参考資料

Application Note

外部 RAM による AM13x の AI 機能の拡張:ソフトウェアおよびハードウェアの設計手順

TEXAS INSTRUMENTS

概要

テキサス インスツルメンツは、さまざまな最終製品向けに、幅広いツール チェーンとマイコンを提供し、エッジ AI アプリケーションをサポートしています。急速に変化するエッジ AI 分野では、比較的「複雑な」エッジ AI モデルをマイコンに展開する際、メモリが障壁となるような事態は避けるべきです。このアプリケーション ノートでは、AM13x の外部ペリフェラル インターフェイス (EPI) を使用して AI モデルを外部 SDRAM にオフロードし、システム レベルで性能、コスト、消費電力のバランスを実現できるよう支援します。このアプリケーション ノートでは、これらのアプローチと手順をステップバイステップで説明します。デモ プロジェクトは、AM13E230x MCU_SDK の「am13e230x_sdk_26_01_00_00/examples/driverlib/epi/epi_sdram_read_write」に用意されており、SDK バージョン 26.01.00 でサポートされ、SDRAM AS4C32M16S [14] を使用してテストされています。

目次

1 概要.....2
1.1 TI AM13E230x の概要.....2
1.2 TI マイコン向け AI ツールチェーンの概要.....3
2 詳細説明.....5
2.1 EPI の SysConfig 設定.....5
2.2 外部 SRAM 実行用の AI モデル ライブラリの展開.....6
2.3 ハードウェア設計のベスト プラクティス.....9
3 まとめ.....11
4 参考資料.....12

商標

1 概要

TI は、AM13x、C2000、AM26x、MSPM0 などの各種 TI マイコン シリーズで AI アプリケーションを展開できるよう、包括的なツールチェーンを提供しています。システム レベルでは、AI モデルは低レイテンシかつ高精度で結果を生成する必要があります。または、レイテンシを犠牲にしても、より高い精度を維持する必要があります。これは、システム アーキテクトが一般的に直面するトレードオフです。

処理速度が重要となる場面では、AI モデルは「軽量」であり、必要なフラッシュ容量と SRAM 容量は数十キロバイト程度です。NPU エンジンは、AI モデルの処理を高速化し、結果をリアルタイムで出力できます。一方、モデル サイズが重要となる場合、モデルの展開はマイコンのオンチップ メモリによって制限されますが、CPU でもモデル処理には十分です。その制約により、ユーザーはプロセッサ レベルのデバイスを選択する必要があり、コストが高く消費電力が大きいため、アプリケーションの実現が困難になります。

TI AM13x マイコンは、外部ペリフェラル インターフェイス (EPI) を活用してこの課題を解決します。EPI は外部 SDRAM、ASRAM、SRAM、NOR フラッシュ メモリと接続できるため、AI モデルを外部 SDRAM に配置することができます。この機能により、AI モデルの実行用として最大 64MB の外部 SDRAM を利用できます。また、この構成によりシステム コストと消費電力も大幅に削減できます。

こうした利点がある一方で、この設計には 2 つの欠点があります。1 つ目はサイズです。EPI 接続には 32 ピンが必要です。32 ピン EPI 信号をサポートする AM13E230x の最小パッケージは LQFP100 です。このパッケージの外形は 15.5mm × 15.5mm であるため、SRAM サイズも考慮する必要があります。そのため、小型サイズが求められるアプリケーションでは、この設計は実用的ではなくなります。

2 つ目の欠点は速度です。AM13E230x のオンチップ NPU は、外部 SRAM へアクセスするようには設計されていないため、外部 SDRAM にオフロードされたモデルは、200MHz で動作する ARM® Cortex®-M33 によって実行される必要があります。命令アクセス速度も EPI バスによって制限されているため、超低レイテンシでリアルタイムに AI 出力を必要とするアプリケーションなど、高速が求められるアプリケーションでは実用的ではありません。

本資料で説明する手法は、エッジ AI アプリケーションだけを対象としたものではありません。同様のアプローチを拡張することで、AM13x マイコン上であらゆるアプリケーションを保存および実行できます。

1.1 TI AM13E230x の概要

AM13E230x マイコン (MCU) は、最大 200MHz の周波数で動作する Arm Cortex-M33 32 ビット CPU をベースとした、高集積、低コストの 32 ビット マイコン ファミリである AM13x シリーズの一部です。これらのリアルタイム制御向けに最適化されたマイコンは、高性能なアナログ、制御、およびデジタル ペリフェラルを統合しています。このマイコンは、誤り訂正コード (ECC) を備えた最大 512KB の組込みフラッシュ プログラム メモリ (最大 256KB × 2 バンク) と、ハードウェア パリティを備えた最大 128KB の SRAM を搭載しています。より小さなメモリ構成のバリアントも用意されています。

拡張通信インターフェイスは、1 つの MCAN と最大 6 つの UNICOMM ペリフェラルにサポートされ、UART/LIN、I2C/SMBUS、SPI の組み合わせに対応します。外部デバイスまたはメモリに接続するために、高速 EPI は SDRAM または FPGA や ASIC などの非同期 RAM デバイスに接続できます。

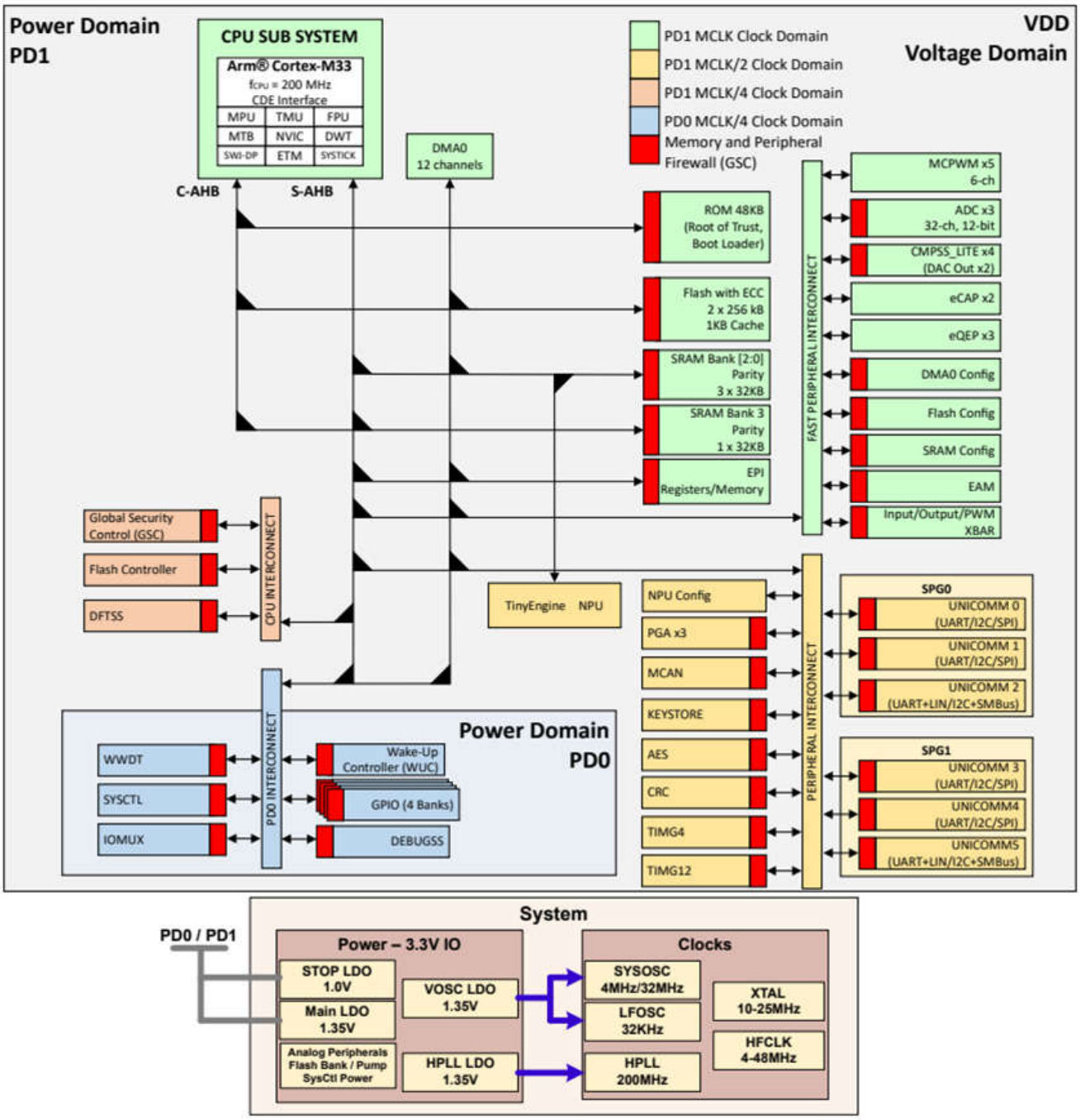

図 1-1. AM13E230x の機能ブロック図

1.2 TI マイコン向け AI ツールチェーンの概要

TI は、マイコン向けエッジ AI を支援する強力なツールチェーンを提供しています。さまざまな経験レベルのユーザーが、TI のツールチェーンを活用してマイコン上で AI モデルを構築および展開できます。データ アクイジションからデータ ラベル付け、AI モデルのトレーニング、コンパイル、展開まで、TI ツールチェーンはプロセス全体を管理します。

各種ツールのユーザー ガイドは、「参考資料」セクションに添付されています。

- git hub 内の tinyml-lensorlab リリース [1] と、TI.com の公式 Web サイト内の Edge AI Studio GUI リリース [2] には、TI による検証済みの AI モデル コンパイル用のモデルとツールチェーン一式が含まれます。ユーザーは、これらのモデルをアプリケーションに直接統合することができます。git hub リリースでは、.py ファイルを直接読み込んで、ツールチェーンがどのように動作するかを確認できます。これらのモデルはすべて速度を重視して開発されており、外部

SRAM を必要としません。ユーザーは TI の LaunchPad を使用して、モデルのトレーニングと検証を直接実行できます。これらのモデルをマイコンに展開する例は、各マイコン デバイスの SDK に収録されています。

- ユーザーが PyTorch などの一般的に使用されている AI モデル開発ツールを使用して独自の AI モデルを作成する場合、TI Neural Network Compiler for MCUs [3] により、.onnx ファイルをマイコン用の .a ライブラリ ファイルにコンパイルできます。この場合、マイコンのオンチップ NPU をアクセラレータとして使用する必要がある場合、モデルを NPU 対応の演算 [4] で構築し、NPU 上で動作可能にするために TI-NPU QAT [5] を .onnx ファイルに適用する必要があります。モデルを CPU コア上で実行する必要がある場合、量子化には DQD を使用するだけで十分です。ユーザー自身でモデルを慎重に検証してください。コンパイラは、各レイヤの入力と出力をユーザーがデバッグできるように、lib.c ファイルも出力します。これらの高度にカスタマイズされたモデルは、異なる AI 学習構造を持つため、大量のメモリを容易に消費する場合があります。一例として、トランスフォーマ構造を使用する AI モデルがあります。この場合、メモリ拡張のために外部 SDRAM が必要です。

どちらの方法でも、マイコン プロジェクトに組込む必要がある最終的な生成 AI モデルは、コードと重みを含む .a ライブラリ ファイルと、そのライブラリのインターフェイスを提供する「.h」ヘッダ ファイルになります。

2 詳細説明

このセクションでは、マイコンのオンチップ フラッシュに AI モデル ファイルをロードし、モデルの実行を外部 SDRAM にオフロードする方法をステップバイステップで説明します。

2.1 EPI の SysConfig 設定

このセクションでは、SysConfig ツールを使用して EPI ペリフェラルの初期化コードを生成します。EPI IP の詳細な仕様については、AM13E230 TRM [6] の EPI セクションを参照してください。生成されたコードはファイル <ti_sdk_dl_config.c> にあり、main 関数内の SYSCFG_DL_init 関数で呼び出されます。EPI の詳細については、「参考資料」セクションに添付されている「AM13x アカデミー」の EPI に関する章を参照してください。

設定対象となる主な仕様は、リフレッシュ カウンタと EPI クロック デバイダの 2 つです。

2.1.1 リフレッシュ カウント

リフレッシュ カウントは、外部クロック速度、バンクあたりの行数、およびリフレッシュ期間に基づきます。RFSH フィールドは、自動リフレッシュが必要になるまでに残っている外部クロック サイクルの数を表します。通常の式は次のとおりです。

$$\text{RFSH} \leq (\text{t\_Refresh\_us} \ / \ \text{number\_rows}) \ / \ \text{ext\_clock\_period\_us} \quad (1)$$

リフレッシュ期間は通常 64ms、つまり 64,000µs です。行の数は通常 4,096 または 8,192 です。ext_clock_period は µs で表される値であり、1,000 をクロック速度 (MHz) で除算することで求められます。したがって、50MHz の場合、1000/50 = 20ns、すなわち 0.02µs です。システム クロックが 50MHz で動作し、EPIBAUD レジスタ値が 0 の場合、標準的な SDRAM は 1 バンクあたり 4,096 行です。

$$\text{RFSH} = (64000 \ / \ 4096) \ / \ 0.02 = \ 15.625\text{us} \ / \ 0.02\text{us} = 781.25 \quad (2)$$

RFSH フィールドのデフォルト値は 10 進数で 750 (0x2EE) です。これは安全率を確保し、リフレッシュ間隔を 15µs とするためです。この数値は常に、式 2 で要求される値以下でなければならないことに注意してください。許容される値より大きい数値を使用すると、SDRAM のリフレッシュ頻度が不足し、データが失われます。

2.1.2 EPI クロック デバイダ

EPI コントローラ SDRAM インターフェイスは、50MHz まで動作できます。EPIBAUD レジスタの COUNT0 フィールドは、EPI クロックの速度を設定します。メイン クロック (MCLK) 速度が 50MHz までの場合、COUNT0 フィールドは 0x0000 にでき、SDRAM インターフェイスは MCLK と同じ速度で動作できます。ただし、MCLK がより高速で動作している場合、バス インターフェイスは最大でもその半分の速度でしか動作できません。そのため、COUNT0 フィールドは 0x0001 以上に設定する必要があります。設定については、図 2-1 を参照してください。

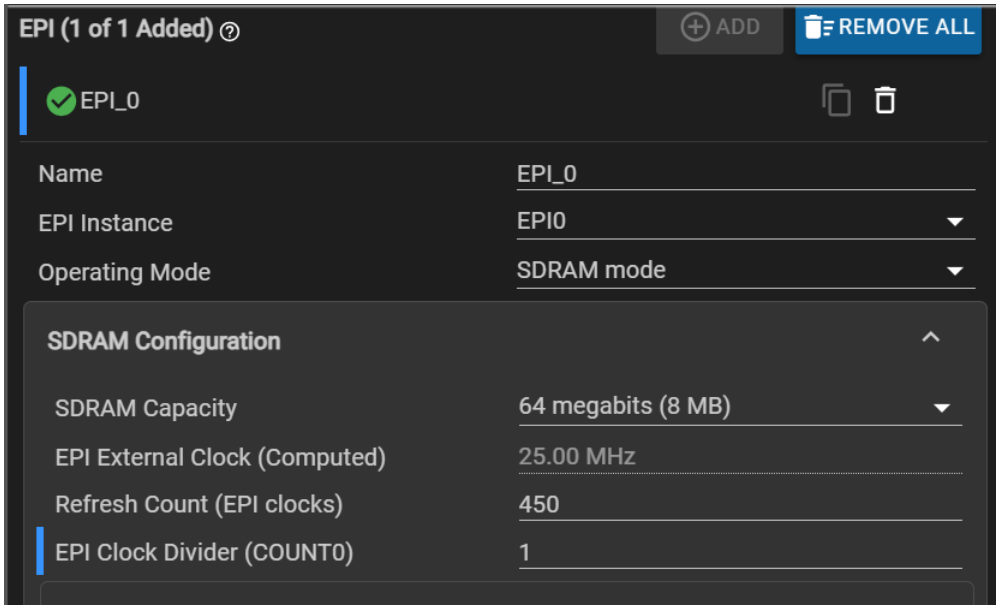

図 2-1. EPI の設定例 #1

EPI は外部 RAM のデータおよび命令用のアドレスを割り当てることができます。接続されているペリフェラルが存在しないため、ベース アドレスの設定は不要です。詳しくは、図 2-2 を参照してください。SDRAM モードでは、割り込みを設定する必要はありません。SysConfig ツールによって、ハードウェアのピン マルチプレクサが自動的に割り当てられます。

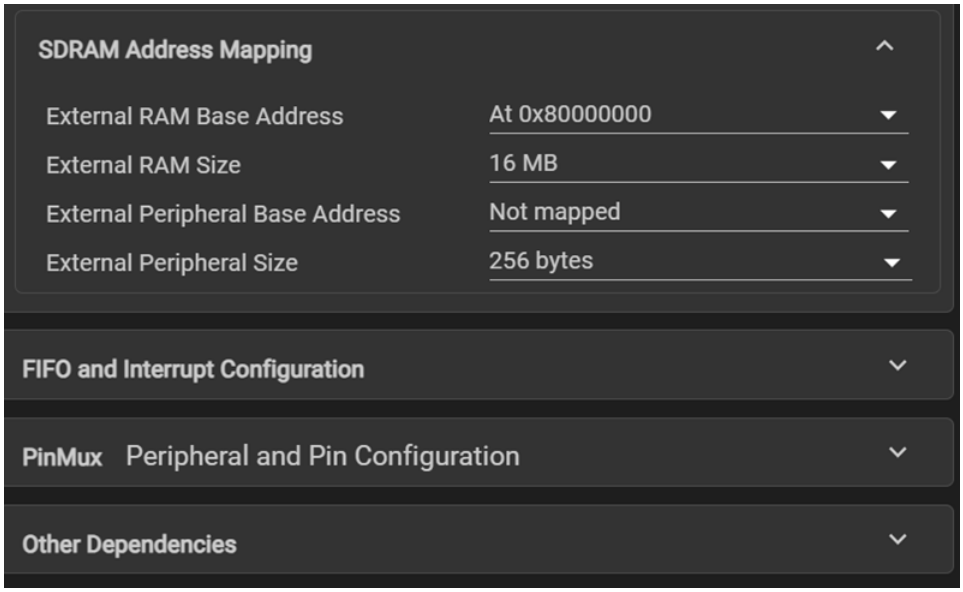

図 2-2. EPI の設定例 #2

2.2 外部 SRAM 実行用の AI モデル ライブラリの展開

AI アプリケーションでは、アテンション行列、重み行列などが、マイコン内の SRAM の大部分を占有します。これらの行列は、AI モデルの実行中のみ存在し、アプリケーション終了後には消滅します。この機能により、AI モデルに必要なフラッシュの容量は RAM 容量に比べて大幅に小さくなります。AI モデルをマイコンのオンチップ フラッシュに格納し、外部 SDRAM を使用して AI モデルを実行することで、AI モデルのメモリ要件を満たすことができます。このセクションでは、コンパイラのキーワードを使用して、このメモリ割り当てを実現する方法について説明します。

概要 で説明したように、AI モデルをプロジェクトに統合するために必要なファイルは、「.a」ライブラリです。このライブラリには、AI 関数用の .text セクションと、重みおよびテーブル用の .rodata セクションの 2 つのセクションが含まれていま

す。ユーザーは、リンカ セクションを使用して、これらのセクションをロード イメージとしてフラッシュに格納し、ブート時に SDRAM へコピーできます。すべての関数アドレスは、SDRAM アドレス空間 (0x80000000+) から割り当てられます。表 2-1 に、メモリ割り当てを示します。

表 2-1. フラッシュに AI モデルをロードし、外部 SDRAM で実行する場合のメモリ割り当て

フラッシュ (ロード イメージ格納)		SDRAM (ここからコードを実行)
0x000xxxxx	-> copyCode() ->	0x80000000 (syscfg で設定)
.text (AI 関数)		.text (PC はここからフェッチ)
.rodata (重みとテーブル)		.rodata

以下のセクションでは、目的とするメモリ割り当てを実現するための設定を、ステップバイステップで説明します。この例では、「.a」ファイル名として「customer_ai_model.a」を使用しています。ファイル名は必要に応じて変更できます。基本的な AI 組込みプロジェクトでは、「..\am13e230x_sdk_<version>\examples\ai\generic_timeseries_classification」[11] を参照しています。

コンパイラ シンボルの詳細と cmd ファイルの詳細については、「参考資料」セクションの <TI ARM CLANG コンパイラ ツール ユーザー ガイド > [8] および <TI リンカ コマンド ファイルの概要> [9] を参照してください。デモ プロジェクトは、AM13E230x MCU_SDK 内の「am13e230x_sdk_26_01_00_00/examples/driverlib/epi/epi_sdram_read_write」にあります。

2.2.1 ステップ 1: CCS プロジェクトに AI「.a」ファイルを追加する

`customer_ai_model.a` をプロジェクト フォルダにコピーし、「libCMSISDSP_m33_ti_arm_clang.a」ファイルを置き換えます。

CCS で次を開きます。**「Project Properties」(プロジェクト プロパティ) ->「Build」(ビルド) ->「Arm Linker」(Arm リンカ) ->「File Search Path」(ファイル検索パス)**

- ライブラリ ファイル名を追加します:「customer_ai_model.a」
- 「.a」ファイルを含むフォルダ パスをライブラリ検索パスに追加します

2.2.2 ステップ 2: リンカ スクリプトの変更 (linker_m33_ti_arm_clang.cmd)

MEMORY セクション「EPI_CS2_EXEC_1」をプロジェクトに追加します。お客様の AI モデルのサイズに合わせて、「length」の値を調整してください。CMD ファイルの変更内容については、以下を参照してください。

```
MEMORY
{
    RAM_S        : ORIGIN = 0x20000000, LENGTH = 0x18000   // 96KB internal SRAM
    RAM_C        : ORIGIN = 0x00C18000, LENGTH = 0x8000    // 32KB code SRAM
    FLASH_BANK0  : ORIGIN = 0x00000000, LENGTH = 0x40000   // 256KB Flash
    FLASH_BANK1  : ORIGIN = 0x00040000, LENGTH = 0x40000   // 256KB Flash

    // External SDRAM through EPI - AI model code + weights execute from here
    // Customer with 300KB AI model: increase to 0x0080000 (512KB) or 0x0100000 (1MB)
    EPI_CS2_EXEC_1(RWX) : origin = 0x80000000, length = 0x0010000  // 64KB (demo size)
}、、
```

.CS1_EXECUTE セクションを追加し、「libCMSISDSP_m33_ti_arm_clang.a」を「customer_ai_model.a」のファイル名に置き換えてください。CMD ファイルの変更内容については、以下を参照してください。

```
.CS1_EXECUTE : LOAD = FLASH_BANK0, palign(8),
               RUN  = EPI_CS2_EXEC_1,
               LOAD_START(emif1_cs2_sec1_loadstart),
               LOAD_SIZE(emif1_cs2_sec1_loadsize),
               RUN_START(emif1_cs2_sec1_runstart),
               RUN_SIZE(emif1_cs2_sec1_runsize)
{
    // Replace with the customer's AI .a filename.
    // The <*> wildcard grabs ALL objects in the archive automatically.
    customer_ai_model.a<*>(.text .text.* .rodata .rodata*)
}
```

各キーワードの意味については、表 2-2 を参照してください。

表 2-2. リンク コマンド ファイルの変更におけるキーワードの意味

キーワード	意味
LOAD = FLASH_BANK0`	リンク時にフラッシュに格納される AI モデルのバイナリ イメージ
RUN = EPI_CS2_EXEC_1	SDRAM (0x80000000+) から割り当てられるすべての AI 関数アドレス
LOAD_START (emif1_cs2_sec1_loadstart)	リンカ シンボル = イメージの開始位置を示すフラッシュ アドレス
LOAD_SIZE (emif1_cs2_sec1_loadsize)	リンカ シンボル = コピーするバイト数
RUN_START (emif1_cs2_sec1_runstart)	リンカ シンボル = SDRAM の配置先 (0x80000000)
<*> (.text .text.* .rodata .rodata*)	ワイルドカード:アーカイブからすべてのコードと重みをキャプチャ

2.2.3「main.c」の変更内容

このセクションでは、プロジェクトを展開するために必要な main.c ファイルの変更内容について説明します。

2.2.3.1 ヘッダのインクルード

以下のヘッダ ファイルをプロジェクトにインクルードする必要があります。

```
#include "device.h"
#include "log.h"
#include "stdint.h"
#include "dl_epi.h"
#include "dl_epi.c"           /* EPI driver – included directly              */
#include "basic_math_functions.h"  /* CMSIS DSP – defines float32_t           */
#include "ti_sdk_dl_config.h"
```

2.2.3.2 リンカ生成シンボルの宣言

これら 4 つのシンボルは、セクション属性に基づいてリンカが自動的に生成します。C 言語では、「&symbol」によって、リンカが割り当てた値を取得できます。

```
extern unsigned int emif1_cs2_sec1_loadstart;  // Flash address of load image start
extern unsigned int emif1_cs2_sec1_loadsize;   // Size of load image in bytes
extern unsigned int emif1_cs2_sec1_runstart;   // SDRAM destination (0x80000000)
extern unsigned int emif1_cs2_sec1_runsize;    // Size of section at run time
```

2.2.3.3 memcpy 関数

この関数は、TI リンカ シンボルの規約に従って、セクションをフラッシュから SDRAM へコピーします。「&sdram_code_loadstart」は、リンカが割り当てた値を示します。

```
memcpy(&sdram_code_runstart,&sdram_code_loadstart,(size_t)&sdram_code_loadsize);
```

2.2.3.4 Entry 関数を SDRAM セクションに配置

デモ プロジェクトでは、「ExecuteFromSDRAM()」が SDRAM 上で実行される関数であり、「arm_dot_prod_f32()」を呼び出します。この関数は、「__attribute__」によって「.CS1_EXECUTE」に配置されます。

```
/* This function executes from SDRAM. memcpy() must be called first. */
__attribute__((section(".CS1_EXECUTE"), __noinline__))
void ExecuteFromSDRAM(void)
{
    LOG("Executing from SDRAM!\r\n");

    /* Call arm_dot_prod_f32 – this function is also in .sdram_code (SDRAM).
     * Input data lives in RAM_S and is accessed through the normal data bus.  */
    arm_dot_prod_f32(App_DotProd_SrcA,
                     App_DotProd_SrcB,
                     APP_DOT_PROD_SIZE,
                     &App_DotProd_Result);
}
```

2.2.3.5 main() のブート シーケンス

これは、マイコンを適切にブートし、外部 SDRAM を使用して AI モデルの実行を開始するための main 関数シーケンスです。

```
int main(void)
{
    /* Step 1: Hardware init – Device_Init() sets up clocks and cache.
     * SYSCFG_DL_init() configures the EPI peripheral, making SDRAM
     * accessible at 0x80000000. Both must run before any SDRAM access. */
    Device_Init();
    SYSCFG_DL_init();

    /* Step 2: Copy .CS1_EXECUTE section from Flash to SDRAM.
     * Transfers all AI model functions and weights from their Flash
     * load addresses to their SDRAM run addresses (0x80000000+).
     * After this call, all AI functions execute from SDRAM. */
    memcpy(&sdram_code_runstart,&sdram_code_loadstart,(size_t)&sdram_code_loadsize);


    /* Step 3: Call AI inference – PC moves to 0x80000000+ region.
     * In the demo project this is executeFromCs0().
     * For customer: replace with the AI model inference entry point. */
    ExecuteFromSDRAM();/* customer replaces this with their AI inference call */


    /*Verify dot product result. */
    LOG_ASSERT((App_DotProd_Result == APP_DOT_PROD_EXPECTED),
               "arm_dot_prod_f32 result mismatch: expected 20.0\r\n");
    LOG("PASS: arm_dot_prod_f32 = %.1f – executed from SDRAM\r\n",
        App_DotProd_Result);

    LOG("Example PASS\r\n");

    return 0;
}
```

2.2.4 ステップ 4: CSS プロジェクト検証

お客様の AI「.a」ファイルに適用する前に、このアプローチを検証するため、CMSIS DSP ライブラリ (libCMSISDSP_m33_ti_arm_clang.a) を代わりに使用します。このライブラリには、アーカイブ内の「BasicMathFunctions.c.obj」に含まれる積和演算関数である「arm_dot_prod_f32()」が含まれています。

2.3 ハードウェア設計のベスト プラクティス

このセクションでは、EPI ハードウェア設計の一般的なガイドラインと、Altium を使用して要件をより適切に実現する方法について説明します。プロジェクト ファイルについては、TM4C EPI リファレンス デザイン ドキュメント [12] を参照してください。

AM13E230 ハードウェア設計ガイド [13] に、マイコン ハードウェアの設計に関する包括的なガイダンスを示しています。遵守すべき EPI ガイドラインは次のとおりです。

- ターゲット デバイス (メモリ IC または FPGA) をこのマイコンの近くに配置して、パターン長を短くします
- インピーダンスを制御するために、50Ω のシングルエンド パターンを使用します
- 同一レイヤ上で信号グループを配線し、ビアおよびレイヤ遷移を最小限に抑えることで、反射とクロストークを低減します
- 次のパターン長を一致させて、タイミング スキューを低減します。
  - データ バス:±25mil (0.64mm) 以内で揃えます (1mil は 1/1000 インチ)
  - アドレスおよびコマンド信号:±25mil (0.64mm) 以内で揃えます
  - 制御信号:±25mil (0.64mm) ～ ±50mil (1.27mm) 以内で揃えます
  - クロック信号:最長のデータ パターンに対して、±5mil (0.13mm) ～ ±10mil (0.25mm) 以内で揃えます
- アドレスおよびコマンド信号は、並列に配線します
- スタブは避け、配線はできるだけ直接的にします
- データ ラインとクロック ラインに直列終端抵抗 (22Ω ～ 33Ω) を入れ、トランスミッタの近くに配置することを検討してください。

図 2-3 に示すように、信号の反射を最小限に抑えるために、EPI ピンからアドレス / データまで、さらに他のピンまで単一のパターンで配線することを推奨します。

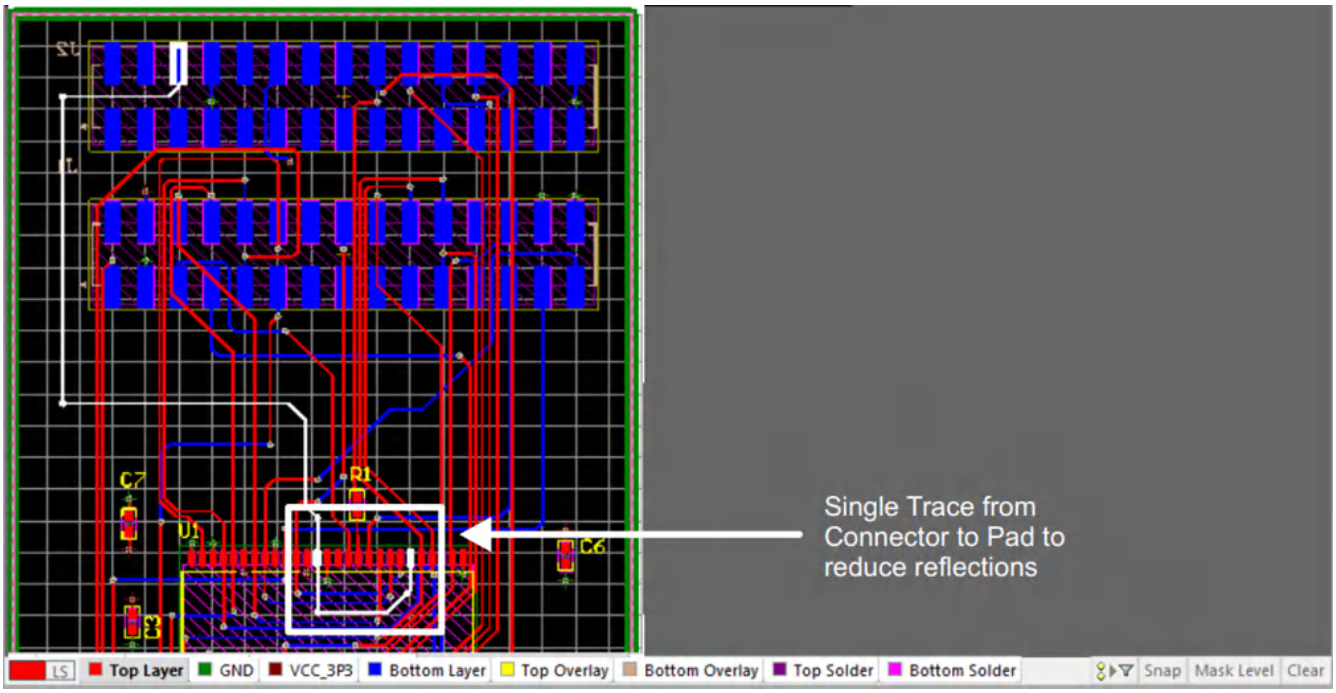

図 2-3. EPI ピンからアドレス / データへの単一パターン

Altium を使用してハードウェア設計を容易に進めるには:

- アドレス パターンとデータ パターンは、2 つの異なるネット名を使用する必要があります。これにより、長さ調整機能を利用してパターン長を揃えることができます。(抵抗の後段で) 両方のネット名が同じになっている場合、Altium で簡単に設定できなくなります。両方の間で異なるパターン名を使用することを推奨します。また、配線が AM13E23019 の EPI ピンに接続されている場合は、抵抗を使用してそれらを互いに短絡するか、ネット タイ部品を使用してピンでネットを互いに短絡することができます。

Altium で次の手順を使用することで、パターン マッチングを利用できます。

- 上部メニューから「Design」(設計) >「Rules」(ルール) に移動し、「PCB Rules and Constraints」(PCB ルールおよび制約) エディタを開きます。
- 「High-Speed」(高速) カテゴリを展開し、「Matched Lengths」(長さの一致) を選択します。
- 右クリックして「New Rule」(新しいルール) を選択し、ルールに名前を付けて構成します。
- 「Where the Object Matches」(オブジェクトが一致する場所) スコープで、一致対象となる特定のネット、xSignal、または Net Class を選択します。
- 「Constraints」(制約) セクションで、許容範囲を設定します。これにより、グループ内の最短パターンと最長パターンの差として許容される最大値が設定されます。
- ユーザーは、すべてのネットをこの許容差グループにまとめて、許容値を 25mil (0.64mm) に設定できます。

なお、対応する SDRAM ピンの近くに抵抗を配置し、すべてのピンと対応する抵抗間の距離を均一に保つことを推奨します。そうすることにより、その部分のパターンについて長さ調整をする必要がなくなります。

3 まとめ

このアプリケーション ノートでは、AM13E230 EPI を使用して AI モデルを外部 SRAM にオフロードする方法をご紹介します。この手法により、マイコン上で AI モデルを展開する際のメモリ不足問題を解決し、システム レベルで性能、コスト、消費電力の最適なバランスを確保できます。本資料では、アプローチとステップバイステップのガイダンスを説明します。また、デモ プロジェクトは AM13E230x MCU_SDK の「am13e230x_sdk_26_01_00_00/examples/driverlib/epi/epi_sdram_read_write」で利用できます。

4 参考資料

1. テキサス インスツルメンツ、『Tiny ML Tensorlab ユーザー ガイド』、ユーザー ガイド。
2. テキサス インスツルメンツ、『EDGE-AI-STUDIO』。
3. テキサス インスツルメンツ、『TI Neural Network Compiler for MCUs ユーザー ガイド - 2.1.2 LTS』、ユーザー ガイド。
4. テキサス インスツルメンツ、『NPU でサポートされるレイヤ構成』。
5. テキサス インスツルメンツ、『TI-NPU QAT』。
6. テキサス インスツルメンツ、『AM13E230x マイコン』、テクニカル リファレンス マニュアル。
7. Varahmihir, Arpit、『外部ペリフェラル インターフェイス (EPI)』。
8. テキサス インスツルメンツ、『SECTIONS ディレクティブ』。
9. テキサス インスツルメンツ、『TI リンカ コマンド ファイル概要』。
10. テキサス インスツルメンツ、『AM13E23019：EPI コード実行のデモ』。
11. テキサス インスツルメンツ、『AM13E2X-SDK』。
12. テキサス インスツルメンツ、『高性能マイコンでの SDRAM インターフェイス設計ガイド』、設計ガイド。
13. テキサス インスツルメンツ、『AM13E230x ハードウェア設計ガイドライン』、アプリケーション ノート。
14. Alliance Memory、『AS4C32M16SB データシート』、データシート。

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月