メモリリソース

KEK IPNS E-sys
本多良太郎

この回の概要

RAMとFIFOは多くの人がすでに利用経験のある機能だと思います。
ここでは典型的な利用方法と気を付ける事の説明を行います。

- ランダムアクセスが必要ならRAM。
- 保存したものを後から順番に取り出したいならFIFO。

Random Access Memory (RAM)

Xilinx FPGAのブロックRAMは1つの記憶領域を
2つの独立ポートがシェアしているデュアルポートRAM

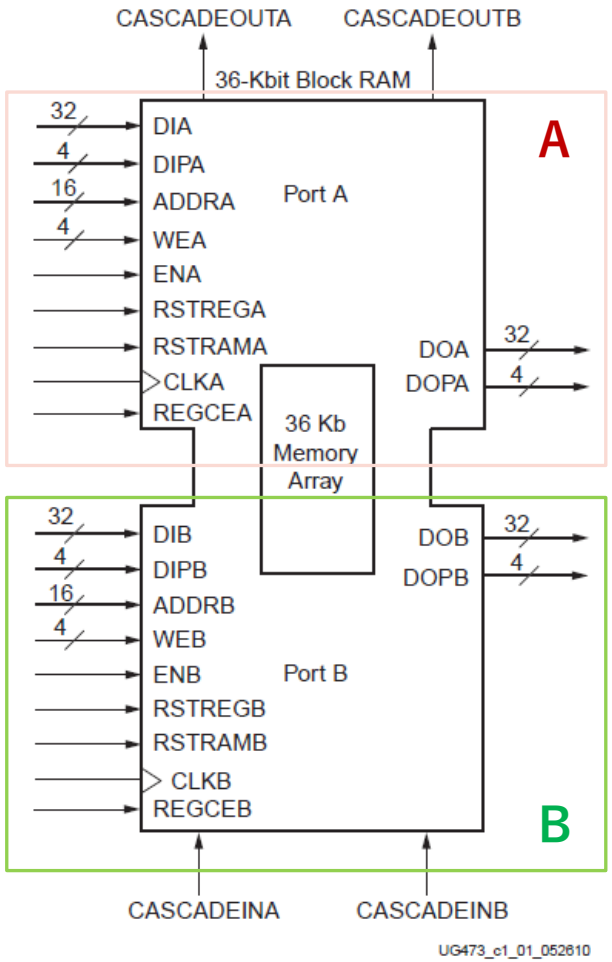


図 1-1 : RAMB36 の TDP データ フロー

ポイント

True Dual Port (TDP)

- 2つの独立ポートから1つのメモリ領域の任意の場所にアクセス可能なモード
- 独立の読み書きが可能

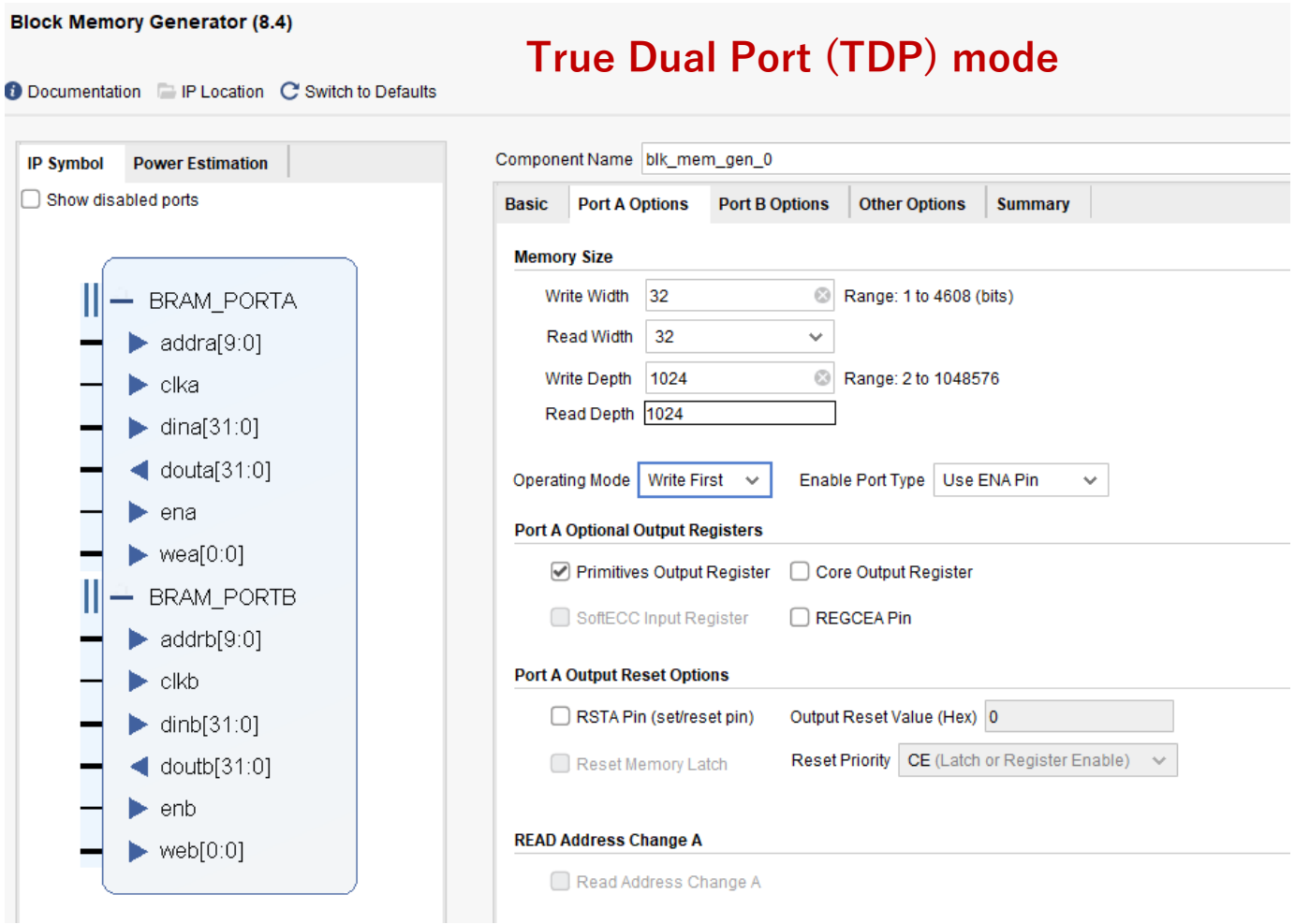
Simple Dual Port (SDP)

- ポートAを読み出し、ポートBを書き込みとするモード
- AとBのポートをまとめてポート幅を倍に出来る

- DI:　　データ入力
- DIP:　　パリティビット（拡張データとしても利用可）
- ADDR: アドレス

Block RAMはIPによって生成することがほとんどだと思います。
ここではIPカタログで生成できるコンポーネントを使って
利用法の勘所を説明します。

Block RAM (TDP mode)



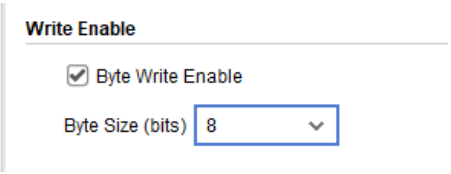
共有のメモリへポートAとBから独立に読み書きできるモード。
最もBRAMプリミティブに近い構成であるので、TDPの事をよく理解すれば他のモードにも応用できる。

ENA, ENB

- **Port** enable. 読み書き両方に対するイネーブルである事を忘れないように。

WEA, WEB

- BRAMの生ポートはbyte write enable. IPカタログで生成すると、byte write enableモードにチェックを入れない限りデータ幅全部を一気に書く（通常のwrite enable）になる。



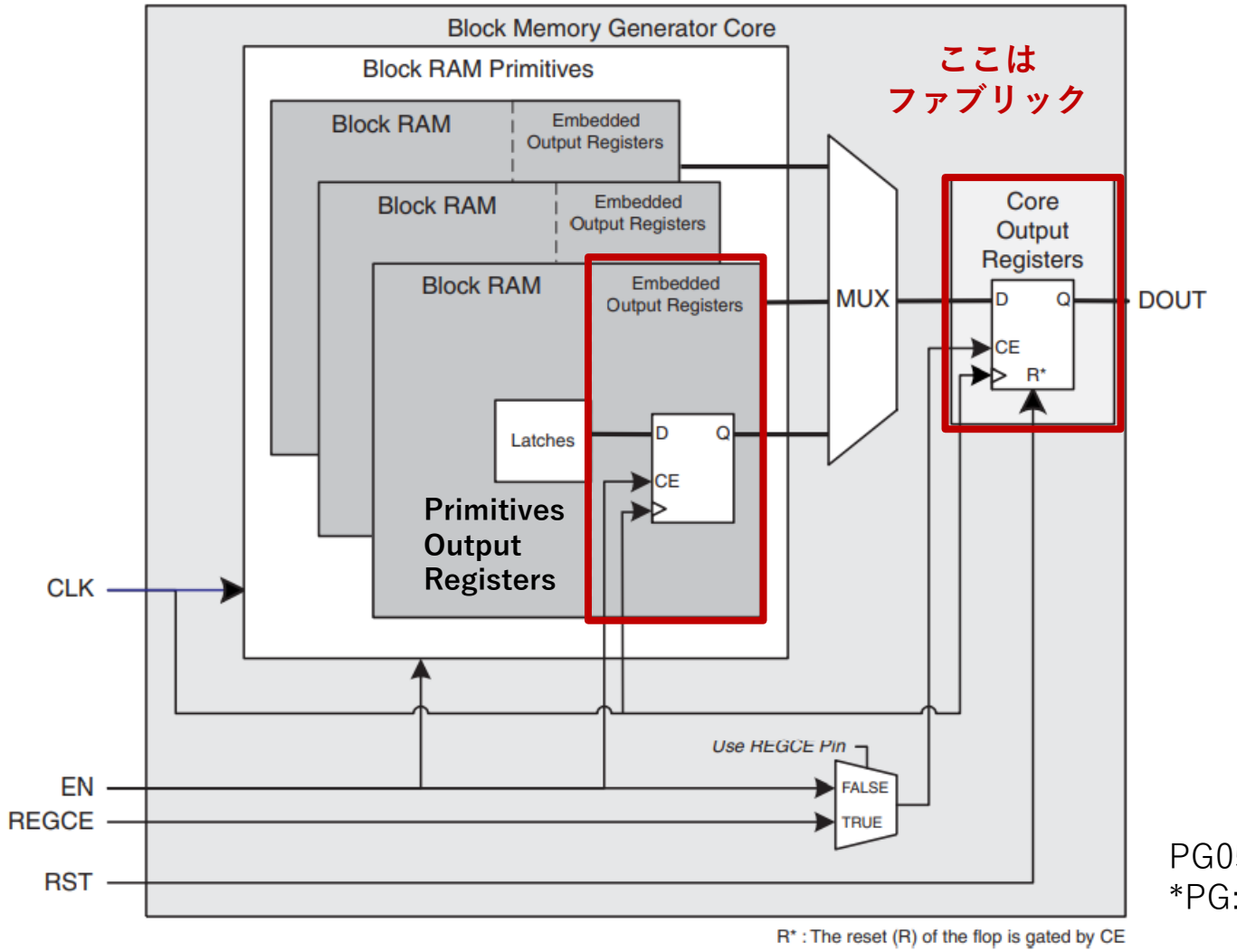
RSTA (B)

- **出力レジスタ**のリセット入力。「**メモリに対して初期値は与えられるが動的な一斉リセット信号は存在しない**」ことに注意。

Output registers

- 次のページ

Output registers

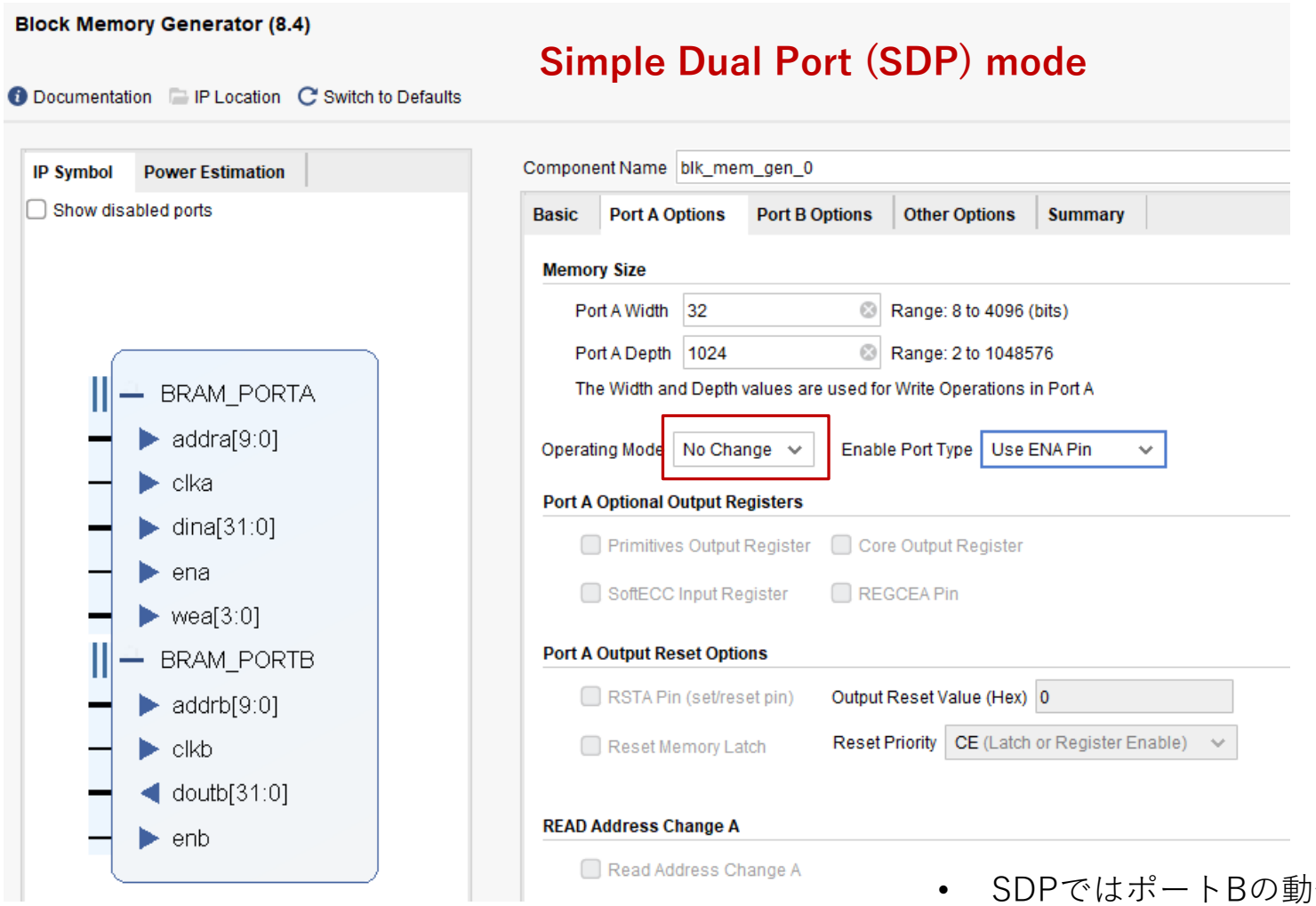


IPコアは複数のBRAMプリミティブを束ねる可能性があるためMUX後にもレジスタが設定できるようになっている。

MUXは組み合わせ回路であるため高速動作させたい場合は両方とも実装したほうがスループットが出やすいであろう。

PG058
*PG: product guide

Block RAM (SDP mode)



ポートAが書き込み専用、ポートBが読み出し専用となるモード。アドレス指定は独立。

ENA, ENB

- **Port** enable. 読み書き両方に対するイネーブルである事を忘れないように。

WEA

- TDPと同様だがポートAのみに存在。

RSTA (B)

- **出力レジスタ**のリセット入力。「**メモリに対して初期値は与えられるが動的な一斉リセット信号は存在しない**」ことに注意。

Output registers

- TDPと同様

- SDPではポートBの動作はWRITE_FIRST固定となる。

Block RAMの利用例

Electronics System Group

とりあえず私が思いつく範囲で。他にあれば是非コメントを。

ヒストグラム
- アドレスがビン番号, 値がカウント数。

補正テーブル
- 補正関数f(x)が非常に複雑場合、離散テーブルとして補正関数を与える。
- アドレスがx軸, 値が補正定数。

巨大なエンコーダ・デコーダ
- 補正テーブルと同じ考え方。アドレス値から変換後の値を取り出す。

リングバッファ
- 読み出しポインタ位置を動的に変更しうる場合。
- 書いたデータを順番に読み出すだけならFIFOの利用を検討。

いずれの場合でもアドレスをランダムに指定する必要があり
RAMでないと対応できない

TDPとSDP

これも私の思いつく範囲で。

TDP

- 同じメモリへ2つの異なったクロックドメインから読み書きが発生するパターン。
  - 低速と高速のADCデータを1つにまとめる（連続したアドレス上に配置する）。
  - 同一の補正テーブルを異なった2つのクロックドメインで利用する。
- 同一クロックドメインからアクセスする場合メモリ効率向上やスループット向上が見込める。

書き込み・読み出しどちらかでも2ポート必要ならTDP。

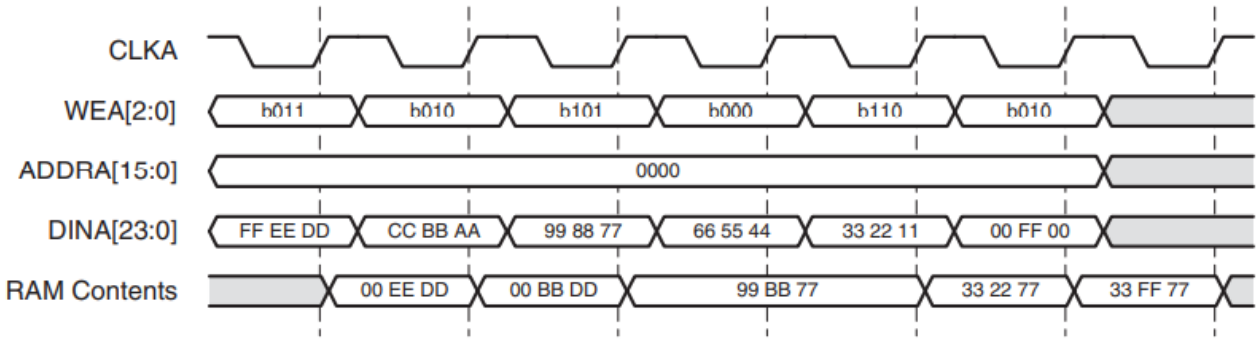
SDP

- 書く側と読む側の役割分担が決まっている場合。
- 多くの場合でリングバッファはSDPで実装できる。

TDPで実装すると恩恵があればTDP、無ければSDPでよいだろう。

バイト書き込みの勧め

Byte writeの例



8(9)-bit毎にメモリの内容を変更できる機能
TDPモードと相性が良いだろう

利用例

- 64-bit dataの下位8-bitだけ後から補正する。
- 可変長データを取り扱う。

- アドレス幅を拡張するよりもバイト書き込みで対応した方がリソース効率が良い。(BRAMプリミティブの機能であるため)
- アドレス長を拡張するよりも高速化しやすい（だろうと思う）

Byte writeも衝突には注意しましょう

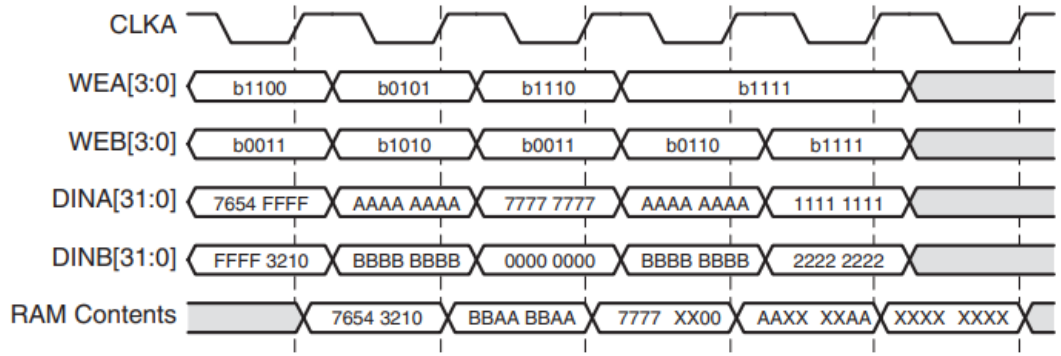


Figure 3-15: **Write-Write Collision Example**

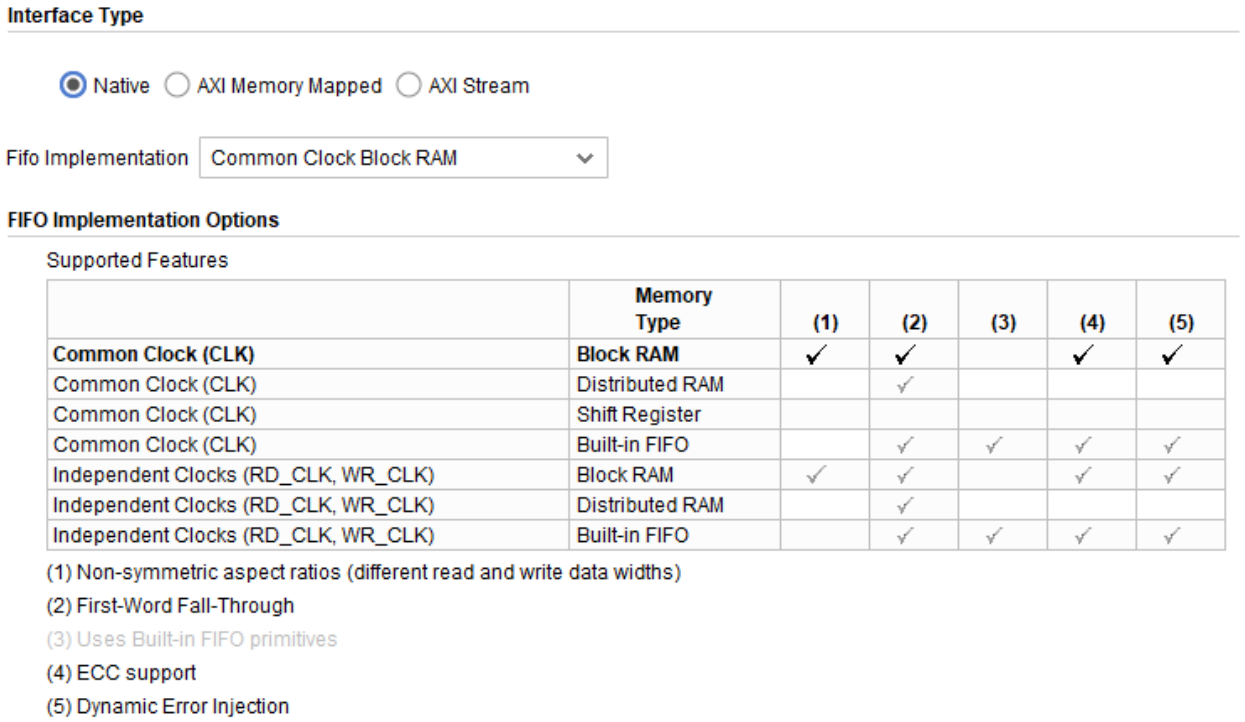
Byte write enableを使う事を考えると
FPGA内部データ構造は8(9)-bit単位になって
いる方が都合が良い
（リソース効率は最適化されない）

Electronics System Group

FIFO

openit
Open source consortium of Instrumentation

FIFOの生成



	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	**Block RAM**	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Block RAM	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection

メモリブロックに何を選択するか

Distributed RAM
- CLBでメモリを実装する。

Block RAM
- BRAMプリミティブを利用する。
  - ECCのサポートはBRAMと同様。
- FIFO制御の部分はファブリックに依存する。

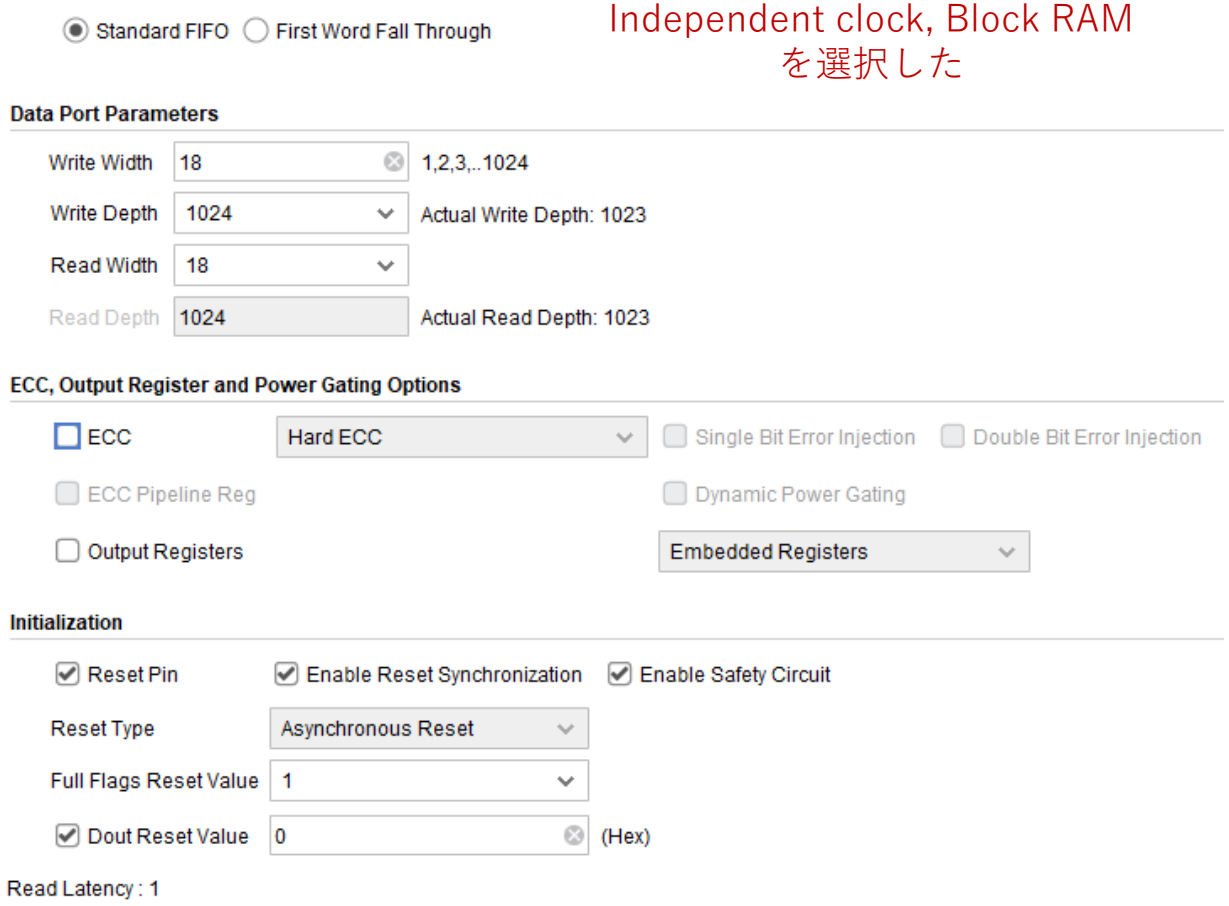
Built-in FIFO
- BRAMプリミティを利用する。
  - ECCサポートはBRAMと同様。
- FIFO制御部分も専用ブロックのリソースを使い ファブリック非依存。
  - 高速動作させやすい。CLBを消費しない。

Non-symmetric aspect ratios (入出力のデータ幅が異なるFIFO)
はbuilt-in FIFOでは使用できない点に注意

SiTCPを使う実験グループはこの機能にお世話になっているのでは

FIFOの生成

Independent clock, Block RAM
を選択した



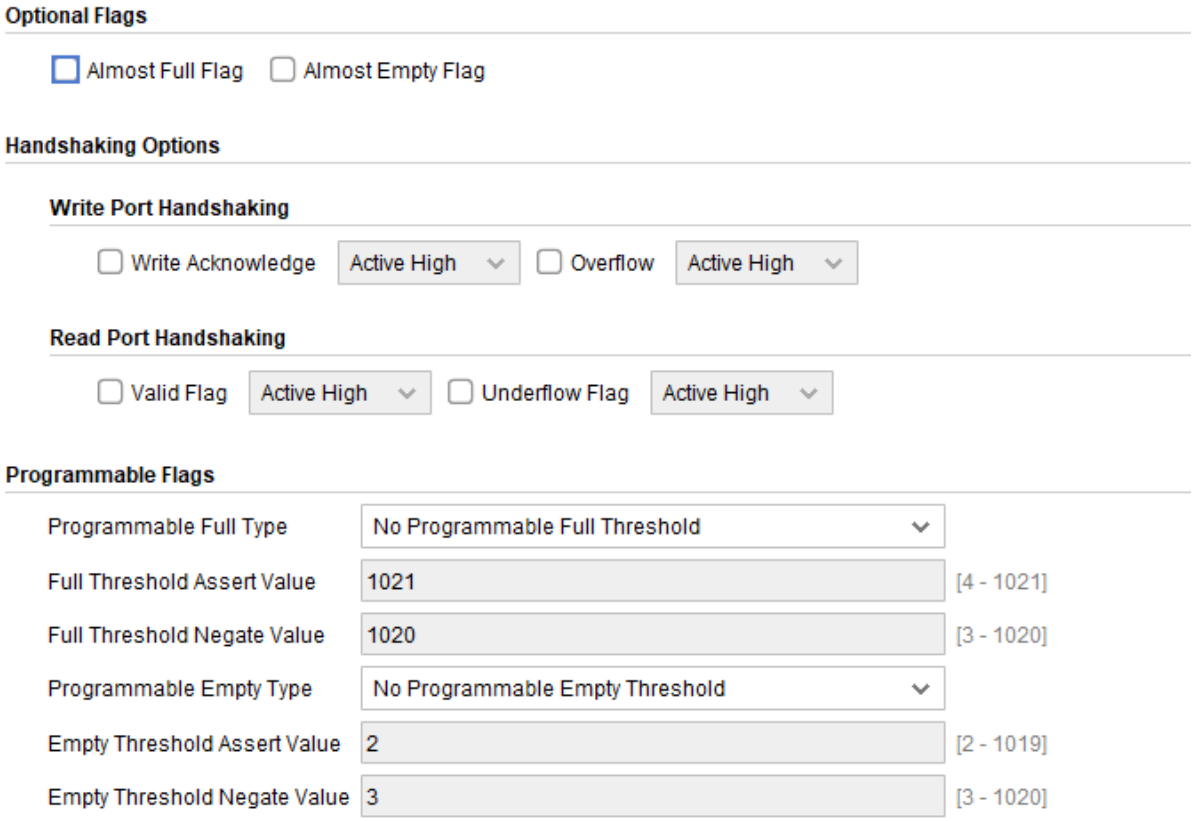
ECCと出力レジスタはBRAMと同様。

Dynamic power gating

- Built-in FIFOのみで利用可能。使っていない時にスリープさせて電力消費を抑える。

中身はBRAMであるのでメモリのコンテンツを一斉に消去する機能は存在しない。
しかしFIFOの中身が空になった、という状態を作り出すことが出来る。

FIFOの生成



Read valid

- 同クロックエッジで出力されたデータが有効であることを示す信号。とても重要。
- Read latencyはFIFOパラメータに依存するので、read enableを何クロック前にHIGHにしたから…などと考えるような設計にしてしまってはバグの元。
- 下流の回路はread validがHIGHなら決まった動作をするようにする。
- 次のFIFOへのwrite enableにそのままなる。

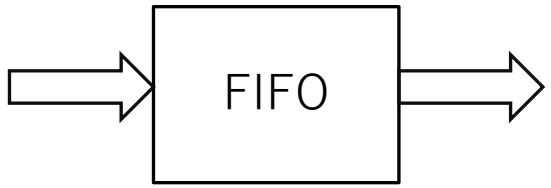
Programmable Full Threshold

- 設定した値を超えたらHIGHになるフラグ。安全な設計には欠かせない。
- 自身がいっぱいになりデータを受けられなくなったときに上流の回路へデータ送信停止を要求しないといけない。
- 上流の回路に到達するまで何クロックかかかるかもしれないし、上流のメモリにもread latencyがあるだろう。
- 停止要求を出してから実際にwrite enableがlowになるまでの猶予をこのフラグで与えるように設計する。

FIFOの利用例

WRCLKとRDCLKが共通

上流の回路は
好きなタイミングで
書けるだけ書く



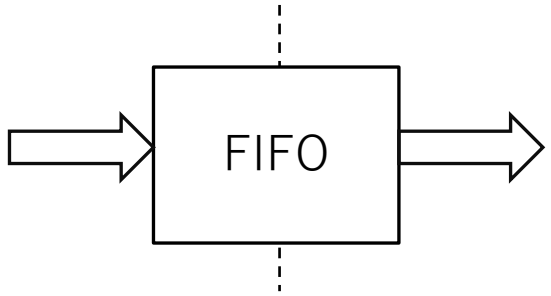
下流の回路は
好きなタイミングで
空になるまで読む

最も簡単な利用方法。データの一時置き場。
Programmable Full (pg-full)ThresholdとRead Valid, Empty
さえ見ていればトラブルはまず起きない

余談
EmptyはRDCLKに同期して遷移する。
Emptyの反転をread enableに入れておく
と自動的に読み始めてくれる。

WRCLK≠RDCLK
クロック乗り換え

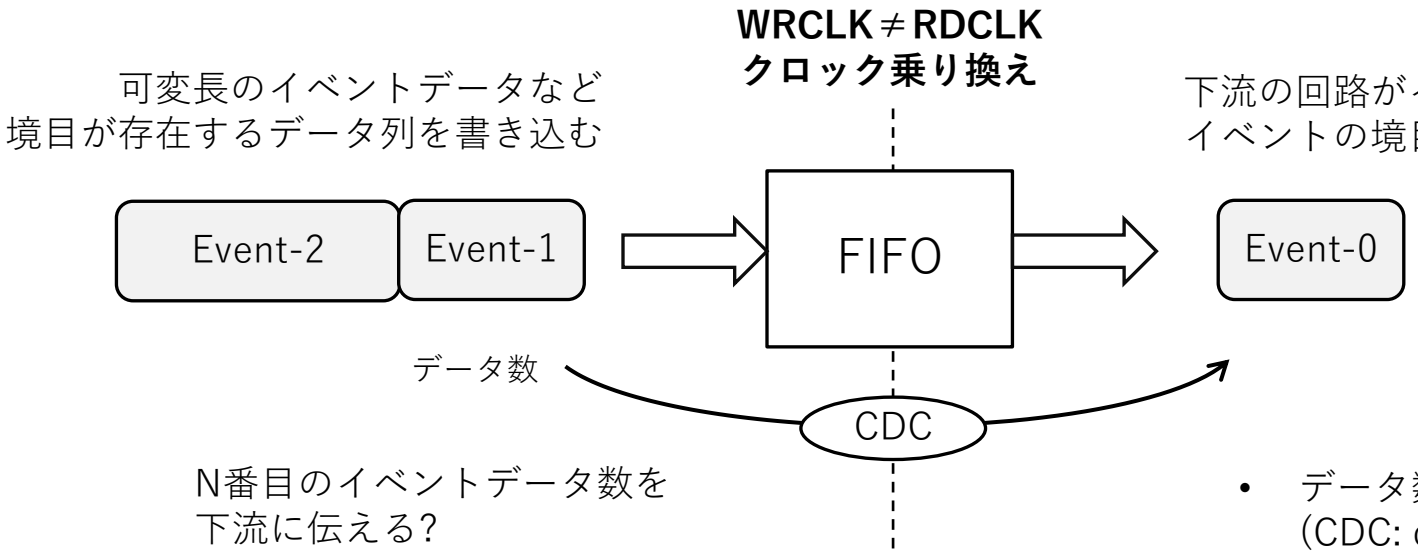
上流の回路は
好きなタイミングで
書けるだけ書く



下流の回路は
好きなタイミングで
空になるまで読む

FIFOの典型的な利用方法
安全なクロック乗り換え用の緩衝材
これもpg-fullとvalid, empty信号を見ていればほぼトラブルにならない

FIFOの利用例

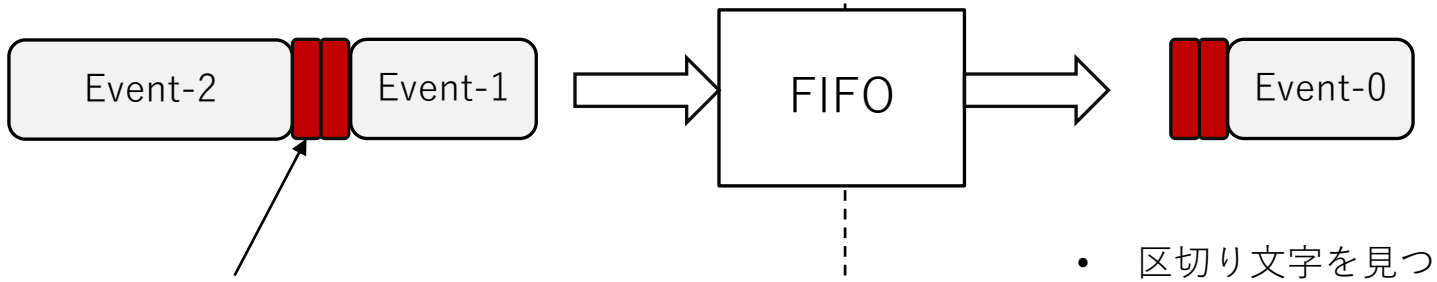


- データ数の伝送にもクロックドメイン乗り換え(CDC: clock domain crossing)が発生する。
- 数を数えて読み出し制御をするので多少複雑。

FIFOの利用例

WRCLK≠RDCLK
クロック乗り換え

可変長のイベントデータなど
境目が存在するデータ列を書き込む



イベントデータ間に区切り文字を
挟んでおく

下流の回路がイベント毎の処理をしたいので
イベントの境目で一度読み出しを止めたい

- 区切り文字を見つけたら読み出しを一旦止め、データフロー制御をしている回路の判断を待つ。

処理が簡単になった

ポイント

- 読み出し結果を見てからread enableの制御をすると必ず遅れます
  （余分なデータを読み出してしまう）
  - Read latencyが1なら2つ区切り文字を連続して挟む。
  - Read latencyが2なら3つ。

余談

- First-data-fall-through modeではDOに次のデータが現れますが本当に有効なデータかどうかは各フラグを見て判断する必要があり、なおかつフラグアサートまでのレイテンシを正しく理解している必要があります。
- Read validを伴ったDO出力から判断した方が無難です。

FIFOの同期性

FIFOを使うときは非同期動作であることが多いと思います。
WRCLKとRDCLKがたとえ同期していたとしてもフラグレイテンシは複雑です。
どうしても同期を取りたい場合UG473とPG057を熟読する必要があります。

ケース 1 : 空の FIFO への書き込み

図 2-6 は、完全に空の FIFO に対して書き込み操作を行った場合のタイミングを示しています。

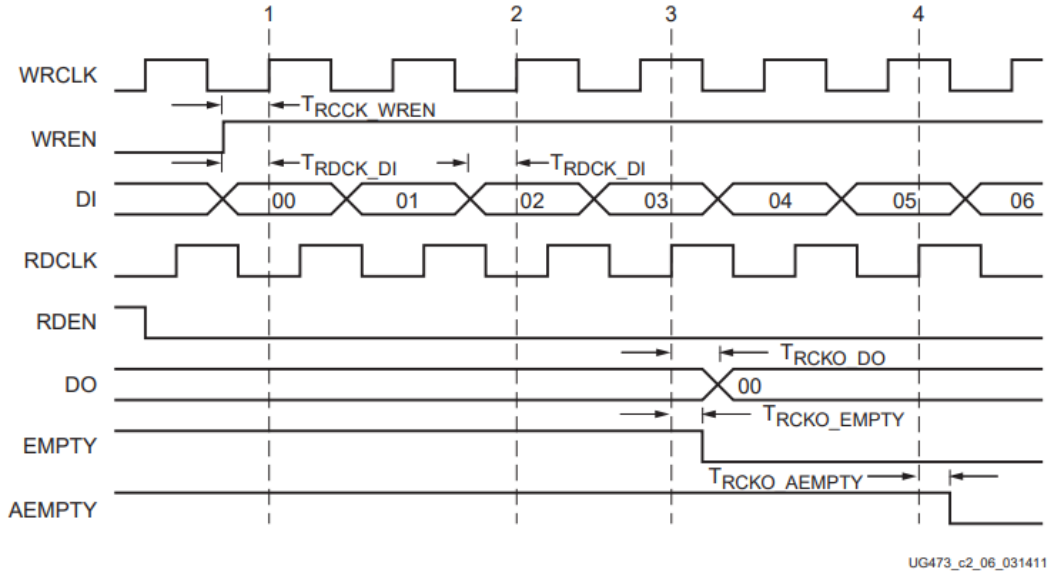


図 2-6 : 空の FIFO への書き込み (FWFT モード)

Built-in FIFOプリミティブでは書き込みが有効になってから
3 RDCLKサイクル (通常モード), もしくは4サイクル (FDFT)
後にemptyがディアサートされる。

ただし、

WRCLK と RDCLK の立ち上がりエッジが近接していると、EMPTY 信号が RDCLK の 1 周期後にディアサートされることがあります。

とあり位相関係によって一意に決まらない事が分かる。

UG473の記述はbuilt-in FIFOプリミティブの
説明である点に注意。

IPで生成したFIFOのフラグレイテンシはPG057のLatencyの
章を参照。

FIFOのリセット

- UG473ではbuilt-in FIFOプリミティブの全フラグリセットにはWRCLKとRDCLKそれぞれに対して3クロックサイクル分、リセット信号をHIGHに保つと記述がある。
- PG057にはリセット長について記述が無いように思える。

FIFOへのリセットは3サイクル長は入力しておく方が無難だろう。

Electronics System Group

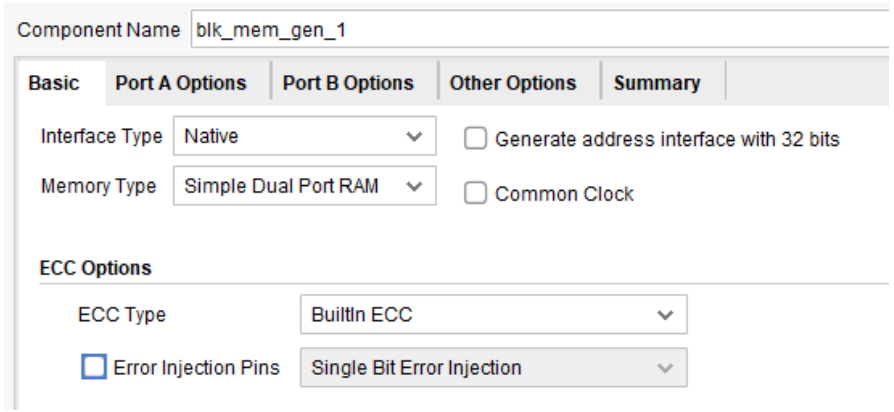
Error Correction Code

openit
Open source consortium of Instrumentation

Xilinx 7シリーズFPGAのECC

- Xilinx FPGAではECC (誤り訂正符号)にHamming符号とHSIAO符号をサポートします。
  - (原理は長くなるので割愛します。)
- 両者ともパリティビットを基にしています。
  - 1-bitのパリティビットはsingle bit errorが起きたことは分かりますが、どこで起きたのか分からないので訂正まではできません。
  - Hamming符号もHSIAO符号もsingle bit errorは起きた場所が特定できdouble bit errorは起きたことまでしか分かりません。

Block RAMの場合



ECCはBRAMプリミティブレベルでSDPモードしかサポートしていないため
IPカタログでもECCを選択可能なのはSDPモードのみ。

72x512に構成されるのでデータ幅によってはリソース効率が悪い。

Single bit error

- ビット反転が起きているメモリ領域を読むとデータと一緒にsbiterrがアサートされる。
- ECC機能により**出力は誤り訂正される**。メモリコンテンツは**訂正されずそのまま**。

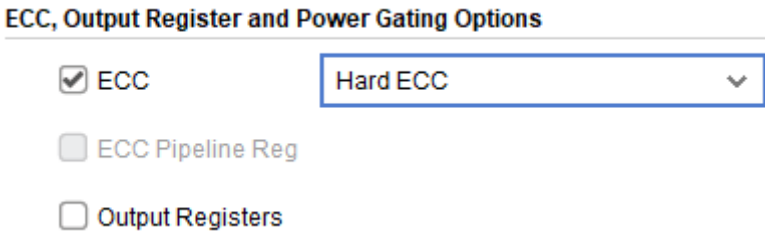
Double bit error

- ビット反転が起きているメモリ領域を読むとデータと一緒にdbiterrがアサートされる。
- **訂正不可能なので出力も訂正されずそのまま。**

rdaddrecc

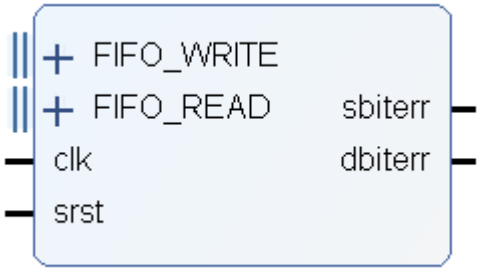
- 現在のデータがどのアドレスに格納されていたのかを表す線。
- Bit errorが起きていた場所の特定に利用する。

FIFOの場合



Block RAMとbuilt-in FIFOでしかサポートされていない。
中身は両者ともBRAMプリミティブなので72x512に構成される。
やはりデータ幅によってはリソース効率が悪い。

SummaryでBRAMをいつく消費するかチェックしてみるとよい。
18K BRAMで収まるはずのでデータ幅でも必ず36K BRAMが選択される。



Single bit error

- ビット反転が起きているメモリ領域を読むとデータと一緒にsbiterrがアサートされる。
- ECC機能により**出力は誤り訂正される**。

Double bit error

- ビット反転が起きているメモリ領域を読むとデータと一緒にdbiterrがアサートされる。
- **訂正不可能なので出力も訂正されずそのまま。**

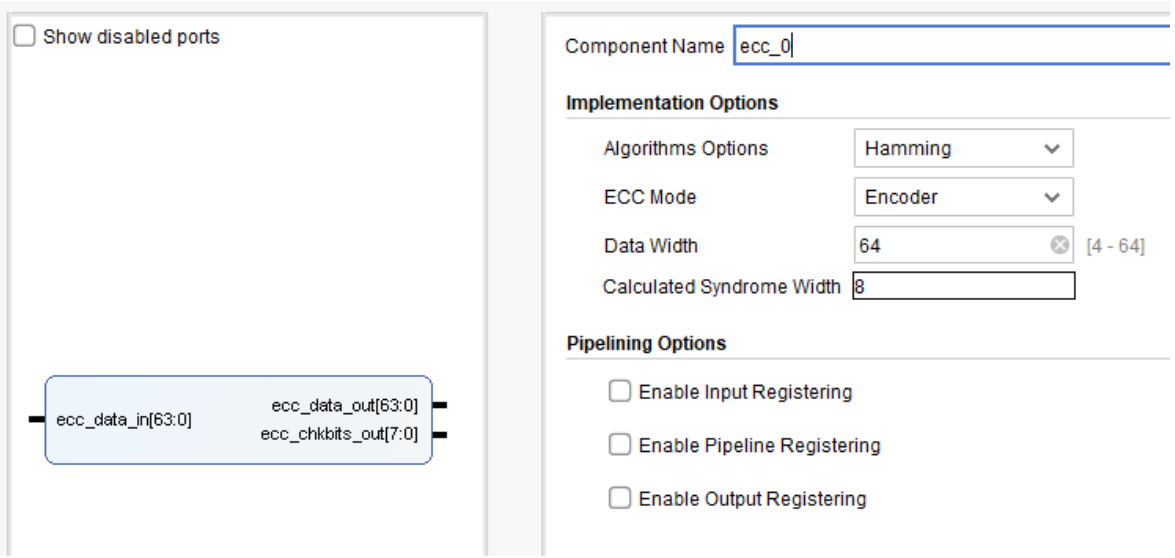
FIFOでは一度読みだしたデータは二度と読み出さないため、エラーが起きたアドレスを特定する仕組みは存在しない。

FPGAのメモリリソースに依存しないECC

ECCのエンコーダ・デコーダはIPで提供されている。
外部メモリや分散RAMに対してECCを付与したい場合に使う。

Encoder
- 入力データからECCの計算を行い、チェックビット列を与える。
- 外部メモリ等にデータと一緒に書き込む。

Decoder
- チェックビットとデータから誤り検知と可能であれば訂正を行う。



ECCエンコーダ・デコーダは組み合わせ回路なのでCLBを消費して生成される。

高速動作させる場合にはパイプライン化を施す。

ECCの動作を実際に見てみる

BRAMのbuilt-in ECC機能とECC encoder/decoderの機能を実際に見てみましょう。

BRAM

- SDP modeでBRAMを生成しBuiltin ECCを有効にしてください。
- Error Injection Pinsを有効にして、Single and Double Error injectionを選択してください。
  - 意図的にデータ破損を挿入する信号。
  - ECCがどのように動作するのか、またその下流の回路がそれに対処できるかチェックするために使う。
  - Injection pinsへの入力はone shot pulse。
- 入力データおよびアドレスはVIOが与えるものとする。

ECC encoder/decoder

- Encoderとdecoderをそれぞれ生成し直接つなぐ。
- 意図的にデータ破損を生み出すためにビット反転パターンをVIOから出力し、encoderが出力したデータと排他的論理和をとる。
- 入力データはVIOが与えるものとする。

演習手順書
EX3