すと値が変化
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage1_raw
2048
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage2_raw
4095
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage3_raw
0
[armadillo ~]# cat /sys/bus/iio/devices/iio\:device0/in_voltage2-voltage3_raw
2048                                                    ←CH2-CH3 の差動電圧
```

図 6.9 コマンド実行例

6.2.4. プロトコル

MCP3204 は、SPI モード 0(CPOL=0、CPHA=0)または SPI モード 3(CPOL=1、CPHA=1)で通信をおこないます。MCP3204 の通信フォーマットを「図 6.10. MCP3204 通信フォーマット」に示します。

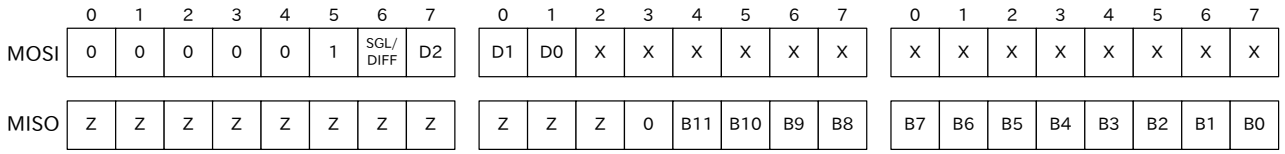

図 6.10 MCP3204 通信フォーマット

MOSI から 1 を出力することで、転送の開始を MCP3204 に指示します。SGL/DIFF*、D2、D1、D0 の組み合わせにより、A/D 変換をおこなうチャンネルを指定します。D0 以降、MOSI から出力されるデータは意味を持ちません。

表 6.2 MCP3204 チャンネル指定

SGL/DIFF	D2[a]	D1	D0	入力構成	チャンネル
1	d.c.	0	0	シングルエンド	CH0
1	d.c.	0	1	シングルエンド	CH1
1	d.c.	1	0	シングルエンド	CH2
1	d.c.	1	1	シングルエンド	CH3
0	d.c.	0	0	ディファレンシャル	CH0=IN+、CH1=IN-
0	d.c.	0	1	ディファレンシャル	CH0=IN-、CH1=IN+
0	d.c.	1	0	ディファレンシャル	CH2=IN+、CH3=IN-
0	d.c.	1	1	ディファレンシャル	CH2=IN-、CH3=IN+

[a]MCP3204 では D2 は意味を持ちません。

MCP3204 は、11 番目のクロックの立ち上がりでアナログ入力のサンプリングを開始し、次のクロックの立ち下がりで完了します。A/D 変換結果は、B11～B0 に出力されます。

7. 1-Wire デバイスの活用

7.1. 1-Wire とは

1-Wire は主に機器内のセンサー等との通信に用いられる通信方式です。

一般的な接続方法を下図に示します。なお、Armadillo-640 は、マスターとして別のデバイスと通信できます。

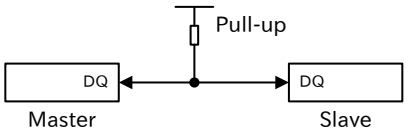

図 7.1 1-Wire の接続方法

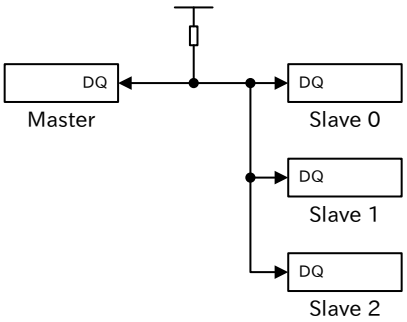

図 7.2 複数のスレーブを接続する場合

1-Wire の主な特徴を以下に示します。

- 非同期式シリアル通信
- デバイスはマスターとスレーブが存在
- 1 つのバス上にマスターは 1 つ、スレーブは複数接続可能
- 信号線は 1 本のみ[1]
- 信号線はオープンドレインなため、プルアップが必要
- スレーブは製造時に与えられた 64 ビットの固有 ID を持つ
- 通信は常にマスターが開始し、固有 ID でスレーブを選択
- 約 16.3kbps

固有 ID はマスターがバスを起動した際、もしくはスレーブがバスに接続された際にマスターによって読み出され記憶されます。

1-Wire ではクロック信号を別途設けず、Low パルスの幅を用いてデータの転送をおこないます。1 ビットに対応する周期を「タイムスロット」といいます。

なお、タイムスロットにはスタンダード(1 タイムスロット 60μsec)とオーバードライブ(1 タイムスロット 8μsec)の二つがあります。以下の説明はスタンダードの場合のタイミングです。

[1]通信線で電源供給も行う場合もあります。

マスターは 1 を送信する場合は、1μs 以上かつ短いパルスを出力します。0 を送信する場合は、60~120μs のパルスを出力します。スレーブは立ち下がりエッジから 15〜60μs の間に信号線のサンプリングをおこないます。

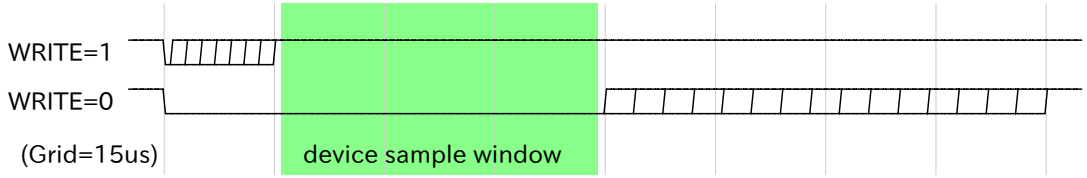

図 7.3 1-Wire プロトコル(書き込みスロット)

値を読み出す場合、まず、マスターが 1μs 以上かつ短いパルスを出力します。スレーブは 1 を送信したい場合は何もしません。0 を送信したい場合、立ち下がりエッジから 15μs 信号線を L レベルに保ちます。マスターはマスター自体が出力するパルス期間が終わったあと、立ち下がりエッジから 15μs 以内に信号線のサンプリングをおこないます。

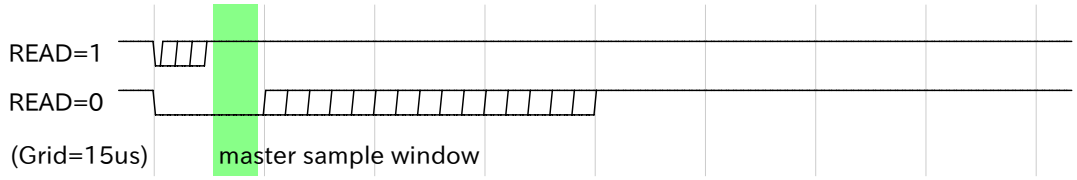

図 7.4 1-Wire プロトコル(読み出しスロット)

マスターとスレーブ間でのデータ転送は 3 つのシーケンスでおこないます。3 つのシーケンスは、リセットシーケンス、ROM コマンドシーケンス、ファンクションシーケンスの順番に実行されます。

リセットシーケンスでは、まず、マスターがリセットパルスを出力します。1-Wire バスにスレーブが接続されている場合、スレーブはプレゼンスパルスを出力します。

ROM コマンドシーケンスでは、マスターが 8 ビットの ROM コマンドを出力した後、64 ビットの ROM ID を出力します。ROM ID の先頭 8 ビットはデバイス種類を示すファミリーコードです。続く 48 ビットがシリアルナンバーになっています。最後の 8 ビットは CRC です。ROM コマンドには次のものがあります。

1. SEARCH ROM(0xF0): バスに接続されているスレーブデバイスの ROM ID を得ることができます。1 回の SEARCH ROM コマンドで 1 つのデバイスの ROM ID を特定することができます。

2. READ ROM(0x33): バスに接続されているスレーブデバイスが一つだけの場合、SEARCH ROM コマンドの代わりに READ ROM コマンドを使用して、ROM ID を得ることができます。

3. MATCH ROM(0x55): MATCH ROM コマンドでマスターが出力した ROM ID に一致したスレーブデバイスが、続くファンクションコマンドに応答します。それ以外のデバイスは、次のリセットシーケンスを待ちます。

4. SKIP ROM(0xcc): SKIP ROM コマンドに続いて送信されたファンクションコマンドは、バスに接続されているスレーブデバイス全てに同時に適用されます。

ファンクションシーケンスは、スレーブデバイスごとに異なります。基本的には、マスターが 8 ビットのフォワードコマンド出力した後、データの読み出しまたは書き込みをおこないます。

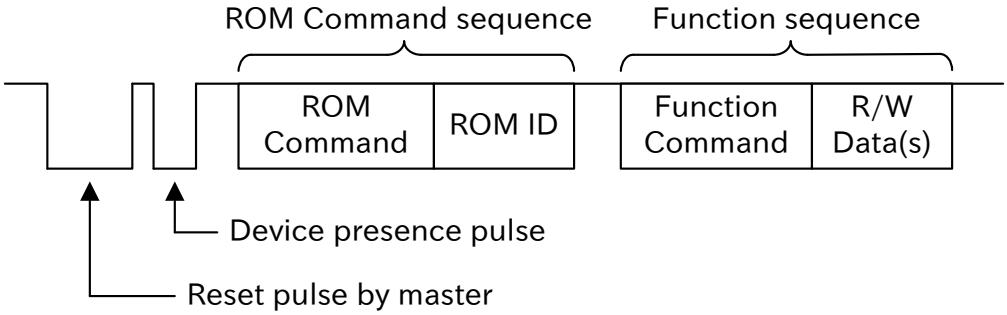

図 7.5 1-Wire プロトコル

7.2. 温度センサー(DS18B20)を使用する

ここでは、1-Wire バスに温度センサー IC を接続する方法を紹介します。

使用するデバイスは以下のとおりです。

- DS18B20(Maxim Integrated 製)

今回使用する DS18B20 は、以下の特長を持ちます。

- 単電源動作(3.0～5.5V)
- 電源 信号線供給・電源線供給
- 温度計測範囲 -55～125℃
- 分解能 9～12 ビット(設定により可変)
- 変換時間(9 ビット) 94msec
- 変換時間(12 ビット) 750msec

7.2.1. 接続方法

Armadillo-640 と温度センサーとの接続を示します。信号線は、CON9 2 ピンに接続します。また、今回は VDD に+3.3V を接続し電源は外部から供給します。

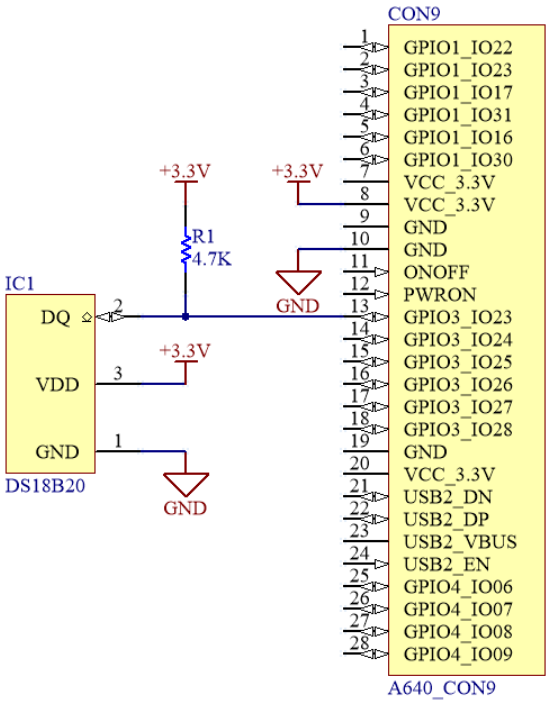

図 7.6 1-Wire 接続温度センサー回路図

7.2.2. 対応カーネルイメージの作成

1-Wire ドライバ・DS18B20 のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の集
2. カーネルコンフィギュレーションでの 1-Wire の有効化
3. カーネルコンフィギュレーションでのデバイスドライバの有効化
4. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-w1.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_w1: w1grp {
                fsl,pins= <
                        MX6UL_PAD_LCD_DATA18__GPIO3_IO23 0x40010808
                >;
        };
};

/{
        onewire@0 {
                compatible = "w1-gpio";
                gpios = <&gpio3 23 GPIO_ACTIVE_LOW>;
```

```
                pinctrl-names = "default";
                pinctrl-0 = <&pinctrl_w1>;
                status="okay";
        };
};
```

図 7.7 armadillo-640-w1.dtsi

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-w1.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 7.8 armadillo-640.dts

続いて「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 7.9 menuconfig の実行

```
Device Drivers  --->
  [*] Dallas's 1-wire support  --->                  ← 有効にする
     1-wire Bus Masters  --->
        [*] GPIO 1-wire busmaster                    ← 有効にする
     1-wire Slaves  --->
        [*] Thermal family implementation            ← 有効にする
   [*] Hardware Monitoring support  --->             ← 有効にする
```

図 7.10 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

7.2.3. 使用例

実際に、DS18B20 から値の取得をおこなう手順を説明します。

```
[armadillo ~]# cat /sys/bus/w1/devices/28-000002219791/w1_slave
ae 01 4b 46 7f ff 02 10 aa : crc=aa YES
ae 01 4b 46 7f ff 02 10 aa t=26875
[armadillo ~]# cat /sys/bus/w1/devices/28-000002219791/w1_slave
```

```
ad 01 4b 46 7f ff 03 10 ab : crc=ab YES
ad 01 4b 46 7f ff 03 10 ab t=26812
```

図 7.11 コマンド実行例

Hardware Monitoring デバイスとしてアクセスする場合は以下のようにします。温度(1/1000℃)が求められます。

```
[armadillo ~]# cat /sys/class/hwmon/hwmon0/temp1_input
26875
[armadillo ~]# cat /sys/class/hwmon/hwmon0/temp1_input
26812
```

図 7.12 コマンド実行例(Hardware Monitoring)

7.2.4. プロトコル

DS18B20 のファンクションコマンドには以下のものがあります。

- CONVERT T(0x44):このコマンドにより、温度変換がおこなわれます。変換結果は、DS18B20 の内蔵 2 バイトレジスタに格納されます。
- WRITE SCRATCHPAD(0x48):DS18B20 の内蔵メモリに書き込みをおこないます。書き込むデータは 3 バイト長で、T_H、T_L、Configuration Register の順番に送信します。
- READ SCRATCHPAD(0xbe):DS18B20 の内蔵メモリを読み出します。読み出すデータのバイト数は最大 9 バイトです。途中で、マスターからリセットパルスを送信することで、データの読み出しを中断できます。

DS18B20 内蔵レジスタは次のようになっています。

表 7.1 DS18B20 内蔵レジスタ

バイト	内容
0	Temperature Register LSB
1	Temperature Register MSB
2	T_H or User Byte 1
3	T_L or User Byte 2
4	Configuration Register
5	Reserved (0xff)
6	Reserved
7	Reserved (0x10)
8	CRC

DS18B20 の温度センサー分解能は、Configuration Register の 5 ビット目と 6 ビット目で決まります。それ以外の Configuration Register のビットは内部的に使用され、上書きすることはできません。

表 7.2 DS18B20 温度センサー分解能

BIT 6	BIT 5	分解能
0	0	9 ビット
0	1	10 ビット
1	0	11 ビット
1	1	12 ビット(デフォルト)

Temperature Register のフォーマットは次のようになっています。温度は摂氏で格納されています。12 ビット分解能の場合は、BIT10〜BIT0 全てのビットが有効です。11 ビット分解能の場合、BIT0 が不定となります。10 ビット、9 ビット分解能の場合も同様です。BIT15〜BIT11 は、温度が正の場合 0、負の場合 1 となります。

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
LS_BYTE	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

	BIT15	BIT14	BIT13	BIT12	BIT11	BIT10	BIT9	BIT8
MS_BYTE	S	S	S	S	S	2^6	2^5	2^4

S = SIGN

図 7.13 DS18B20 Temperature Register フォーマット

8. CAN の活用

8.1. CAN とは

CAN は「Controller Area Network」の略で、主に自動車や船舶で機器間の通信に用いられる通信方式です。

一般的な接続方法を下図に示します。この図ではバスの終端回路は省略しています。なお、Armadillo-640 は、CAN でほかのノードと通信できます。

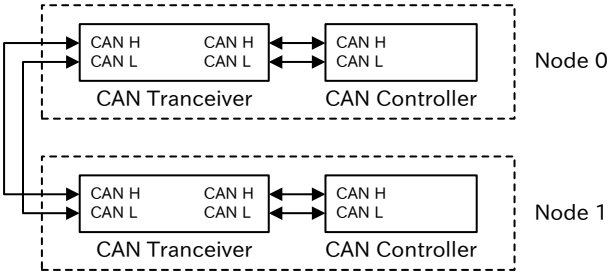

図 8.1 CAN の接続方法

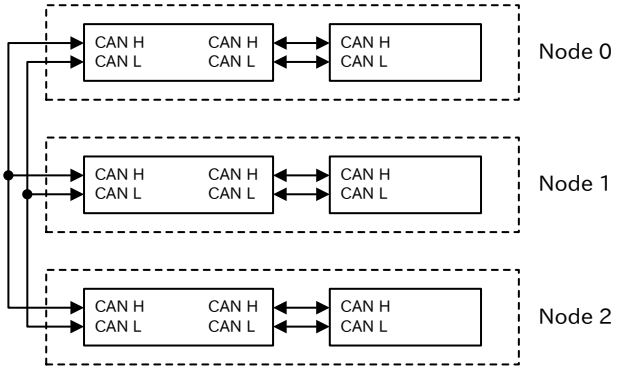

図 8.2 3 つ以上のノードの接続方法

CAN の主な特徴を以下に示します。

- 各ノードは対等
- 3 つ以上のノードを接続可能
- 差動電圧方式による高いノイズ耐性
- エラー処理等が規定されており、高信頼性
- 最高 125kbps(ISO11898-2)
- 最高 1Mbps(ISO1159-2)

CAN の規格は、通信速度が 125kbps までの低速 CAN(ISO1159-2)、通信速度 1Mbps までの高速 CAN(ISO11898-2)が代表的です。

一般的に CAN のノードは、SoC の外部の CAN トランシーバ[1]を通してバスに接続します。

CAN の信号は CAN+と CAN-の電位差で表します。ISO11898-2 では、信号線の電位差がある状態(CAN+ > CAN-)がドミナント、ない状態がリセッシブとなります。ドミナントを論理 0、リセッシブを論理 1 として扱うことが一般的です。

CAN はクロックを用いないため、あらかじめ全ノードに同じボーレート[2]を設定しなければいけません。

[1]CAN の差動信号と、SoC 等が扱いやすいシングルエンド信号を相互に変換します。
[2]デジタルデータを変調・復調する速さ

データの転送は、フレームという単位でおこないます。フレームには 4 つの種類があります。

表 8.1 CAN プロトコルフレーム

名称	機能
データフレーム	データを送信する。
リモートフレーム	データフレームを要求する。
オーバーロードフレーム	前回のフレーム処理が完了していないことを通知する。
エラーフレーム	エラーが発生したことを通知する。

データフレームとリモートフレームを合わせて、メッセージフレームといいます。CAN では、ノードごとのアドレスを持たず、メッセージが固有な ID(識別子、Identifier)を持っています。

CAN は全ノードが送信を開始できるため、複数のノードが同時に送信開始し、衝突が起こることがあります。衝突した場合はドミナントが優先され、小さいメッセージ ID のメッセージの送信が継続します。大きいメッセージ ID のメッセージを出力していたノードは送信エラーとなります。

受信ノードは、ID によって、自分が処理すべきメッセージかどうか判断します。メッセージに含まれる ID の長さによって、メッセージフレームには標準フォーマット(11 ビット長)と拡張フォーマット(29 ビット長)の 2 種類の形式があります。

データフレームの形式を「図 8.3. CAN プロトコル(データフレーム)」に示します。上の線はリセッシブを、下の線はドミナントを意味します。データフレームは、データを送信するノードがバスをドミナントにすることから始まります。これをスタート・オブ・フレーム(SOF)と呼びます。SOF に続き、アービトレーションフィールド(ARBI)、コントロールフィールド(CONT)、データフィールド(DATA)、CRC フィールド(CRC)が順に送信されます。続いて、受信ノードは ACK フィールド(ACK)を送信します。最後に、7 ビット以上リセッシブに保ちエンド・オブ・フレーム(EOF)とします。

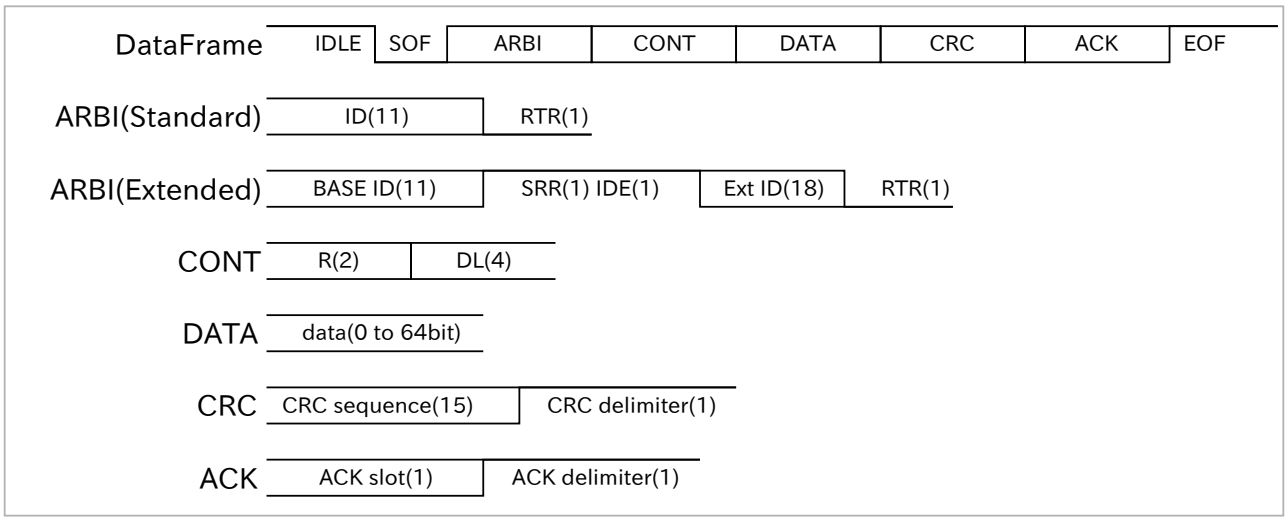

図 8.3 CAN プロトコル(データフレーム)

アービトレーションフィールドは、標準フォーマットか拡張フォーマットかによって異なります。標準フォーマットの場合、11 ビットの ID を送信したあと、リモート・トランスミッション・リクエスト・ビット(RTR)にドミナントを送信します。拡張フォーマットの場合、11 ビットのベース ID(BASE ID)を送信したあと、代替リモート・リクエスト・ビット(SRR)、アイデンティファイヤ・エクステンション・ビット(IDE)として、2 ビット分バスをリセッシブに保ちます。続いて、18 ビットの拡張 ID(Ext ID)を送信したあと、RTR にドミナントを送信します。

CAN プロトコルのバリエーション

CAN プロトコルには、バージョン 2.0A と 2.0B の 2 つがあります。プロトコルバージョン 2.0A では、標準フォーマットしか扱うことができません。もし拡張フォーマットのフレームを受信した場合、エラーフレームを送信します。プロトコルバージョン 2.0B パッシブでは、拡張フォーマットの送信はできませんが、拡張フォーマットを受信しても、無視します。プロトコルバージョン 2.0B アクティブでは、拡張フォーマットの送受信が可能です。

Armadillo-640 は、プロトコルバージョン 2.0B アクティブに対応しています。

コントロールフィールドは、最初の 2 ビットが予約ビットとなっており、常にドミナントとします。続く 4 ビットのデータ長コード(DLC)に送信するデータのバイト数を送信します。そのため、データフィールドは 0〜8 バイト(64 ビット)長となります。

CRC フィールドの CRC シーケンス(CRC sequence)には、SOF からデータフィールドまでの CRC(Cyclic Redundancy Check)を送信します。CRC フィールドの区切りを示す CRC デリミタとして、1 ビット分リセッシブとします。

受信ノードは、受信したメッセージの CRC が一致した場合、ACK スロット(ACK slot)でドミナントを送信します。ACK スロットでバスがドミナントとなることで、送信ノードは少なくとも一つの受信ノードがデータフィールドを正常に受信できたことを確認できます。ACK スロットに続いて、ACK フィールドの区切りを示す ACK デリミタ(ACK delimiter)として、1 ビット分リセッシブとします。

リモートフレームは、データフレームの要求に使用されます。リモートフレームを受信したノードは、リモートフレームで指定された ID と同じ ID のメッセージを返信します。リモートフレームの形式を、「図 8.4. CAN プロトコル(リモートフレーム)」に示します。RTR をリセッシブにして、データフィールドが無い以外、データフレームと同じです。DLC は、リモートフレームへの返信として帰ってくるデータフレームのデータ長と同一にします。

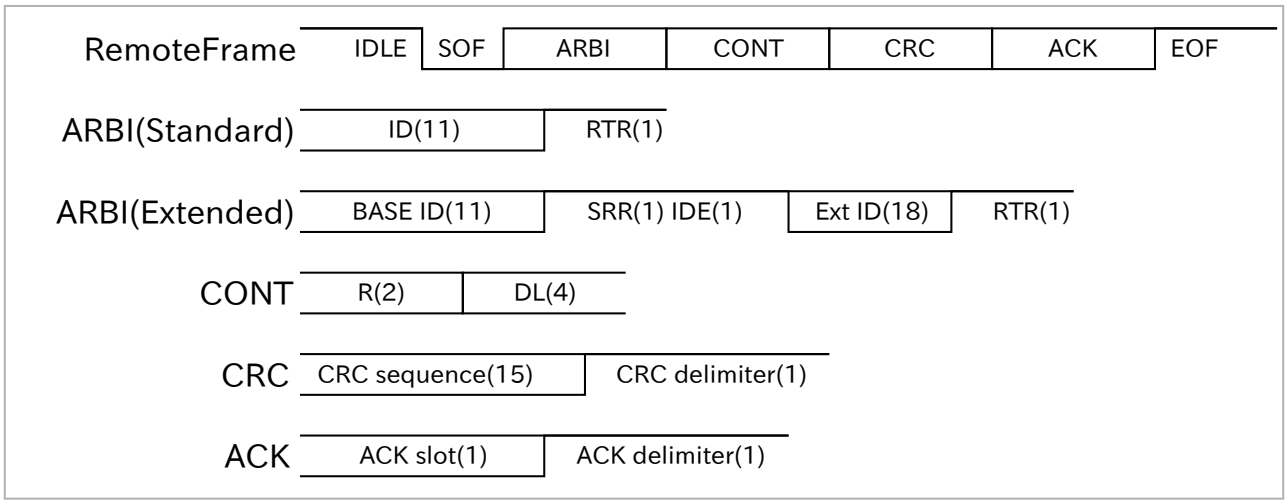

図 8.4 CAN プロトコル(リモートフレーム)

メッセージフレームのアービトレーションフィールドという名前は、このフィールド送信中にバスの調停をおこなうことに由来します。同時に複数のノードがメッセージの送信を開始した場合、バスの衝

突が発生します。送信ノードは、各ビットでバスの状態を確認し、もし自身がリセッシブを送信したにも関わらず、バスがドミナントとなっていた場合、以後の送信を中止します。そのため、より小さな ID が優先して送信されます。また、RTR により、リモートフレームよりもデータフレームが優先されます。

オーバーロードフレームとエラーフレームは、一般に CAN コントローラによって自動で処理されます。そのため、ここでは説明を割愛します。

同期とビット・スタッフィング・ルール

CAN では、ビットレートに従ってデータの送受信をおこなうため、ノードごとのクロックに誤差がある場合、タイミングが少しずつずれていきます。これを補正するため、バスがリセッシブからドミナントへ変化するとき、タイミングの同期をおこないます。

しかし、リセッシブやドミナントだけが続いた場合、この同期が行われないことになります。そこで、ビット・スタッフィング・ルールが適用されます。これは、同じ状態が 5 ビット連続した場合、反対の状態のビット(スタッフビット)を一つ送信するルールです。このルールにより、一定期間内に必ず同期が行われることを保証しています。

なお、ビット・スタッフィング・ルールの処理は CAN コントローラで自動で行われるため、ユーザー側は通常それを意識することはありません。

8.2. CAN を使用する

Armadillo-640 では、CON9 を CAN バスとして使用することができます。ここでは、Armadillo-640 同士を CAN で接続する方法を紹介します。

Armadillo-640 に、CAN トランシーバー回路を接続したものを 2 つ用意します。

USB シリアル変換アダプタ(SA-SCUSB-10)を使用する場合、使用ピンの重複があるため、「4.2. コンソールに使用する UART を変更する」に従ってコンソールに使用する UART を変更してください。

8.2.1. 接続方法

Armadillo-640 と CAN トランシーバーを接続する回路図を示します。Armadillo の CON9 から出ている CAN2 を使用します。CAN トランシーバーには、AMIS-42673 を使用します。5V 電源が必要となりますが、Armadillo の CON9 には 5V 電源が配線されていません。そのため、TPS60130 を用いて昇圧します。

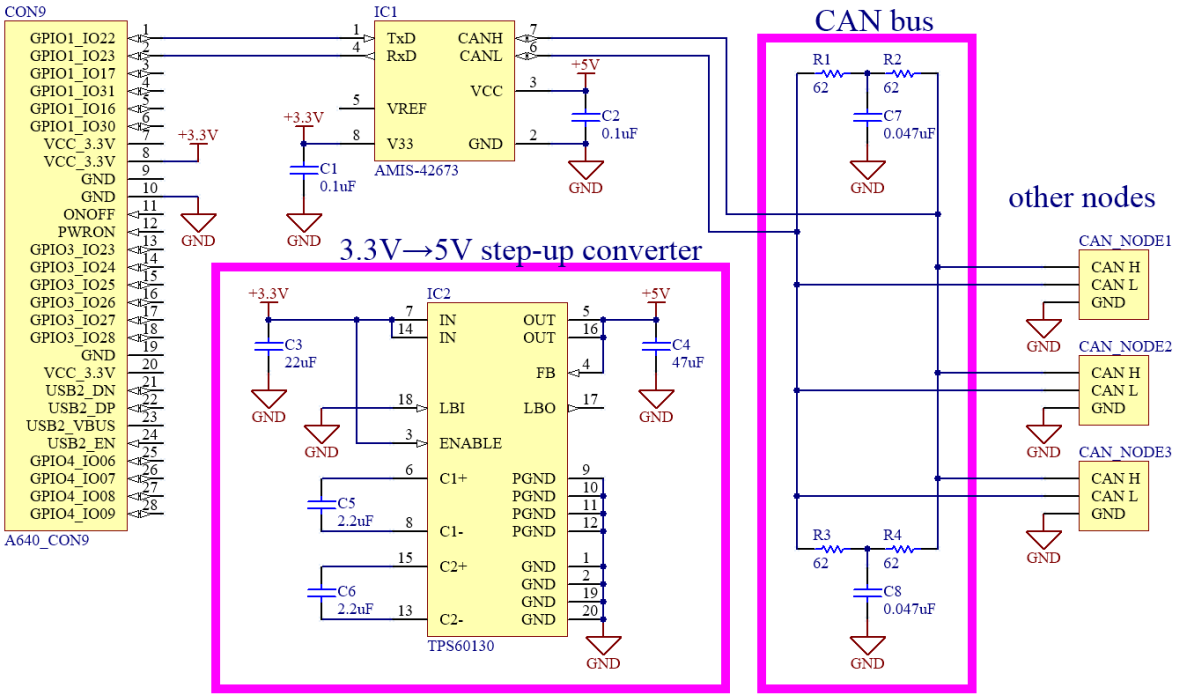

図 8.5 CAN トランシーバー回路図

Armadillo-640 同士を CAN で接続するので、同じ回路をもう 1 つ用意する必要があります。接続は次の通りです。

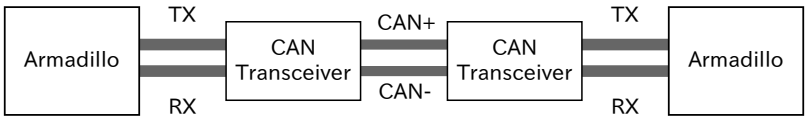

図 8.6 CAN 接続図

8.2.2. 対応カーネルイメージの作成

CAN のドライバを有効にした Linux カーネルと、DTB を作成します。

標準状態のカーネルのソースコードを元に変更する手順は次の通りです。

1. Device Tree の編集
2. カーネルコンフィギュレーションでの CAN の有効化
3. カーネルと DTB をビルドし Armadillo に書き込み

はじめに Device Tree を作成します。ファイル名は arch/arm/boot/dts/armadillo-640-can2.dtsi です(「1.3. サンプルソースコード」のページからダウンロードできます)。

```
&iomuxc {
        pinctrl_can2: can2grp {
                fsl,pins = <
                        MX6UL_PAD_UART2_CTS_B__FLEXCAN2_TX 0x0b0b0
                        MX6UL_PAD_UART2_RTS_B__FLEXCAN2_RX 0x0b0b0
```

```
                >;
        };
};

&can2 {
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_can2>;
        status = "okay";
};
```

図 8.7 armadillo-640-can2.dtsi

さきほどのファイルへの include を arch/arm/boot/dts/armadillo-640.dts に追加してください。

```
#include "armadillo-640-lcd70ext-l00.dtsi"
#endif
#include "armadillo-640-can2.dtsi"        //追加行

/ {
        model = "Atmark Techno Armadillo-640";
```

図 8.8 armadillo-640.dts に include を追記

続いて「イメージをカスタマイズする」と同様に menuconfig を使用してカーネルコンフィギュレーションを変更します。

```
[ATDE ~/linux-4.14-at[version]]$ make ARCH=arm menuconfig
```

図 8.9 menuconfig の実行

```
[*] Networking support  --->
  [*]   CAN bus subsystem support  --->                ← 有効にする
    CAN Device Drivers  --->
      [*] Platform CAN drivers with Netlink support
      [*]   Support for Freescale FLEXCAN based chips   ← 有効にする
```

図 8.10 menuconfig

変更を加え、カーネルコンフィギュレーションを確定してください。

以上の変更後、「イメージをカスタマイズする」と同様にカーネルと DTB をビルドし、Armadillo に書き込んでください。

8.2.3. CAN 通信プログラムの準備

CAN 通信プログラムのサンプルとして can-utils を使用します。can-utils には、一つのメッセージを送信する cansend、複数のメッセージを連続して送信する cangen、受信したメッセージを表示する candump があります。

can-utils をインストールします。

```
[armadillo ~]# sudo apt install can-utils
```

図 8.11 can-utils のインストール

8.2.4. 使用例

実際に、CAN バスを通じて Armadillo 同士で通信をおこなう手順を説明します。

まず、通信速度を設定します。通信速度は送受信をおこなうノード全てで一致している必要があるので、それぞれの Armadillo でおこなってください。

通信速度は ip コマンドで設定します。以下の例では通信速度を 125kbps に設定しています。

```
[armadillo ~]# ip link set can0 type can bitrate 125000 loopback off
```

図 8.12 CAN 信速度設定

次に、CAN インターフェースを有効にします。これも、それぞれの Armadillo で実行します。

```
[armadillo ~]# ifconfig can0 up
```

図 8.13 CAN を有効化

CAN メッセージを受信する Armadillo で、candump を実行しておきます。

```
[armadillo ~]# candump can0
```

図 8.14 candump 実行開始

別の Armadillo で cansend を実行すると、一つのメッセージを送信できます。下記の例では、ID=0x5a5、データ=0x01234567 を送信しています。

```
[armadillo ~]# cansend can0 5a5#01234567
```

図 8.15 cansend でのメッセージ送信

candump を実行している受信側の Armadillo では、下記のような受信結果が得られます。

```
[armadillo ~]# candump can0
  can0  5a5   [4]  01 23 45 67
```

図 8.16 candump でのメッセージ受信結果

また、cangen を実行すると、連続したメッセージを送信できます。オプションに CAN インターフェース名だけを指定した場合、cangen はアドレス、データ共にランダムな値を送信します。

```
[armadillo ~]# cangen can0
```

図 8.17 cangen での連続メッセージ送信

candump を実行している受信側の Armadillo では、下記のような受信結果が得られます。

```
[armadillo ~]# candump can0
  can0  567   [6]  69 98 3C 64 73 48
  can0  451   [8]  4A 94 E8 2A EC 58 55 62
  can0  729   [8]  BA 58 1B 3D AB D7 7E 50
  can0  1F2   [8]  E3 A9 E2 79 46 E1 45 75
  can0  07C   [2]  54 08
  :
  :
  :
```

図 8.18 candump での連続メッセージ受信結果

付録 A Linux カーネルサポートデバイス情報

Linux カーネルがサポートしているデバイス情報を紹介します。本章で紹介するのは、次のサブシステムに含まれるデバイスです。

- iio(Industrial I/O)
- hwmon(Hardware Monitoring)

それぞれのデバイスの追加方法については、表中に示す Device Tree 資料と、カーネルコンフィギュレーションのシンボル名を参考にしてください。Device Tree 資料の場所は、Linux カーネルのソースコードの Documentation/devicetree/bindings/ からの相対パスで記載します。

A.1. 加速度センサー

加速度を測定し、機器の向きによって画面表示を回転させたり、衝撃や落下などの異常検知に使用できます。

表 A.1 Linux カーネルがサポートする加速度センサー

メーカー	型番	接続	資料	シンボル
Analog Devices	ADXL345	I2C	iio/accel/adxl345.txt	ADXL345_I2C
		SPI		ADXL345_SPI
Bosch Sensortec	BMA180	I2C	iio/accel/bma180.txt	BMA180
	BMA250			
Domintech Technology	DMARD05	I2C	iio/accel/dmard06.txt	DMARD06
	DMARD06			
	DMARD07			
Kionix	KXSD9	I2C	iio/accel/kionix,kxsd9.txt	KXSD9_I2C
STMicroelectronics	LIS302	I2C	iio/accel/lis302.txt	IIO_ST_ACCEL_I2C_3AXIS
		SPI		IIO_ST_ACCEL_SPI_3AXIS
NXP Semiconductors	MMA8451Q	I2C	iio/accel/mma8452.txt	MMA8452
	MMA8452Q			
	MMA8453Q			
	MMA8652FC			
	MMA8653FC			
	FXLS8471Q			

A.2. A/D コンバーター

可変抵抗や、アナログセンサーの値を測定できます。

表 A.2 Linux カーネルがサポートする A/D コンバーター

メーカー	型番	接続	資料	シンボル
Holt Integrated Circuits	HI-8435	SPI	iio/adc/hi8435.txt	HI8435

メーカー	型番	接続	資料	シンボル
Maxim Integrated	MAX1027	SPI	iio/adc/max1027-adc.txt	MAX1027
	MAX1029			
	MAX1031			
	MAX11100	SPI	iio/adc/max11100.txt	MAX11100
	MAX1117	SPI	iio/adc/max1118.txt	MAX1118
	MAX1118			
	MAX1119			
Maxim Integrated	MAX1361	I2C	iio/adc/max1363.txt	MAX1363
	MAX1362			
	MAX1363			
	MAX1364			
	MAX1036			
	MAX1037			
	MAX1038			
	MAX1039			
	MAX1136			
	MAX1136			
	MAX1137			
	MAX1138			
	MAX1139			
	MAX1236			
	MAX1237			
	MAX11238			
	MAX1239			
	MAX11600			
	MAX11601			
	MAX11602			
	MAX11603			
	MAX11604			
	MAX11605			
	MAX11606			
	MAX11607			
	MAX11608			
	MAX11609			
	MAX11610			
	MAX11611			
	MAX11612			
	MAX11613			
	MAX11614			
	MAX11615			
	MAX11616			
	MAX11617			
	MAX11644			
	MAX11645			
	MAX11646			
	MAX11647			
	MAX9611	I2C	iio/adc/max9611.txt	MAX9611
	MAX9612			

メーカー	型番	接続	資料	シンボル
Microchip Technology	MCP3001	SPI	iio/adc/mcp320x.txt	MCP320X
	MCP3002			
	MCP3004			
	MCP3008			
	MCP3201			
	MCP3202			
	MCP3204			
	MCP3208			
	MCP3301			
	MCP3421	I2C	iio/adc/mcp3422.txt	MCP3422
	MCP3422			
	MCP3423			
	MCP3424			
	MCP3425			
	MCP3426			
	MCP3427			
	MCP3428			
	MCP3021	I2C	hwmon/mcp3021.txt	SENSORS_MCP3021
	MCP3221			
Nuvoton Technology	NAU7802	I2C	iio/adc/nuvoton-nau7802.txt	NAU7802

メーカー	型番	接続	資料	シンボル
Texas Instruments	ADC0831	SPI	iio/adc/ti-adc0832.txt	TI_ADC0832
	ADC0832			
	ADC0834			
	ADC0838			
	ADC084S021	SPI	iio/adc/ti-adc084s021.txt	TI_ADC084S021
	ADC108S102	SPI	iio/adc/ti-adc108s102.txt	TI_ADC108S102
	ADC128S102			
	ADC12130	SPI	iio/adc/ti-adc12138.txt	TI_ADC12138
	ADC12132			
	ADC12138			
	ADC128S052	SPI	iio/adc/ti-adc128s052.txt	TI_ADC128S052
	ADC122S021			
	ADC124S021			
	ADC141S626	SPI	iio/adc/ti-adc161s626.txt	TI_ADC161S626
	ADC161S626			
	ADS7950	SPI	iio/adc/ti-ads7950.txt	TI_ADS7950
	ADS7951			
	ADS7952			
	ADS7953			
	ADS7954			
	ADS7955			
	ADS7956			
	ADS7957			
	ADS7958			
	ADS7959			
	ADS7960			
	ADS7961			
	ADS8684	SPI	iio/adc/ti-ads8688.txt	TI_ADS8688
	ADS8688			
	ADC128D818	I2C	hwmon/adc128d818.txt	SENSORS_ADC128D818
	ADS1015	I2C	hwmon/ads1015.txt	SENSORS_ADS1015
	ADS1115			
	ADS7828	I2C	hwmon/ads7828.txt	SENSORS_ADS7828
	ADS7830			

A.3. 化学センサー

液体の pH(水素イオン指数)を測定できます。

表 A.3 Linux カーネルがサポートする化学センサー

メーカー	型番	接続	資料	シンボル
Atlas Scientific	EC-SM OEM	I^2C	iio/chemical/atlas,ec-sm.txt	ATLAS_PH_SENSOR
	ORP-SM OEM			
	pH-SM OEM			

A.4. D/A コンバーター

電圧(もしくは電流)のアナログ信号の出力ができます。

表 A.4 Linux カーネルがサポートする D/A コンバーター

メーカー	型番	接続	資料	シンボル
Analog Devices	AD5592R	SPI	iio/dac/ad5592r.txt	AD5592R
	AD5593R	I2C		AD5593R
	AD5755	SPI	iio/dac/ad5755.txt	AD5755
	AD5755-1			
	AD5757			
	AD5735			
	AD5737			
	AD7303	SPI	iio/dac/ad7303.txt	AD7303
	LTC2632-L12	SPI	iio/dac/ltc2632.txt	LTC2632
	LTC2632-L10			
	LTC2632-L8			
	LTC2632-H12			
	LTC2632-H10			
	LTC2632-H8			
Maxim Integrated	MAX5821	I2C	iio/dac/max5821.txt	MAX5821
Microchip Technology	MCP4725	I2C	iio/dac/mcp4725.txt	MCP4725
	MCP4726			
Texas Instruments	DAC7512	SPI	iio/dac/ti,dac7512.txt	TI_DAC7512

A.5. 周波数シンセサイザ

表 A.5 Linux カーネルがサポートする周波数シンセサイザ

メーカー	型番	接続	資料	シンボル
Analog Devices	ADF4350	SPI	iio/frequency/adf4350.txt	ADF4350
	ADF4351			

A.6. ジャイロスコープ

機器の回転を測定できます。

表 A.6 Linux カーネルがサポートするジャイロスコープ

メーカー	型番	接続	資料	シンボル
InvenSense	MPU-3050	I2C	iio/gyroscope/invensense,mpu3050.txt	MPU3050_I2C

A.7. ヘルスセンサー

脈拍等を測定できます。

表 A.7 Linux カーネルがサポートするヘルスセンサー

メーカー	型番	接続	資料	シンボル
Texas Instruments	AFE4403	SPI	iio/health/afe4403.txt	AFE4403
	AFE4404	I2C	iio/health/afe4404.txt	AFE4404
Maxim Integrated	MAX30100	I2C	iio/health/max30100.txt	MAX30100
	MAX30102	I2C	iio/health/max30102.txt	MAX30102

A.8. 湿度センサー

湿度と温度を測定できます。

表 A.8 Linux カーネルがサポートする湿度センサー

メーカー	型番	接続	資料	シンボル
Umemoto LLC	DHT11	GPIO	iio/humidity/dht11.txt	DHT11
	DHT22			
Texas Instruments	HDC1000	I2C	iio/humidity/hdc100x.txt	HDC100X
	HDC1008			
	HDC1010			
	HDC1050			
	HDC1080			
STMicroelectronics	HTS221	I2C	iio/humidity/hts221.txt	HTS221_I2C
		SPI		HTS221_SPI
Measurement Specialties	HTU21	I2C	iio/humidity/htu21.txt	HTU21

A.9. 慣性測定センサー

加速度センサーとジャイロセンサーが一体化したデバイスです。さらに地磁気センサーを内蔵しているデバイスもあります。

表 A.9 Linux カーネルがサポートする慣性測定センサー

メーカー	型番	接続	資料	シンボル
Bosch Sensortec	BMI160	I2C	iio/imu/bmi160.txt	BMI160_I2C
		SPI		BMI160_SPI
InvenSense	MPU6050	I2C	iio/imu/inv_mpu6050.txt	INV_MPU6050_IC2
		SPI		INV_MPU6050_SPI
	MPU6500	I2C		INV_MPU6050_IC2
		SPI		INV_MPU6050_SPI
	MPU9150	I2C		INV_MPU6050_IC2
		SPI		INV_MPU6050_SPI
	MPU9250	I2C		INV_MPU6050_IC2
		SPI		INV_MPU6050_SPI
	ICM20608	I2C		INV_MPU6050_IC2
		SPI		INV_MPU6050_SPI
STMicroelectronics	LSM6DS3	I2C	iio/imu/st_lsm6dsx.txt	IIO_ST_LSM6DSX_I2C
		SPI		IIO_ST_LSM6DSX_SPI
	LSM6DS3H	I2C		IIO_ST_LSM6DSX_I2C
		SPI		IIO_ST_LSM6DSX_SPI
	LSM6DSL	I2C		IIO_ST_LSM6DSX_I2C
		SPI		IIO_ST_LSM6DSX_SPI
	LSM6DSM	I2C		IIO_ST_LSM6DSX_I2C
		SPI		IIO_ST_LSM6DSX_SPI

A.10. 照度センサー

明るさを測定できます。距離や色などの測定ができるデバイスもあります。

表 A.10 Linux カーネルがサポートする照度センサー

メーカー	型番	接続	資料	シンボル
Broadcom	APDS9300	I2C	iio/light/apds9300.txt	APDS9300
	APDS9960	I2C	iio/light/apds9960.txt	APDS9960
Capella Microsystems	CM36651	I2C	iio/light/cm36651.txt	CM36651
Sharp Corporation	GP2AP020A00F	I2C	iio/light/gp2ap020a00f.txt	GP2AP020A00F

メーカー	型番	接続	資料	シンボル
Renesas Electronics	ISL29018	I2C	iio/light/isl29018.txt	SENSORS_ISL29018
	ISL29023			
	ISL29035			
Texas Instruments	OPT3001	I2C	iio/light/opt3001.txt	OPT3001
ams AG	TSL2560	I2C	iio/light/tsl2563.txt	SENSORS_TSL2563
	TSL2561			
	TSL2562			
	TSL2563			
	TSL2580	I2C	iio/light/tsl2583.txt	TSL2583
	TSL2581			
	TSL2583			
STMicroelectronics	VL6180	I2C	iio/light/vl6180.txt	VL6180

A.11. 磁力計センサー

地磁気から方角を測定することができます。

表 A.11 Linux カーネルがサポートする磁力計センサー

メーカー	型番	接続	資料	シンボル
Asahi Kasei Corporation	AK8974	I2C	iio/magnetometer/ak8974.txt	AK8974
	AK8975		iio/magnetometer/ak8975.txt	AK8975
Aichi Micro Intelligent Corporation	AMI305	I2C	iio/magnetometer/ak8974.txt	AK8974
	AMI306			
Bosch Sensortec	BMC156	I2C	iio/magnetometer/bmc150_magn.txt	BMC150_MAGN_I2C
	BMM150			
Honeywell International	HMC5843	I2C	iio/magnetometer/hmc5843.txt	SENSORS_HMC5843_I2C
	HMC5883			
	HMC5883L			

A.12. デジタルポテンショメーター

デジタル回路から抵抗値を調整できるデバイスです。可変抵抗のように、測定回路のキャリブレーションや音量調整等、アナログ回路の調整に使用できます。

表 A.12 Linux カーネルがサポートするデジタルポテンショメーター

メーカー	型番	接続	資料	シンボル
Maxim Integrated	DS1803-010	I2C	iio/potentiometer/ds1803.txt	DS1803
	DS1803-050			
	DS1803-100			
	MAX5481	SPI	iio/potentiometer/max5481.txt	MAX5481
	MAX5482			
	MAX5483			
	MAX5484			

メーカー	型番	接続	資料	シンボル
Microchip Technology	MCP4131	SPI	iio/potentiometer/mcp4131.txt	MCP4131
	MCP4132			
	MCP4141			
	MCP4142			
	MCP4151			
	MCP4152			
	MCP4161			
	MCP4162			
	MCP4231			
	MCP4232			
	MCP4241			
	MCP4242			
	MCP4251			
	MCP4252			
	MCP4261			
	MCP4262			

A.13. ポテンショスタット

液体の電気化学測定に用いる電源デバイスです。

表 A.13 Linux カーネルがサポートするポテンショスタット

メーカー	型番	接続	資料	シンボル
Texas Instruments	LMP91000	I2C	iio/potentiostat/lmp91000.txt	LMP91000

A.14. 気圧センサー

気圧を測定し、高度の推定等が行えます。

表 A.14 Linux カーネルがサポートする気圧センサー

メーカー	型番	接続	資料	シンボル
Bosch Sensortec	BMP085	I2C	iio/pressure/bmp085.txt	BMP280_I2C
		SPI		BMP280_SPI
	BMP180	I2C	iio/pressure/bmp085.txt	BMP280_I2C
		SPI		BMP280_SPI
	BMP280	I2C	iio/pressure/bmp085.txt	BMP280_I2C
		SPI		BMP280_SPI
	BME280	I2C	iio/pressure/bmp085.txt	BMP280_I2C
		SPI		BMP280_SPI
HOPE Microelectronics	HP03	I2C	iio/pressure/hp03.txt	HP03
Measurement Specialties	MS5611	I2C	iio/pressure/ms5611.txt	MS5611_I2C
		SPI		MS5611_SPI
	MS5607	I2C	iio/pressure/ms5611.txt	MS5611_I2C
		SPI		MS5611_SPI
	MS5637	I2C	iio/pressure/ms5637.txt	MS5637
	MS5805			
	MS5837			
	MS8607			
村田製作所	ZPA2326	I2C	iio/pressure/zpa2326.txt	ZPA2326_I2C

A.15. 距離センサー

雷センサー、超音波距離センサー、タッチセンサーを含みます。

表 A.15 Linux カーネルがサポートする距離センサー

メーカー	型番	接続	資料	シンボル
ams AG	AS3935	SPI	iio/proximity/as3935.txt	AS3935
Devantech	SRF04	GPIO	iio/proximity/devantech-srf04.txt	SRF04
Semtech	SX9500	I2C	iio/proximity/sx9500.txt	SX9500

A.16. 温度センサー

温度を測定できます。

表 A.16 Linux カーネルがサポートする温度センサー

メーカー	型番	接続	資料	シンボル
Maxim Integrated	MAX6675	SPI	iio/temperature/maxim_thermocouple.txt	MAXIM_THERMOCOUPLE
	MAX31855			
	MAX6646	I2C	hwmon/lm90.txt	SENSORS_LM90
	MAX6647			
	MAX6649			
	MAX6657			
	MAX6658			
	MAX6659			
	MAX6680			
	MAX6681			
	MAX6695			
	MAX6696			
	MAX6604	I2C	hwmon/jc42.txt	SENSORS_JC42
	MAX6581	I2C	hwmon/max6697.txt	SENSORS_MAX6697
	MAX6602			
	MAX6622			
	MAX6636			
	MAX6689			
	MAX6693			
	MAX6694			
	MAX6697			
	MAX6698			
	MAX6699			
Melexis	MLX90614	I2C	iio/temperature/mlx90614.txt	MLX90614
Texas Instruments	TMP007	I2C	iio/temperature/tmp007.txt	TMP007
	LM70	SPI	hwmon/lm70.txt	SENSORS_LM70
	TMP121			
	TMP122			
	LM71			
	LM74			
	LM87	I2C	hwmon/lm87.txt	SENSORS_LM87
	LM90	I2C	hwmon/lm90.txt	SENSORS_LM90
	LM86			
	LM89			
	LM99			
	TMP108	I2C	hwmon/tmp108.txt	SENSORS_TMP108

メーカー	型番	接続	資料	シンボル
Measurement Specialties	TSYS01	I2C	iio/temperature/tsys01.txt	TSYS01
Analog Devices	ADT7408	I2C	hwmon/jc42.txt	SENSORS_JC42
Microchip Technology	AT30TS00	I2C	hwmon/jc42.txt	SENSORS_JC42
	AT30TSE004			
	MCP9804			
	MCP9805			
	MCP9808			
	MCP98243			
	MCP98244			
	MCP9843			
ON Semiconductor	CAT6095	I2C	hwmon/jc42.txt	SENSORS_JC42
	CAT34TS02			
	ADM1024	I2C	hwmon/lm87.txt	SENSORS_LM87
	ADM1032	I2C	hwmon/lm90.txt	SENSORS_LM90
	ADT7461			
	ADT7461A			
	NCT1008			
NXP Semiconductors	SE97	I2C	hwmon/jc42.txt	SENSORS_JC42
	SE98			
	SA56004	I2C	hwmon/g762.txt	SENSORS_LM90
STMicroelectronics	STTS2002	I2C	hwmon/jc42.txt	SENSORS_JC42
	STTS2004			
	STTS3000			
	STTS424			
	STTS424E			
	STTS751	I2C	hwmon/stts751.txt	SENSORS_STTS751
Integrated Device Technology	TSE2002	I2C	hwmon/jc42.txt	SENSORS_JC42
	TSE2004			
	TS3000			
	TS3001			
Global Mixed-Mode Technology	G781	I2C	hwmon/g762.txt	SENSORS_LM90
Nuvoton	W83L771	I2C	hwmon/g762.txt	SENSORS_LM90
TDK Corporation	B57330V2103	ADC	hwmon/ntc_thermistor.txt	SENSORS_NTC_THERMISTOR
村田製作所	NCP15WB473	I2C	hwmon/ntc_thermistor.txt	SENSORS_NTC_THERMISTOR
	NCP18WB473			
	NCP21WB473			
	NCP03WB473			
	NCP15WL333			
	NCP03WF104			
	NCP15XH103			
Sensirion AG	SHT10	GPIO	hwmon/sht15.txt	SENSORS_SHT15
	SHT11			
	SHT15			
	SHT71			
	SHT75			

A.17. ファン速度コントローラ

ファン速度の制御や監視に使用できます。

表 A.17 Linux カーネルがサポートするファン速度コントローラ

メーカー	型番	接続	資料	シンボル
Global Mixed-Mode Technology	G762	I2C	hwmon/g762.txt	SENSORS_G762
	G763			
Maxim Integrated	MAX6650	I2C	hwmon/max6650.txt	SENSORS_MAX6650
	MAX6651			

A.18. 電源モニタ

電源の出力電流/電力などを監視できます。

表 A.18 Linux カーネルがサポートする電源モニタ

メーカー	型番	接続	資料	シンボル
Texas Instruments	INA209	I2C	hwmon/ina2xx.txt	SENSORS_INA2XX
	INA219			
	INA220			
	INA226			
	INA230			
	INA231			
Analog Devices	LTC2974	I2C	hwmon/ltc2978.txt	SENSORS_LTC2978
	LTC2975			
	LTC2977			
	LTC2978			
	LTC2980			
	LTC3880			
	LTC3882			
	LTC3883			
	LTC3886			
	LTC3887			
	LTM2987			
	LTM4675			
	LTM4676			
	LTC4151	I2C	hwmon/ltc4151.txt	SENSORS_LTC4151

Armadillo 標準ガイド ハードウェア拡張編

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2019/04/09	・初版発行

Armadillo 標準ガイド ハードウェア拡張編
Version 1.0.0
2019/04/09