

Vitis HLS 移行ガイド

UG1391 (v2020.2) 2020 年 11 月 24 日

この資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。



改訂履歴

次の表に、この文書の改訂履歴を示します。

セクション	改訂内容
2020 年 11 月 24 日 バージョン 2020.2	
第 2 章: サポートされない機能	更新
2020 年 7 月 28 日 バージョン 2020.1	
資料全体	リリースに合わせた更新
2020 年 6 月 3 日 バージョン 2020.1	
初版。	なし

目次

改訂履歴.....	2
第 1 章: Vitis HLS への移行.....	4
HLS 動作の違い.....	4
第 2 章: サポートされない機能.....	16
最上位関数引数.....	16
HLS ビデオ ライブラリ.....	16
C の任意精度型.....	17
第 3 章: 廃止予定およびサポートされない Tcl コマンド オプション.....	18
付録 A: その他のリソースおよび法的通知.....	20
ザイリンクス リソース.....	20
Documentation Navigator およびデザイン ハブ.....	20
参考資料.....	20
お読みください: 重要な法的通知.....	21

Vitis HLS への移行

特定のバージョンの Vivado® HLS でインプリメントされたカーネル モジュールを移行する際は、HLS のバージョン間での違いと、それらの違いがデザインにどのように影響するかを理解しておくことが重要です。

主な考慮事項は、次のとおりです。

- 動作の違い
- サポートされない機能
- 廃止予定のコマンド

HLS 動作の違い

Vitis™ HLS では、C コードを合成する方法に基本的な変更が加えられており、言語コンストラクトおよび既存のコマンド、プラグマ、指示子がサポートされます。たとえば、`std::complex<long double>` 型は Vitis HLS でサポートされないため、使用しないでください。これらの変更は、アプリケーションの QoR (結果の品質) に影響します。ザイリンクスでは、ツールを使用する前に、このセクションに目を通すことをお勧めします。



ヒント: Vitis HLS と Vivado HLS の動作は異なるので、Vitis ツールで使用するコードは異なるものにすることが必要な場合があります。同じソース コードを両方のツールで使えるようにするため、Vitis HLS ではこのツール用に記述されたソース コードを囲む `__VITIS_HLS__` という定義済みのマクロがサポートされています。ツール特定のコードを囲むには、`#if defined(__VITIS_HLS__)` タイプのプリプロセッサ宣言を使用します。

デフォルトのユーザー制御設定

デフォルト グローバル オプションは、Vitis アプリケーション アクセラレーション開発フローまたは Vivado IP 開発フローのソリューションを設定します。

```
open_solution -flow_target [vitis | vivado]
```

このグローバル オプションは、以前のコンフィギュレーション オプション (`config_sdx`) に置き換わります。

Vivado フロー:

Vivado IP 生成フローで使用可能なソリューションを設定します。これには、プラグマおよび指示子を厳密に使用する必要があります。結果は Vivado IP としてエクスポートされます。

```
open_solution -flow_target vivado
```

表 1: デフォルト制御設定

デフォルト制御設定	Vivado HLS	Vitis HLS
<code>config_compile -pipeline_loops</code>	0	64
<code>config_export -vivado_optimization_level</code>	2	0
<code>set_clock_uncertainty</code>	12.5	27%
<code>config_export -vivado_optimization_level</code>	20	255
<code>config_interface -m_axi_alignment_byte_size</code>	なし	0
<code>config_interface -m_axi_max_widen_bitwidth</code>	なし	0
<code>config_export -vivado_phys_opt</code>	place	none
<code>config_interface -m_axi_addr64</code>	false	true
<code>config_schedule -enable_dsp_full_reg</code>	false	true
<code>config_rtl -module_auto_prefix</code>	false	true
インターフェイス プラグマのデフォルト	IP モード	IP モード

Vitis フロー (カーネル モード):

Vitis アプリケーション アクセラレーション開発フローで使用するためのソリューションを設定します。Vitis HLS では、INTERFACE プラグマや指示子を指定しなくても関数引数に対して正しくインターフェイスが推論され、合成した RTL コードが Vitis カーネル オブジェクト ファイル (.xo) として出力されます。

```
open_solution -flow_target vitis
```

表 2: デフォルト制御設定

デフォルト制御設定	Vivado HLS	Vitis HLS
インターフェイス プラグマのデフォルト	IP モード	カーネルモード (デフォルト インターフェイスをチェック)
<code>config_interface -m_axi_alignment_byte_size</code>	なし	64
<code>config_interface -m_axi_max_widen_bitwidth</code>	なし	512
<code>config_compile -name_max_length</code>	256	255
<code>config_compile -pipeline_loops</code>	64	64
<code>set_clock_uncertainty</code>	27%	27%
<code>config_rtl -register_reset_num</code>	3	3
<code>config_interface -m_axi_latency</code>	0	64

ループ II 制約のデフォルト設定

Vivado HLS では、ループ II 制約のデフォルトは 1 に設定されていましたが、Vitis HLS では auto に設定されます。ツールでデフォルトの II を達成できない場合は、可能な最良の II を達成することが試みられます。

デフォルト インターフェイス

インターフェイス合成で作成されるインターフェイスのタイプは、C 引数のデータ型、デフォルトのインターフェイスモード、およびインターフェイス指示子によって異なります。Vitis HLS では、デフォルトのインターフェイスは C 引数およびコンフィギュレーションで使用するデータ型によります。デフォルトのインターフェイス設定は、`open_solution -flow_target [vitis | vivado]` を使用してユーザーが選択します。

下の表に、デフォルトのインターフェイス プロトコルをイネーブルにした場合の変更を示します。

引数タイプの定義 (下の表で使用):

- I: 入力のみ (引数から読み出しのみ可能)
- O: 出力のみ (引数に書き込みのみ可能)
- IO: 入力および出力 (引数からの読み出しおよび引数への書き込みが可能)
- Return: データ出力を返す
- Block: ブロック レベルの制御
- D: 各タイプのデフォルト モード。

注記: 無効なインターフェイスを指定すると、Vitis HLS から警告メッセージが表示され、デフォルトのインターフェイスモードがインプリメントされます。

Vitis フロー (アクセラレータ モード)

Vitis フローで HLS を使用すると、次が自動的に設定されます。

```
open_solution -flow_target vitis
```

表 3: 引数のタイプ

引数のタイプ	スカラー		配列を指すポインター			hls::stream
インターフェイスモード	入力	戻り値	I	I/O	O	I および O
ap_ctrl_none						
ap_ctrl_hs						
ap_ctrl_chain		D				
axis						D
m_axi			D	D	D	

注記:

1. D = デフォルト インターフェイス

AXI4-Lite スレーブ インターフェイス指示子を指定すると、インターフェイス プラグマの動作は次のように変わります。

```
config_interface -default_slave_interface s_slave
```

表 4: 引数のタイプ

引数のタイプ	スカラー		配列を指すポインター			hls::stream
インターフェイスモード	入力	戻り値	I	I/O	O	I および O
s_axi_lite	D	D	D	D	D	

注記:

1. D = デフォルト インターフェイス

注記: これらのデフォルトのインターフェイス プラグマ設定は、ユーザー指定のインターフェイス プラグマで変更できます。

Vivado デザイン フロー

Vitis HLS をスタンドアロン モードで使用して IP を作成できます。ツールではデフォルトでこのフローが実行され、次のグローバル オプションが設定されます。

- `open_solution -flow_target vivado`
- `config_interface -default_slave_interface s_axilite`

この場合、次のデフォルト インターフェイスが使用されます。

表 5: 引数のタイプ

引数のタイプ	スカラー		配列			ポインターまたは参照			hls::stream
インターフェイスモード	入力	戻り値	I	I/O	O	I	I/O	O	I および O
ap_ctrl_none	3	1	3	3	3	3	3	3	3
ap_ctrl_hs	3	1	3	3	3	3	3	3	3
ap_ctrl_chain	3	D ¹	3	3	3	3	3	3	3
axis	1	3	1	3	1	1	3	1	1
s_axilite	1	1	1	1	1	1	1	1	3
m_axi	3	3	1	1	1	1	1	1	3
ap_none	D ¹	3	3	3	3	D ¹	1	1	3
ap_stable	1	3	3	3	3	1	3	3	3
ap_ack	1	3	3	3	3	1	1	1	3
ap_vld	1	3	3	3	3	1	1	D ¹	3
ap_ovld	3	3	3	3	3	3	D ¹	1	3
ap_hs	1	3	1	3	1	1	1	1	1
ap_memory	3	3	D ¹	D ¹	D ¹	3	3	3	3
bram	3	3	1	1	1	3	3	3	3
ap_fifo	3	3	1	3	1	1	3	1	D ¹

注記:

1. サポートあり
2. D = デフォルト インターフェイス
3. サポートなし

構造体

内部変数およびグローバル変数などの構造体は、デフォルトでメンバー要素に分割されます。作成される要素の数とタイプは、その構造体の内容によって決まります。構造体の配列は、複数の配列 (構造体の各メンバーに別々の配列) としてインプリメントされます。

C/C++ の構造体は、データを揃えるため、コンパイラにより追加のバイトでパディングされます。Vitis HLS のカーネルコードが `gcc` に準拠するようにするため、カーネルコードに含まれる構造体は追加のバイトでパディングされます。



重要: 構造体の配列が AXI4-Stream インターフェイスにマップされている場合、構造体の配列に `ARRAY_PARTITION` または `ARRAY_RESHAPE` を適用することはできません。

任意精度型のデータ レイアウト (ap_int ライブラリ)

`ap_int` などのカスタム データ幅を持つ構造体内のデータ型は、2 のべき乗のサイズで割り当てられます。Vitis HLS は、データ型のサイズを 2 のべき乗に揃えるためにパディング ビットを追加します。

次の例では、サイズ `varA` の構造体が 5 ビットではなく 8 ビットにパディングされます。

```
struct example {
  ap_int<5> varA;
  unsigned short int varB;
  unsigned short int varC;
  int d;
};
```



ヒント: Vitis HLS は、`bool` 型もパディングして 8 ビットに揃えます。

構造体のパディングとアライメント

Vitis HLS の構造体では、`__attributes__` または `#pragmas` を使用して異なるタイプのパディングおよびアライメントを設定できます。次に、これらの機能を示します。

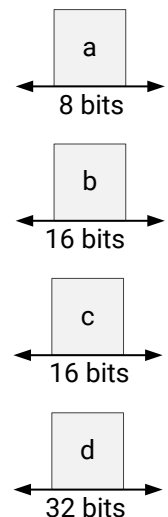
- 分割: デフォルトでは、コード内の内部変数としての構造体は個別の要素に分割されます。作成される要素の数とタイプは、その構造体の内容によって決まります。Vitis HLS により、構造体が分割されるか、特定の最適化条件に基づいているかが判断されます。



ヒント: `set_directive_aggregate` プラグマまたは指示子を使用すると、コードの構造体がデフォルトで分割されないようにできます。

図 1: 分割された構造体

```
struct example {
  ap_int<5> a;
  unsigned short int b;
  unsigned short int c;
  int d;
};
void foo()
{
  example s0;
  #pragma HLS disaggregate variable=s0
}
```



X24681-100520

- 集約: インターフェイスの構造体のデフォルトです (「インターフェイス合成および構造体」セクションを参照)。Vitis HLS で構造体の要素がまとめられ、1 つのデータ ユニットに集約されます。これは、`pragma HLS aggregate` に従って実行されます。これがインターフェイスの構造体のデフォルトなので、このプラグマを指定する必要はありません。集約プロセスでは、バイト構造をデフォルトの 4 バイト アライメントに揃えるため、要素がパディングされることがあります。構造体は、`AGGREGATE` プラグマまたは指示子で説明されているように分割できません。



ヒント: `-Wpadded` をコンパイラ フラグとして指定すると、構造体をパディングするためビットが追加された場合に警告が表示されます。

- アライメント: デフォルトでは、Vitis HLS で構造体の要素がパディングされ、4 バイト (32 ビット幅) に揃えられます。 `__attribute__((aligned(X)))` (X はバイト境界) を使用してバイト境界を指定し、構造体の要素をパディングすることもできます。次の図では、構造体を 2 バイト境界に揃えています。



重要: X は、2 のべき乗で指定する必要があります。

`ap_int` などのカスタム データ幅を持つ構造体内のデータ型は、2 のべき乗のサイズで割り当てられます。Vitis HLS は、データ型のサイズを 2 のべき乗に揃えるためにパディング ビットを追加します。

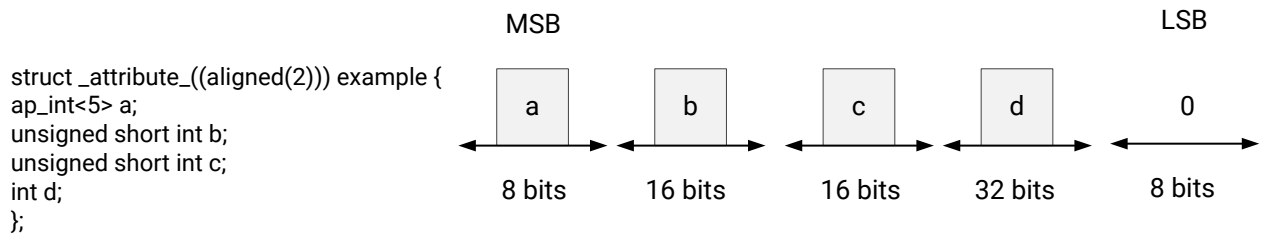
次の例では、サイズ `varA` の構造体が 5 ビットではなく 8 ビットにパディングされます。

```
struct example {
  ap_int<5> varA;
  unsigned short int varB;
  unsigned short int varC;
  int d;
};
```



ヒント: Vitis HLS は、`bool` 型もパディングして 8 ビットに揃えます。

図 2: アライメントされた構造体のインプリメンテーション



X24682-102220

使用されるパディングは、構造体の要素の順序とサイズによって異なります。次のコード例では、構造体が 4 バイトに揃えられ、Vitis HLS で最初の要素 `varA` の後に 2 バイトのパディングが追加され、3 番目の要素 `varC` の後に 2 バイトのパディングが追加されます。構造体の合計サイズは 96 ビットになります。

```

struct data_t {
  short varA;
  int varB;
  short varC;
};
        
```

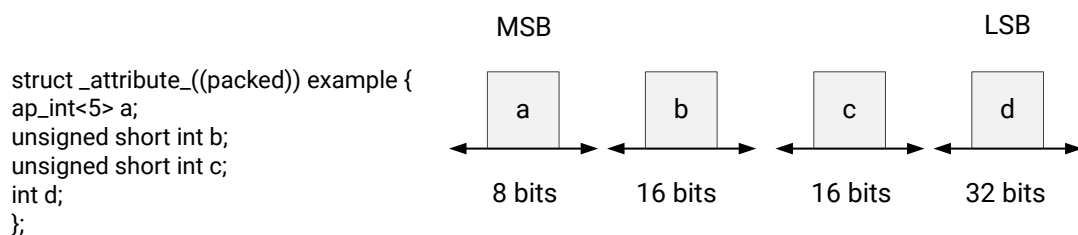
構造体を次のように記述し直すと、パディングは必要なくなり、構造体の合計サイズは 64 ビットになります。

```

struct data_t {
  short varA;
  short varC;
  int varB;
};
        
```

- **パッキング:** `__attribute__((packed(X)))` を指定すると、Vitis HLS で構造体が各要素の実際のサイズに基づいてパックされます。次の例では、構造体のサイズは 72 ビットになります。

図 3: パッキングされた構造体のインプリメンテーション



X24680-102220

インターフェイス バンドルの規則

インターフェイス プラグマには、関数引数を AXI インターフェイス ポートにグループ化するバンドル オプションがあります。次のセクションに、ユーザー定義のポートとユーザー定義されていないポートが混ざっている場合のバンドルの適用規則を示します。

S_AXI Lite

ユーザー定義およびデフォルトのバンドル名

- 規則 1: ユーザー指定のバンドル名: 同じ `bundle=<string>` が指定されたすべてのインターフェイス ポートが 1 つの AXI4-Lite インターフェイス ポートにまとめられ、RTL ポートの名前は `s_axi_<string>` になります。

```
void top(char *a, char *b, char *c, char *d) {
#pragma HLS INTERFACE s_axilite port=a bundle=terry
#pragma HLS INTERFACE s_axilite port=b bundle=terry
#pragma HLS INTERFACE s_axilite port=c bundle=stephen
#pragma HLS INTERFACE s_axilite port=d bundle=jim
}
```

```
INFO: [RTGEN 206-100] Bundling port 'd' to AXI-Lite port jim.
INFO: [RTGEN 206-100] Bundling port 'c' to AXI-Lite port stephen.
INFO: [RTGEN 206-100] Bundling port 'a' and 'b' to AXI-Lite port terry.
INFO: [RTGEN 206-100] Finished creating RTL model for 'example'
```

- 規則 2: デフォルトのバンドル名: バンドル名が指定されていないすべてのインターフェイス ポートが 1 つの AXI4-Lite インターフェイス ポートにまとめられ、ツールのデフォルト `bundle=<default>` が使用され、RTL ポートの名前は `s_axi_<default>` になります。

```
void top(char *a, char *b, char *c, char *d) {
#pragma HLS INTERFACE s_axilite port=a
#pragma HLS INTERFACE s_axilite port=b
#pragma HLS INTERFACE s_axilite port=c
#pragma HLS INTERFACE s_axilite port=d
}
```

```
Log fileINFO: [RTGEN 206-100] Bundling port 'a', 'b', 'c' to AXI-Lite
port control.
INFO: [RTGEN 206-100] Finished creating RTL model for 'example'.
```

- 規則 3: 一部にバンドル名を指定: 一部のポートにバンドル名を指定した場合、次のバンドル規則に従って、複数の `s_axi lite` インターフェイス ポートが作成されます。
 - バンドル名 (下の例では `bundle=control`) が指定されたインターフェイス ポートが 1 つの AXI4-Lite インターフェイス ポートにまとめられます。
 - バンドル名が指定されていないインターフェイス ポートがデフォルト名のインターフェイス ポートにまとめられます。

```
void top(char *a, char *b, char *c, char *d) {
#pragma HLS INTERFACE s_axilite port=a
#pragma HLS INTERFACE s_axilite port=b
#pragma HLS INTERFACE s_axilite port=c bundle=control
#pragma HLS INTERFACE s_axilite port=d bundle=control
}
```

```
INFO: [RTGEN 206-100] Bundling port 'c' and 'return' to AXI-Lite port
control.
INFO: [RTGEN 206-100] Bundling port 'a' and 'b' to AXI-Lite port
control_r.
INFO: [RTGEN 206-100] Finished creating RTL model for 'example'.
```

MAXI

- **グローバルバンドル設定なし:** `config_interface -m_axi-auto-max-ports false` を使用すると、バンドル規則は次のようになります。

- **規則 1: ユーザー指定のバンドル名:**

- この規則では、同じ `bundle=<string>` が指定されたすべてのインターフェイス ポートが 1 つの AXI MAXI インターフェイス ポートにまとめられ、RTL ポートの名前は `m_axi-<string>` になります。

```
#pragma HLS INTERFACE m_axi port=a depth=50 bundle=terry
#pragma HLS INTERFACE m_axi port=a depth=50
#pragma HLS INTERFACE m_axi port=a depth=50
Log file
INFO: [RTGEN 206-500] Setting interface mode on port 'example/terry' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem0' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem1' to
'm_axi'.
```

```
#pragma HLS INTERFACE m_axi port=a depth=50 bundle=terry
#pragma HLS INTERFACE m_axi port=a depth=50 bundle=terry
#pragma HLS INTERFACE m_axi port=a depth=50 bundle=terry
Log file
INFO: [RTGEN 206-500] Setting interface mode on port 'example/terry' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/terry' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/terry' to
'm_axi'.
```

- **規則 2: デフォルトのバンドル名:**

- この規則では、バンドル名が指定されていないすべてのインターフェイス ポートが 1 つの AXI インターフェイス ポートにまとめられ、ツールのデフォルト `bundle=<default>` が使用され、RTL ポートの名前は `<default>-m_axi` になります。

```
#pragma HLS INTERFACE m_axi port=a depth=50
#pragma HLS INTERFACE m_axi port=a depth=50
#pragma HLS INTERFACE m_axi port=a depth=50
Log file
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem0' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem0' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem0' to
'm_axi'.
```

```
#pragma HLS INTERFACE m_axi port=a depth=50 bundle=terry
#pragma HLS INTERFACE m_axi port=a depth=50
#pragma HLS INTERFACE m_axi port=a depth=50
Log file
INFO: [RTGEN 206-500] Setting interface mode on port 'example/terry' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem0' to
'm_axi'.
INFO: [RTGEN 206-500] Setting interface mode on port 'example/gmem0' to
'm_axi'.
```

- **グローバルバンドルを設定:**

- グローバル コンフィギュレーション オプション `config_interface -m_axi-auto-max-ports true` を使用すると、次のようになります。
 - この規則では、バンドルが指定されていないインターフェイス ポートは個別の AXI MAXI インターフェイス ポートにマップされ、RTL ポート名は `gmem_0`, `gmem_1`, `gmem_2` のようになります。

インターフェイス オフセット

インターフェイス オプションには、AXI4-Lite (`s_axilite`) および AXI4 (`m_axi`) インターフェイスのアドレス オフセットを指定するオフセット オプションがあります。次に、ユーザー定義されているポートとされていないポートが混ざっている場合のオフセットの適用規則を示します。

SAXI Lite

- 規則 1: 完全指定: すべてのスカラーおよびオフセットを明示的にユーザー定義の AXI4-Lite ポートにグループ化します。
- 規則 2: デフォルト指定: すべてのスカラーおよびオフセットオフセットをオフセット設定なしでグループ化し、デフォルト AXI4-Lite ポートをアップデートします。
- 規則 3: 部分指定: オフセットを部分的に指定した場合、複数の `s_axi_lite` インターフェイス ポートが作成されます。

MAXI のオフセット

- **オフセットを完全指定:** プラグマのユーザー指定オフセット設定に従います。
- **オフセットの指定なし:** `maxi` のオフセットをプラグマで指定しない場合、オフセット規則にグローバル コンフィギュレーション オプション `config_interface -m_axi-offset <off/direct/slave>` が適用されます。
 - **規則 1:** ユーザー指定 SAXI Lite: `MAXI offset=<slave>` が指定されたすべての `maxi` オフセットがユーザー指定 `AXI_lite` ポートにグループ化されます。

```
void top(int *a) {
  #pragma HLS interface m_axi port=a
  #pragma HLS interface s_axilite port=a
}
```

- **規則 2:** SAXI Lite の指定なし: すべての `maxi` オフセットがツールのデフォルト オフセットにグループ化されます。
 - Vitis: `offset = slave`
 - Vivado: `offset = off`
 - ユーザー指定: `config_interface -m_axi-offset direct`

```
void top(int *a, int *b) {
  #pragma HLS interface m_axi port=a bundle=M0
  #pragma HLS interface m_axi port=b bundle=M0
}
```

サイドチャンネルありの AXI4-Stream インターフェイス

サイドチャンネルは、AXI4-Stream 規格の一部であるオプションの信号です。サイドチャンネル信号は、構造体のメンバー要素が AXI4-Stream のサイドチャンネル信号の名前と一致していれば、C/C++ コードで構造体を使用して直接参照および制御できます。AXI4-Stream サイドチャンネル信号はデータ信号と認識され、TDATA にレジスタが付けられると、レジスタが付けられます。

config_rtl -module_auto_prefix の動作の変更

Vivado HLS では、`config_rtl -module_auto_prefix` をイネーブルにすると、最上位 RTL モジュール名の接頭辞としてそれ自体のモジュール名が付けられました。2020.1 の Vitis HLS では、この自動接頭辞機能はサブモジュールにのみ適用されます。

`-module_prefix` の動作には変更はありません。このオプションを使用すると、指定の接頭辞が最上位モジュールを含むすべてのモジュールに追加されます。また、`-module_prefix` オプションが `-module_auto_prefix` オプションよりも優先されることも変わりません。

```
# vivado HLS 2020.1 generated module names (top module is "top")
top_top.v
top_submodule1.v
top_submodule2.v

# Vitis HLS 2020.1 generated module names
top.v          <-- top module no longer has prefix
top_submodule1.v
top_submodule2.v
```

プラグマ構文

allocation

- allocation プラグマの構文は、次のとおりです。

例:

```
#pragma HLS allocation instances=mul limit=1 operation // Invalid format
#pragma HLS allocation operation instances=mul limit=1 // Valid format
#pragma HLS allocation instances=foo limit=2 function // Invalid format
#pragma HLS allocation function instances=foo limit=2 // Valid format

IMPORTANT: If the referenced function/operation is not located after
"allocation", it will be ignored with a warning message:
WARNING: [HLS 207-1604] unexpected pragma argument 'instances', expects
function/operation.
```

- template 関数を使用する場合は、プラグマ構文は次のようにする必要があります。

```
template <typename DT>
void foo(DT a, DT b){
}
```

```
Invalid syntax
// Below is invalid
```

```
#pragma HLS ALLOCATION function instances = foo
Valid syntax

// Below is valid
#pragma HLS ALLOCATION function instances = foo<DT>
```

dependence

- dependence プラグマの構文には、常に true/false オプションが必要です。

```
// Below will emit a warning
#pragma HLS dependence variable=a

// Below will not have a warning
#pragma HLS dependence variable=a false
```

インターフェイスのメモリ プロパティ

インターフェイス プラグマの `storage_type` オプションを使用すると、使用する RAM のタイプおよび作成する RAM ポート (シングル ポートまたはデュアル ポート) を指定できます。 `storage_type` を指定しない場合、Vitis HLS で次が使用されます。

- シングル ポート RAM (デフォルト)。
- 開始間隔またはレイテンシが削減される場合はデュアル ポート RAM。

Vivado フローでは、`resource` プラグマで `storage_type` を指定する以前の方法に代わり、指定したインターフェイスで RAM ストレージのタイプを指定できます。

```
#pragma HLS INTERFACE bram port = in1 storage_type=RAM_2P
#pragma HLS INTERFACE bram port = out storage_type=RAM_1P latency=3
```

サポートされない機能

このリリースでは、次の機能はサポートされていません。



重要: このセクションにリストされているサポートされない機能に対しては、HLS から警告メッセージまたはエラーメッセージが表示されます。

最上位関数引数

プラグマ

- 2 つ以上のポートに `bundle` を指定する `m_axi` INTERFACE プラグマが含まれる引数に `DEPENDENCE` プラグマを使用することはサポートされません。

```
void top(int *a, int *b) { // both a and b are bundled to m_axi port gmem
    #pragma HLS interface m_axi port=a offset=slave bundle=gmem
    #pragma HLS interface m_axi port=b offset=slave bundle=gmem
    #pragma HLS dependence variable=a false
}
```

- INTERFACE プラグマの `ap_bus` モードは、Vivado HLS ではサポートされていましたが、サポートされなくなっています。 `m_axi` インターフェイスを使用してください。

データ型

最上位関数引数に次のデータ型を使用することは、このリリースではサポートされません。

- enum および enum の使用 (構造体、enum の配列ポインター)
- 複素数
- half、fp16

HLS ビデオ ライブラリ

ビデオ ユーティリティおよび関数用の `hls_video.h` は廃止予定であり、Vitis ビジョン ライブラリに置き換えられています。詳細は、GitHub の [HLS ビデオ ライブラリの Vitis ビジョンへの移行](#)を参照してください。

C の任意精度型

Vitis HLS では、C の任意精度型はサポートされません。サイリンクスでは、任意精度の C++ 型を使用することをお勧めします。

C++ 任意精度型でサポートされる最大幅は、Vivado HLS では 32K ビットですが、Vitis HLS では 4096 ビットです。

C コンストラクト

- ポインター キャストはサポートされなくなっています。
- 仮想関数はサポートされなくなっています。

廃止予定およびサポートされない Tcl コマンド オプション

Vitis™ HLS では、多数の Vivado® HLS コマンドが廃止予定になっています。次の表に示す廃止予定のコマンドは、Vitis HLS ツールではサポートされなくなります。

表 6: Vitis HLS で廃止予定の Vivado HLS コマンド

タイプ	コマンド	オプション	Vitis HLS	詳細
コンフィギュレーション	config_interface	-m_axi_max_data_size	廃止予定	
コンフィギュレーション	config_interface	-m_axi_min_data_size	廃止予定	
コンフィギュレーション	config_interface	--m_axi_alignment_size	廃止予定	
コンフィギュレーション	config_interface	-expose_global	サポートなし	
コンフィギュレーション	config_interface	-trim_dangling_port	サポートなし	
コンフィギュレーション	config_array_partition	-scalarize_all	サポートなし	
コンフィギュレーション	config_array_partition	-throughput_driven	サポートなし	
コンフィギュレーション	config_array_partition	-maximum_size	サポートなし	
コンフィギュレーション	config_array_partition	-include_extern_globals	サポートなし	
コンフィギュレーション	config_array_partition	-include_ports	サポートなし	
コンフィギュレーション	config_schedule	-enable_dsp_fill_reg 以外のすべてのオプション	廃止予定	
コンフィギュレーション	config_bind	*(すべてのオプション)	廃止予定	
コンフィギュレーション	config_rtl	-encoding	廃止予定	
コンフィギュレーション	config_sdx	*(すべてのオプション)	廃止予定	

表 6: Vitis HLS で廃止予定の Vivado HLS コマンド (続き)

タイプ	コマンド	オプション	Vitis HLS	詳細
コンフィギュレーション	config_flow	* (すべてのオプション)	廃止予定	
コンフィギュレーション	config_dataflow	-disable_start_propagation	廃止予定	
コンフィギュレーション	config_rtl	-auto_prefix	廃止予定	config_rtl -module_prefix に置き換え。
コンフィギュレーション	config_rtl	-prefix	廃止予定	config_rtl -module_prefix に置き換え。
コンフィギュレーション	config_rtl	-m_axi_conservative_mode	廃止予定	config_interface -m_axi_conservative_mode を使用。
指示子/プラグマ	set_directive_pipeline	-enable_flush	廃止予定	
指示子/プラグマ	CLOCK	*	サポートなし	
指示子/プラグマ	DATA_PACK	*	サポートなし	AGGREGATE プラグマまたは指示子を使用、必要に応じて__attribute__((packed(X)))を使用。
指示子/プラグマ	INLINE	-region	廃止予定	
指示子/プラグマ	INTERFACE	-mode ap_bus	サポートなし	代わりに m_axi を使用してください。
指示子/プラグマ	ARRAY_MAP	*	サポートなし	
指示子/プラグマ	RESOURCE	*	廃止予定	BIND_OP および BIND_STORAGE プラグマおよび指示子に置き換え。関数引数には INTERFACE プラグマまたは指示子を storage_type オプションと共に使用。
指示子/プラグマ	STREAM	-dim	サポートなし	
プロジェクト	csim_design	-clang_sanitizer	追加/名前変更	
プロジェクト	export_design	-use_netlist	廃止予定	export_design -format ip_catalog に置き換え。
プロジェクト	export_design	-xo	廃止予定	export_design -format xo に置き換え。
プロジェクト	add_files		サポートなし	System C ファイルは Vitis HLS ではサポートされない。

注記:

1. 廃止予定: プラグマが今後のリリースでサポートされなくなることを示す警告メッセージが表示されます。
2. サポートなし: Vitis HLS でエラー メッセージが表示されます。
3. *: コマンドのすべてのオプション。

その他のリソースおよび法的通知

ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポート リソースは、[ザイリンクス サポート](#) サイトを参照してください。

Documentation Navigator およびデザイン ハブ

ザイリンクス Documentation Navigator (DocNav) では、ザイリンクスの資料、ビデオ、サポート リソースにアクセスでき、特定の情報を取得するためにフィルター機能や検索機能を利用できます。DocNav を開くには、次のいずれかを実行します。

- Vivado® IDE で [Help] → [Documentation and Tutorials] をクリックします。
- Windows で [スタート] → [すべてのプログラム] → [Xilinx Design Tools] → [DocNav] をクリックします。
- Linux コマンド プロンプトに「docnav」と入力します。

ザイリンクス デザイン ハブには、資料やビデオへのリンクがデザイン タスクおよびトピックごとにまとめられており、これらを参照することでキー コンセプトを学び、よくある質問 (FAQ) を参考に問題を解決できます。デザイン ハブにアクセスするには、次のいずれかを実行します。

- DocNav で [Design Hub View] タブをクリックします。
- ザイリンクス ウェブサイトで[デザイン ハブ](#) ページを参照します。

注記: DocNav の詳細は、ザイリンクス ウェブサイトの [Documentation Navigator](#) ページを参照してください。DocNav からは、日本語版は参照できません。ウェブサイトのデザイン ハブ ページをご利用ください。

参考資料

このガイドの補足情報は、次の資料を参照してください。

1. 『Vivado Design Suite ユーザー ガイド: IP を使用した設計』 ([UG896](#))
2. 『Vivado Design Suite: AXI リファレンス ガイド』 (UG1037: [英語版](#)、[日本語版](#))

お読みください: 重要な法的通知

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとし、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえば当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務を負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。<https://japan.xilinx.com/legal.htm#tos> で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ 設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとし、セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品の責任の制限を規定する適用法令および規則にのみ従うものとし、

商標

© Copyright 2020 Xilinx, Inc. Xilinx、Xilinx のロゴ、Alveo、Artix、Kintex、Spartan、Versal、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他の各国のザイリンクス社の商標です。

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。