Hadron Universal Logic module
仕様書 兼 ユーザーガイド

2018 年 8 月 22 日

本多 良太郎

目次

更新歴

2017.12.19

- 2 章に Mezzanine DTL の項を追加。
- 2 章に Mezzanine HR-TDC の項を追加。
- 既存のファームウェアのリセットシーケンスの統一。（3 章に記述）
- 既存のファームウェアは 100 Mbps の通信はサポートしてない事を明記。
- 既存のファームウェアのデータヘッダに RM が刺さっている事を示すビットを追加。（3 章に記述）
- HUL MH-TDC の高負荷に対する安定性を強化。イベントワードサイズのビット幅を 12bit へ変更。10^9 までイベントスリップがない事を確認。
- 3 章に HR-TDC の項を追加。
- 5 章に何点か実践的な使い方を追加。
  - SPI flash を MT25QL512 に変更した基板について。
  - 簡単なテストの方法。
  - クレートにたくさん挿して使用する方法。
  - HR-TDC の使い方。
- 制御用 C++ソフトウェアの sitcp_controller.cc を変更。Reset_SiTCP のスリープ設定を行う。
- 制御用 C++ソフトウェアから gcc バージョン依存性が強い gzfilter.hh を削除。データ圧縮を行うオプションを削除。

2018.2.2

- RM, Scaler, MH-TDC, HRTDC_BASE において J0 バスからイベントタグを受け取ると HRM が刺さっているモジュールとの間で、イベント番号が 1 ずれる問題を解決。
- Mezzanine HR-TDC, MH-TDC において、設定したサーチウィンドウ外からデータが返ってくるバグを解決。
- HR-TDC_BASE で BCT::Reset をかけると Local bus がハングするバグを解決。
- 5 章に KEK VME クレートにさして、Level2 を J0 から受け取る設定の場合、モジュールリセット後にスレーブ側がトリガーを受け取れなくなる問題について記述。（未解決）
- 5 章にイベントスリップが発生した場合について記述しました。

2018.8.22

- 5 章に SPI flash を S25FL256SAGNFI001 に交換した基板についてを追記。
- 5 章にオリジナルの SPI flash memory である N25Q128A の取り扱いについて記述。

0. モジュール概要

Hadron Universal Logic (以下 HUL) module はこれまでハドロン実験で利用されてきた Tohoku Universal Logic (TUL) の後継機として開発されたモジュールです。FPGA を最新の Xilinx Kintex7 に切り替え、より複雑で高リソースなロジックを高速に動かすことが可能になりました。HUL は 2 つの固定入力コネクタ、2 つのメザニンカードベースを有しており、メザニンカードを挿す事によって様々な用途へと拡張していくことが可能です。これらポートと FPGA は 64 の固定入力線、および 64 ペアの LVDS 線で接続されており、最大 128 ch の入力を取り扱うことが出来ます。このうち、64 ペアの差動線がメザニンベースと直接接続されているため、メザニンカードとは自由に入出力を行うことが可能です。

HUL は TUL に存在しなかったデータ通信インターフェース（SiTCP）とトリガー入出力バス（KEK-VME J0）を有します。SiTCP は FPGA を用いた TCP/IP 技術のハードウェア的実装であり、HUL はイーサネットケーブルによる 1 Gbps の TCP 通信をサポートします。また、UDP によるスローコントロールも SiTCP によりサポートされます。KEK-VME J0 バスは M-LVDS 8 ペアによるトリガー配布用バスであり、7 ペアの送信線、1 ペアの出力（BUSY）線によって構成されます。HUL はこの J0 バスのドライバ（バスコントローラ）としての役割と、レシーバとしての両方の役割をこなすことが可能です

HUL は Open-It project 「Hadron Universal Logic Module」において開発されたエレクトロニクスです。本回路は Open-It の技術提供を受けています。

http://openit.kek.jp/project/HUL/HUL

1. ハードウェア

この章では HUL controller と HUL Mezzanine card について機械的、電気的特性について述べます

1.1 HUL controller

Hadron Universal Logic (HUL) controller は VME 6U 規格の回路基板です。

ジーエヌディー管理番号：GN-1573-1.

仕様詳細

・入力ポート

差動入力 64 ペア（KEL 8831E コネクタ）

➔ LVDS・ESL・PECL・LVPECL 等サポート

NIM 入力 4 ポート

・出力ポート

NIM 出力 4 ポート

・通信インターフェース

RJ45 1 ポート

➔ GbE サポート（SiTCP による機能）

・FPGA

Xilinx Kintex7 (XC7K160T-1FBG676C)

・コンフィギュレーションメモリ記録媒体

Micron 128 Mb SPI flash 3.3V 用（N25Q128A13EF840E）

➔ SPI (シリアル) コンフィギュレーションモード

・発振器（クロックソース）

50 MHz LVCMOS (~50 ppm)

➔ Peak-To-Peak jitter 30ps

・トリガーバス

KEK-VME J0

➔ ドライバ機能とレシーバ機能の選択可能

・電源

DC +5V

➔ AC/DC アダプタ、もしくは VME-J1 コネクタから給電

➔ (VME J1 は通信機能を有しないので注意)

・消費電力

静的 0.5~0.7 W @3.3 V (主に PHY)、0.5~0.7 W @1.0 V (主に FPGA コア)

動的 FPGA firmware に強く依存

・メザニンスロット

2 スロット

・メザニンベースコネクタ

双方向 LVDS 32 ペアにて FPGA と直接接続

・メザニン電源

HUL controller から+3.3V 給電

HUL controller を図 1 に示します。ここでは各部基板上の部品について説明します。

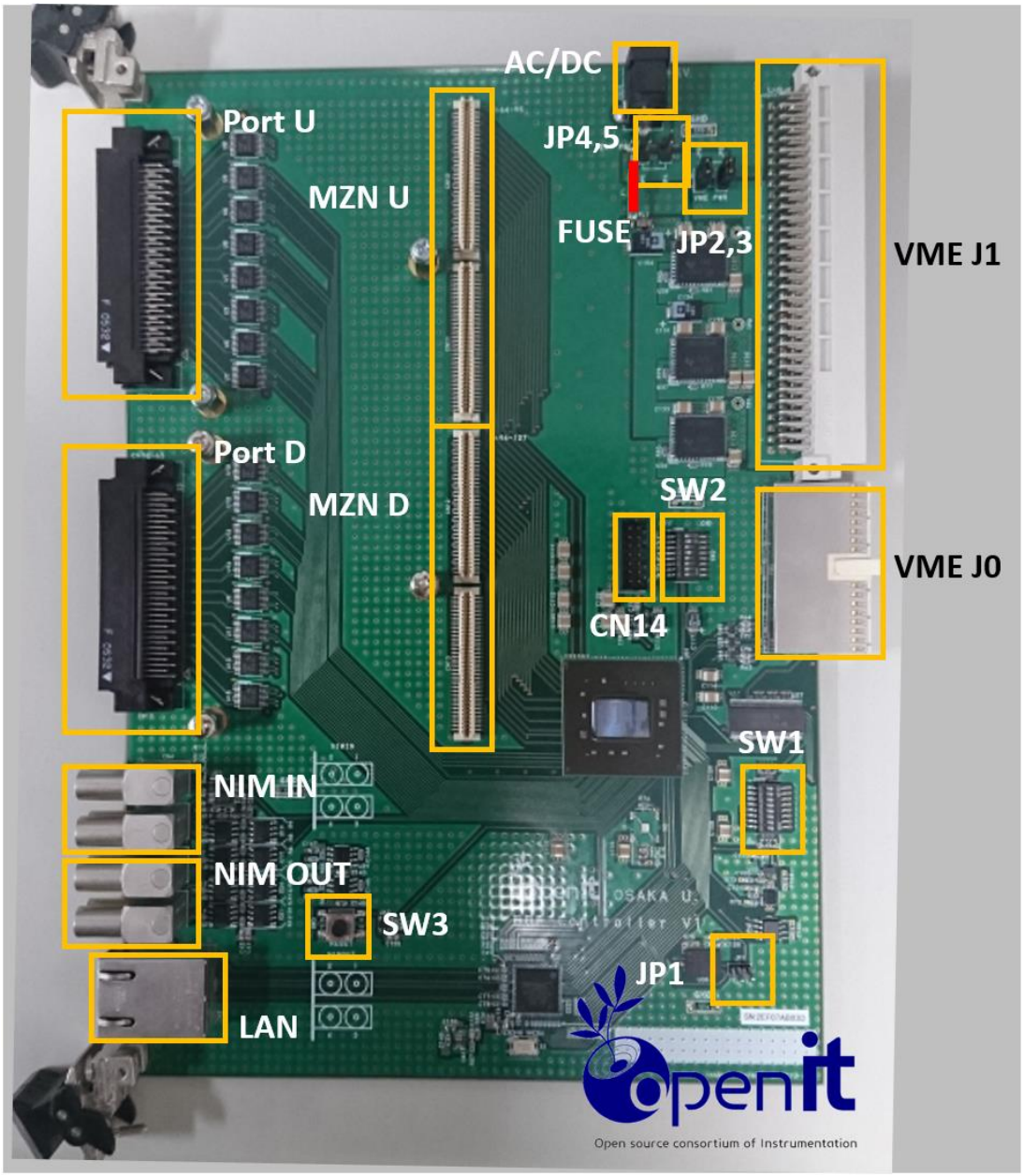


図 1：HUL controller

Port U

固定入力ポート Up です。コネクタはハーフピッチの 68 極コネクタ（KEL 8831E-068-170L-F）です。適合するコネクタでケーブルを作成して接続してください。チャンネルアサインは 0－31 ch で、基準マーカー側が若い番号になります。GND 接続は A1A2、および B1B2 の 2 ペア（基準マーカー直下）です。コモンモードレンジで－4V から+5V までの差動信号をサポートします。LVDS・ECL・PECL・LVPECL などがサポート対象です。**ECL に代表されるエミッタフォロワ型の信号規格を利用する場合、HUL 側に接地抵抗がない事に注意してください。**ドライバ側で電流制御されている必要があります。

固定入力ポートでは差動信号を LVCMOS に変換してから FPGA へ入力します。そのため、コモンモードノイズの影響を受けるので、DCRv1 よりも時間分解能が出ない可能性があります。

Port D

固定入力ポート Down です。電気的、機械的特性は Port U と同様です。チャンネルアサインは 32－63 ch で、基準マーカー側が若い番号になります。GND 接続は A1A2、および B1B2 の 2 ペア（基準マーカー直下）です。

NIM IN

4 入力の NIM 規格ポートです。チャンネルアサインは基板上シルクを参照してください。

NIM OUT

4 出力の NIM 規格ポートです。チャンネルアサインは基板上シルクを参照してください。

LAN

RJ45 のイーサネットコネクタです。1 Gbps までの TCP/IP 通信をサポートします。

MZN U

メザニンカードのベースコネクタ Up です。HUL controller 側のコネクタは MOLEX-071439-0464 64 極です。2 つのコネクタのうち上側が信号線、下側に電源と GND がアサインされています。適合するメザニン側のコネクタはかみ合わせの高さにより複数存在しますが、**MOLEX-071436-1464 を採用した場合 9 mm の足+0.5mm のワッシャーで支える事になります。**

信号線は 64 ペアの差動線で FPGA とつながっています。メザニンベースコネクタまでは信号方向が定まっていないため、双方向 LVDS が利用できます。実際にどのように入出力を行うかはメザニンの設計と FPGA 側の設計に依存します。MZN U に入力ポートを割り当てた場合、チャンネルアサインは 64－95 ch になります。メザニンベースと FPGA 間の差動配線はパターンレイアウトの都合でいくつか p/n が反転しています。**反転している箇所にインバータを挿入してユーザーが使える状態にするためには MZN_NetAssign.vhd を通す必要があります。**

メザニンベースへの供給電源は HUL controller からの+3.3V になります。現状、HUL controller 上の+3.3V 電源容量は 4A であり、controller 上で 1A 程度消費しているため、猶予は 3A（10W）程度です。

MZN D

メザニンカードのベースコネクタ Down です。電気的、機械的特性は MZN U と同様です。MZN D に入力ポートを割り当てた場合のチャンネルアサインは 96－127 ch です。MZN U と同様で p/n 反転している配線が存在し、MZN_NetAssign.vhd を通す必要があります。U と D では反転箇所が異なるため、どちらのメザニンベースの信号かは明確に区別する必要があります。

CN14

JTAG ダウンロードケーブル用のコネクタです。Xilinx の USB ダウンローダーに対応しています。サードパーティ品については保障外です。bitstream や MCS のダウンロード方法については 2 章で述べます。

SW1

VME J0 バスの入出力方向を決める Dip SW です。HUL controller が J0 バスドライバになる場合、1－8 番すべてを ON にします。 レシーバ状態（Default）にする場合、1－8 番全てを OFF にします。1 つのクレートに 2 つドライバ存在するとショートするので、気をつけて運用してください。

SW2

ユーザー用の Dip SW です。各スイッチの役割は FPGA firmware に依存します。

SW3

ユーザー用のスイッチです。押すと一定幅のパルスを FPGA に送ります。役割は FPGA firmware に依存します。押した際のレベル遷移は 1→0 です。(負論理)

JP1

SPI flash へ SiTCP 経由で MCS をダウンロードする場合、JP1 をショートさせておく必要があります。

JP2, 3

VME J1 から給電を受ける場合両方をショートさせます。HUL controller の一次電源は DC +5V です。

JP4, 5

AC/DC アダプタから給電を受ける場合両方をショートさせます。

AC/DC

一般的なプラグ（外径：5.5mmΦ、内径：2.1mmΦ）の AC/DC アダプタの接続口です。

FUSE

容量 5A の FUSE です。替えの FUSE は例えばリテルヒューズの PICO® ヒューズなどを購入して使ってください。

VME J1

VME クレートの J1 コネクタです。電源の+5V ラインのみが接続されています。

VME J0

KEK-VME クレートの J0 バスコネクタです。J0 バスの電源へは接続していません。標準装備ですが取り外すこともできます。外しても J0 バスが使えなくなるだけで機能に違いはありません。注文時に CN9 をはずす様に指示してください。

1.2 HUL Mezzanine cards

製造済みの Mezzanine card について述べます。

1.2.1 HUL Mezzanine Drift Chamber Receiver (DCR) Ver.1

Drift Chamber Receiver (DCR) Ver.1 はドリフトチェンバー用の ASD カード（ジーエヌディー管理番号：GNA200）の出力信号をバッファして、FPGA へ入力するための信号入力用回路基板です。機械的、電気的には 32 ch 入力の LVDS 信号リピート基板が正しい呼称になります。Ver2 と違い全経路で差動信号を維持し、かつ LVDS リピータ IC の性能もよいため、時間分解能の点では Ver2 よりも有利です。同様の理由で HUL controller の入力ポートよりも有利です。

ジーエヌディー管理番号：GN-1573-S1

仕様詳細

・入力ポート

差動入力 32 ペア（KEL 8831E コネクタ）

➔ LVDS（もしくは同等の信号レベル)をサポート

・出力ポート

なし

・通信インターフェース

HUL controller と LVDS で接続される

・電源

DC +3.3V

➔ HUL controller からベースコネクタ経由で給電

DCR v1 を図 2 に示し、各部詳細を説明します。

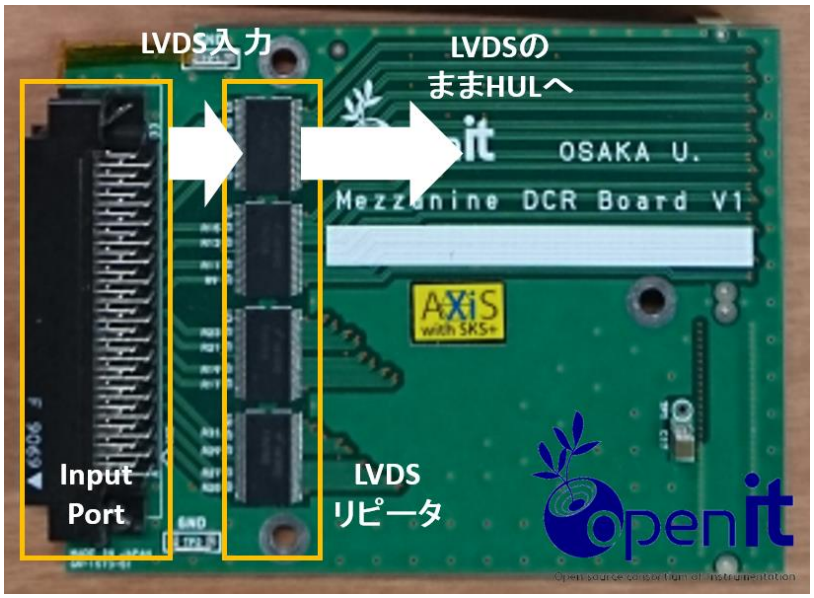


図 2：DCR ver1

Input Port

LVDS 入力ポートです。コネクタはハーフピッチの 68 極コネクタ（KEL 8831E-068-170L-F）です。適合するコネクタでケーブルを作成して接続してください。HUL controller の入力ポート同様、基準マーカー側が若い番号になります。GND 接続は A1A2、および B1B2 の 2 ペア（基準マーカー直下）です。DCR は v1、v2 共に基板の配線パターンの都合で、見た目上のチャンネルの並びから配線パターンがスワップされています。また、HUL controller 上の配線の p/n 反転箇所も考慮に入れなければいけません。**そのため、FPGA firmware でネットアサインを見た目の番号どおり並べ替えるためには、図 3 に示すように DCR_NetAssign.vhd を通す必要があります。**

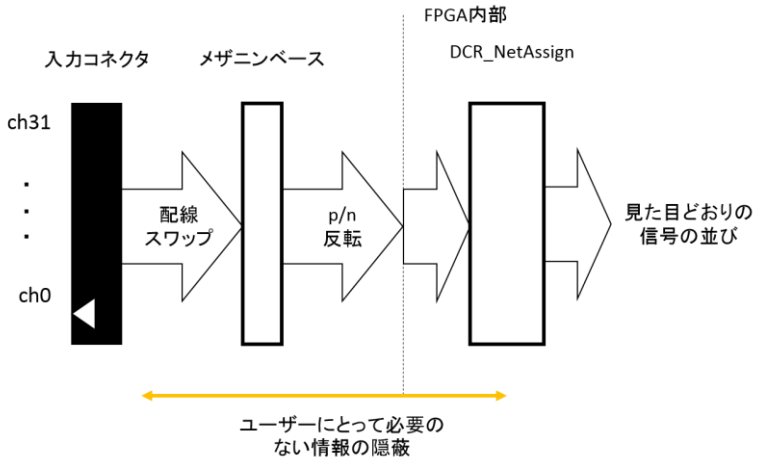


図 3：DCR_NetAssign.vhd の概略

LVDS リピータ

LVDS を受信してそのままリピートする IC です。FPGA 保護の意味合いで一度信号を中継します。この IC の出力信号が FPGA へ入力されます。

1.2.2 HUL Mezzanine Drift Chamber Receiver (DCR) Ver.2

Drift Chamber Receiver (DCR) Ver.2 は機能的には Ver.1 と同様ですが、差動信号レシーバ IC に HUL controller と同じ IC を用いているため、様々な差動信号を受信することができる回路基板です。

ジーエヌディー管理番号：GN-1626-1

仕様詳細

・入力ポート

差動入力 32 ペア（KEL 8831E コネクタ）

➔ LVDS・ESL・PECL・LVPECL 等サポート

・出力ポート

なし

・通信インターフェース

HUL controller と LVDS で接続される

・電源

DC +3.3V

➔ HUL controller からベースコネクタ経由で給電

図 4 に DCR v2 を示し、各部詳細について述べます。

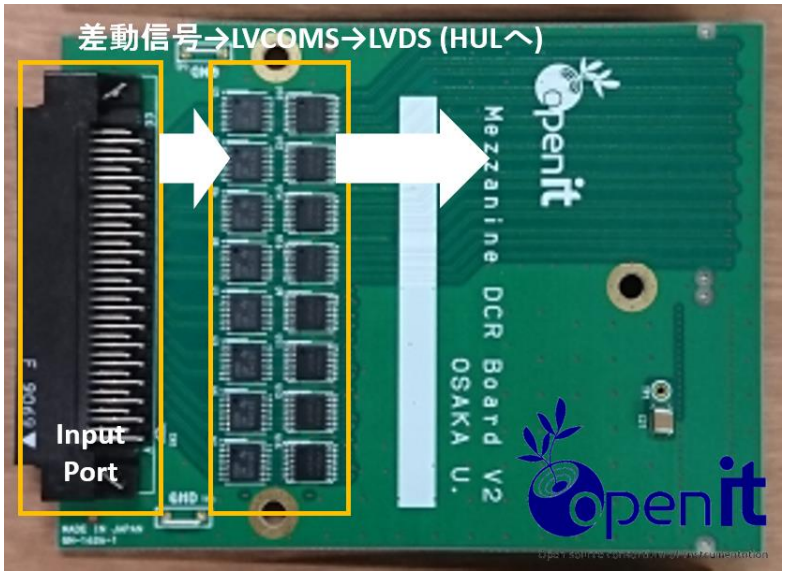


図 4：DCR ver.2

Input Port

機械的には DCR Ver.1 と同様です。電気的にはコモンモードで－4V から+5V までの差動信号入力をサポートします。DCR Ver.1 と同様に、FPGA Firmware でネットアサインを並び替えるために、DCR_NetAssign.vhd が必要です。

1.2.3 HUL Receiver Module (HRM)

Receiver Module は J-PARC K1.8 ビームライン等で利用されている Triger Module (TRM) によって配布されたトリガー信号やイベント番号を受信し、TRM へ BUSY 信号などを返送する回路基板になります。基本的な機能は GPIO RM と同様です。HRM は Cat5e などのツイストペアケーブルで送られてくるトリガー信号やイベント番号をデシリアライズし、ユーザー (FPGA) が扱いやすいバス信号形式変換して FPGA へ信号を渡します。また、FPGA から送られてくる BUSY と RSV2 をツイストペアケーブルで TRM へ送信するために信号変換します。HRM が受信したトリガー信号や HRM へ渡す BUSY 信号などの処理はすべて FPGA で行います。本回路を取り付けることで、HUL cotroller は KEK-VME クレートの J0 バスコントローラとして機能するようになります。

ジーエヌディー管理番号：GN-1627-1

仕様詳細

・トリガーポート

RJ45 2 ポート

・通信インターフェース

HUL controller と LVDS で接続される

・電源

DC +3.3V

➔ HUL controller からベースコネクタ経由で給電

図 5 に HRM を示し、各部詳細を説明します。

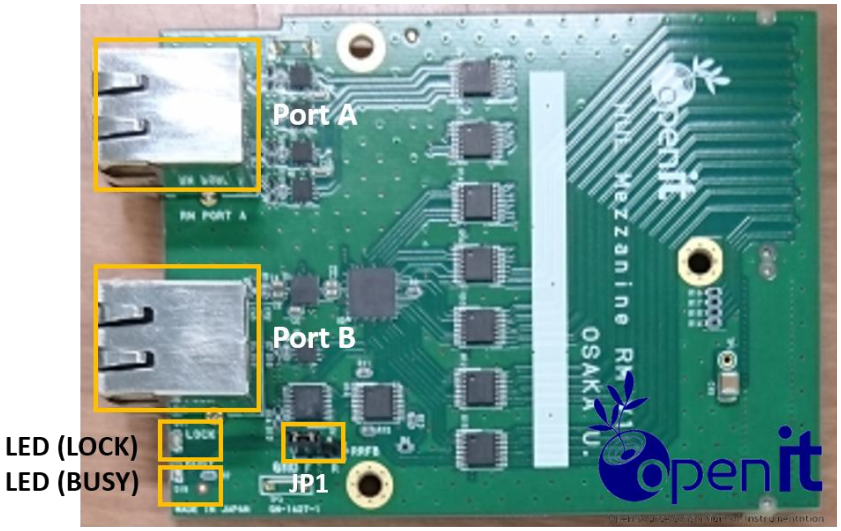


図 5：HRM

Port A

トリガー入力ポート A です。TRM の Port A、もしくはリピータの Port A とツイストペアケーブルで接続します。

Port B

トリガー入力ポート B です。TRM の Port B、もしくはリピータの Port B とツイストペアケーブルで接続します。

LED (LOCK)

Port B から送られてくるイベントタグビットからクロックが正しく復調できている事を示す PLL LOCK ビットの状態を示します。PLL LOCK が HIGH だと緑の LED が点灯します。

LED (BUSY)

TRM へ送信する BUSY 信号の状態を示します。BUSY が HIGH であれば赤の LED が点灯します。ただし、HRM は直接クレートバスを見ていないので、HUL controller の FPGA から正しく BUSY が送られてきている必要があります。

JP1

Port B から送信されてくるイベントタグ情報はシリアル状態で送られてきます。本回路上では受信したシリアルビット列から送信クロックを復調し、イベントタグをデシリアライズします。デシリアライズされたタグ情報はデコーダ IC から出力される RCLK クロックに同期してバスに現れます。JP1 はバス信号がストローブされるタイミングを RCLK の立ち上がりに同期させるか、立下りに同期させるかを決めるジャンパです（RRFB)。R で立ち上がりに同期して、F で立下りに同期してストローブします。RRFB は JP1 でなく FPGA 側で設定することも可能ですが、論理信号的には FPGA

側は弱いHigh、もしくは弱いLow状態であることに注意してください。(電気的にはPoull Up/Down) FPGA 側で自在に変更するためには JP1 は開放である必要があります。ただしその場合は FPGA 側で必ずレベルを設定してください。

長々と書きましたが、通常タグ情報は常に FPGA 側でラッチし続ける必要はなく、Level2 のタイミングで一度 FF へ保存すればよいので、JP1 は R で特に問題ないと思います。

Mezzanine HRM への入出力信号

HRM からの入力信号

RCLK

受信したイベントタグ情報から復元したクロックです。立ち上がりか立ち下がりエッジのタイミングでタグがバスへストローブされます (RRFB によります)。

LEVEL1

Level1 trigger 信号です。

LEVEL2

Level2 trigger 信号です。

Clear

Fast clear 信号です。

RSV1

TRM から送られてくる Reserve 1 信号です。

LOCK

PLL の Lock ビットです。受信したタグからクロックが正しく復元できていることを示します。

SNINC

Spill Number Increment です。この信号を使って FPGA 内部でスピル数を数えることが可能です。

Event counter

TRM から送られてくる 12 ビットのイベント番号です。

Spill counter

TRM から送られてくる 8 ビットのスピル番号です。

HRM への出力信号

RRFB

タグ情報がストローブされるタイミングを決めます。High だと RCLK の立ち上り、Low だと立下りでストローブされます。弱い論理信号のため、基板上の JP1 で上書きされます。

BUSY

TRM へ送る BUSY 信号です。

RSV2

TRM へ送る Reserve 2 信号です。

1.2.4 HUL Mezzanine differential signal transmitter LVDS (DTL)

Mezzanine DTL は HUL 本体の FPGA からメザニンカードを通じて外部へ信号を出力するための拡張カードです。回路的には殆ど DCRv1 と同一で、LVDS リピータの入出力の向きが逆になったものです。HUL 本体との信号接続の並びは DCRv1 と同じです。HUL からは LVDS で信号を出力してください。

ジーエヌディー管理番号：GN-1724-1

仕様詳細

・入力ポート

なし

・出力ポート

LVDS 32 ペア（KEL 8831E コネクタ）

➔ LVDS 規格による信号出力。

・通信インターフェース

HUL controller と LVDS で接続される

・電源

DC +3.3V

➔ HUL controller からベースコネクタ経由で給電

DTL を図 6 に示し、各部詳細を説明します。

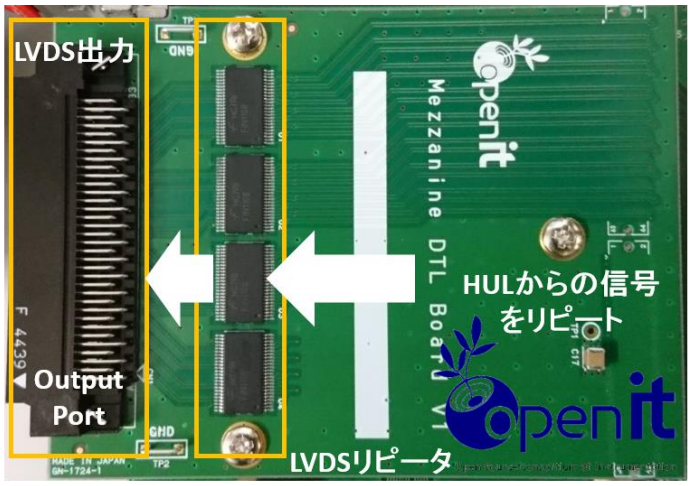


図 6：Mezzanine DTL の写真。

Output Port

LVDS 出力ポートです。信号の並びや GND の位置等、機械的には DCRv1 と同一です。また、**また、本メザニンを使用する際にも DCR_NetAssign.vhd を通す必要がありますが、DCRv1 の場合と入出力の向きが逆になる点に気を付けてください。**

1.2.5 HUL Mezzanine High-resolution TDC

Mezzanine HR-TDC は 30 ps の精度で時間を測定する FPGA high-resolution TDC です。TDC の種別としては、common stop 型の high-resolution-multi-hit TDC となります。メザニンカード上に 1 つ時間計測専用の FPGA を持ち、HUL 本体からの制御とデータ吸出しを必要とします。一方で、本メザニンはトリガー入力により時間を測定し決まったフォーマットで HUL 側の FPGA へ転送する機能しか持たないため、HUL 側の FPGA の記述によっては単純な TDC のみならず、high level trigger など他の方法で利用することが可能です。詳しい動作や制御方法は第 3 章で述べます。

本メザニン上の FPGA は HUL 本体と同じ Kintex-7 160T-1 です。メザニン 1 台で 32ch 分の時間測定（leading/trailing 両エッジ）が可能です。**入力は LVDS のみであり、ECL はサポートしない点に注意してください。**FPGA 用のクロックは基板上の発振器と HUL からのクロックが選択可能です。電源は HUL 本体からの 3.3V によって賄われ、本メザニン上で必要な電位は全て LDO で生成されます。後述するように、メザニンあたり 5W と割と電力を必要とします。HUL 本体上の 3.3V 電源が LMZ30604RKGT（4A 電源）の場合、2 スロットに本メザニンを挿すと電流不足で正しく動作しません。**必ず LMZ30606RKGT（6A 電源）の HUL 基板を利用してください。**また、**2 スロット搭載時はシステム全体でおよそ 20 W 電力を消費します。すごく発熱するので必ず空冷してください。熱を持つと FPGA のリーク電流が爆発的に増えて、予期せぬトラブルを起こすと思われます。**クレートに挿して利用することを強くお勧めします。（裸では GND が安定せず性能が出ません。）

ジーエヌディー管理番号：GN-1644-1

仕様詳細

・入力ポート

LVDS 32 ペア（KEL 8831E コネクタ）

・出力ポート

なし

・通信インターフェース

HUL controller と LVDS で制御及びデータ送信を行う。

- Mezzanine Interface (MIF)を利用した非同期通信。
- 128 MHz の DDR 通信 5 線を用いた、1 Gbps でのデータ転送。

・電源

DC +3.3V

静的電力のみで 5W 消費。

Mezzanine HR-TDC を図 7 に示し、各部詳細を説明します。

Input Port

信号入力ポートです。LVDS 規格のみサポートします。コネクタはハーフピッチの 68 極コネクタ（KEL 8831E-068-170L-F）です。GND 接続は A1A2、および B1B2 の 2 ペア（基準マーカー直下）です。チャンネル番号は基準マーカー側から見た目通り 0 から 31 までです、チャンネルのスワップはありません。

CN4

Xilinx ダウンロードケーブル用のコネクタです。FPGA と SPI flash memory のアクセスが可能です。**本メザニンの SPI flash は 1.8V 用の N25Q128A11EF840E です。**HUL 本体の IC とは異なりますので、MCS ダウンロード時には気を付けてください。

SW1

基板上の FPGA を制御するための Dip スイッチです。現状（v2.5 と v3.2）では FPGA 利用するクロックを基板上の発振器か HUL からのクロックにするかを選ぶことが出来ます。

X1

100 MHz の発振器です。HUL からクロックを送信する場合、周波数は同じである必要があります。

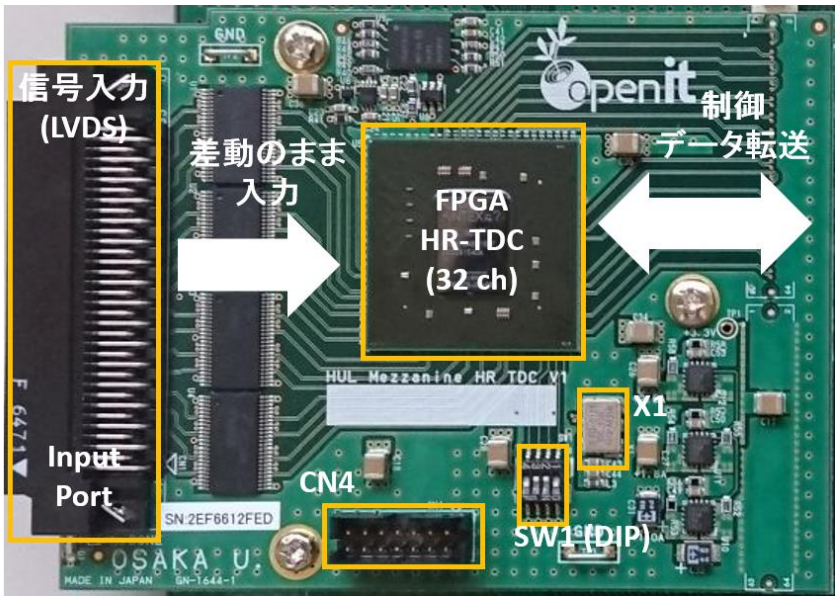


図 7：Mezzanine HR-TDC。

2 FPGA の記述について

この章では FPGA firmware 開発者向けの情報を説明します。

Vivado について

HUL は FPGA に Kintex7 を採用しているため、Xilinx Vivado を用いて開発を行うことが推奨されます。（ISE はもうやめましょう）搭載されている Kintex7 160T は有償のライセンスでなくとも開発が可能なサイズの FPGA のため、無償版の Vivado Design Suite WebPACK を用いて開発を行うことが出来ます。本多は Windows 上で WebPACK を使って開発を行っています。Version 2016.1 から大幅改良されて、不具合の残っていた部分はほぼ解消されました。

Xilinx の開発環境は基本的に既知のバグを含んだ状態でリリースされます。そのため、Vivado の新しいバージョンが出たら積極的に更新するべきですが、いつ行うかは慎重に判断してください。KEK の内田さんいわく ISE のころに比べ Vivado は格段にバージョンアップに対する安定度が増して、基本的に新しいバージョンにクリティカルなバグが報告されていない限り移行したほうがよいとの事ですが、タイミング的に際どい調整をしている回路では合成結果が変わる可能性があるため、開発途上でバージョン更新をすることはお勧めしません。なので、**Vivado は積極的にバージョン更新させるが、動作確認できていない Firmware がある状態では行わないほうがよい、というのが基本方針になると思います。**

Vivado は合成時に大量のメモリを消費します。最低でも 8GB 以上、複数 RUN を同時に走らせる場合 16GB 以上の物理メモリが搭載されていることが望ましいです。また、複雑な Firmware では合成に時間がかかります。例えば、HUL MHTDC は論理合成から配置配線完了まで ThinkPad X230 (Core i7-3520M (2.9 GHz)搭載) 上で物理メモリを 3 GB 消費して約 30 分かかります。単純に CPU パワーとメモリ量が合成にかかる時間を短縮するので、ラップトップ PC で開発するのか？それとも専用のワークサーバー上で行うのか？は自分にとっての利便性と相談して決めてください。Xilinx の開発ツールは本来 CUI で使うことが想定されており、Vivado も TCL と呼ばれるスクリプト言語で合成フローを制御できます。そのため、Makefile など書いてしまえば Linux サーバで並列にジョブを処理することも可能です。TUL の頃と比べ巨大な firmware を開発可能になっているので、安いラップトップ PC に Windows を入れて済ませようというのはお勧めしません。

Vivado project に関して

ターゲット FPGA	xc7k160tfbg676-1
使用言語	VHDL (IP を生成した際のテンプレートに記述される言語になります。) (言語は混在可能なので追記は他言語でも可。ただし既存モジュールはラップする必要あり。)
デフォルトライブラリ名	mylib (何故か xil_default_lib に不具合があるため。)
合成のストラテジー	Vivado Synthesize Default

命名規則

守る必要はありませんが、以下の点に気をつけるとコードの見た目が本多の作ったものとそろいます。

・プロジェクトの最上位ソースは toplevel.vhd もしくは toplevel_hoge.vhd とする。

・インスタンス化の際には u_で始める。toplevel の場合さらに u_hoge_Inst とする。

・Entity のポート名はアンダーバーを含まない。一方モジュールの内部信号にはアンダーバーを付与するか全て小文字で書き、モジュールポートと明確に区別する。ただし、toplevel.vhd の Entity ポートのみ全て大文字＋アンダーバーとする。

合成フロー

Vivado を使った開発については他のドキュメントを参考にしてください。

SiTCP や IP catalog を使って生成した IP ライブラリの移植について

SiTCP は ngc 化されているため、HUL_skelton 内部で使われている形式でほかのプロジェクトで再利用するためには、ngc とそのラップソースをコピーする必要があります。具体的には HUL_skelton/HUL_Skelton/srcs/source_1/以下にある、SiTCP ディレクトリをコピーしてディレクトリごと Add source する必要があります。また、BBT から最新のライブラリを持ってきた場合、同じファイルをコピーして Add source してください。

IP catalog で生成した IP に対しては Vivado から IP container という機能が利用できるようになりました。これまで IP を生成すると対応するディレクトリが生成されその中に細かいファイルがたくさん生成されましたが、Vivado ではこれをアーカイブすることができます。Vivado の画面で IP source を選択して、Enable IP container を選択すると source_1/ip 以下に xcix 形式のファイルが現れます。これをコピーして Add source の Add existing block design source から移植することができます。

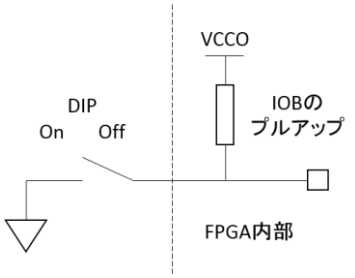
2.1 Toplevel port

FPGA の top level entity のポート（つまり FPGA の足）の説明を行います。

System

信号名	信号規格	
CLKOSC	LVCMOS33	基板上の発振器からの 50 MHz クロック入力です。
PROG_B_ON	LVCMOS33	この信号が Low になると FPGA は SPI Flash から自分自身を再コンフィグレーションします。FPGA の動作がおかしい時などに利用し、電源の On/Off に相当する強力なコマンドです。 通常時は High、もしくは弱い High でないといけません。電源投入時に High 状態を維持できていないとコンフィギュレーションが終わらないので注意が必要です。

User I/O

信号名	信号規格	
LED	LVCMOS33	フロントパネル上部に取り付けられた 4 つの LED につながっています。
DIP	LVCMOS33	DIP スイッチの入力信号です。この信号は負論理になっており、電気的には下図のようになっています。High 状態を作るためには IOB のプルアップが必要なので、忘れずに設定してください。 
USER_RST_B	LVCMOS33	SW3 を押した際の 1 ms 程度のパルス入力です。この信号は負論理です。
NIMIN	LVCMOS33	フロントパネルの NIM 入力信号です。
NIMOUT	LVCMOS33	フロントパネルの NIM 出力信号です。

PHY・EEPROM

サンプルプロジェクトで PHY と EEPROM にカテゴライズされている信号は他のプロジェクトでもその通りに接続してください。また、PHY を未初期化にしないためにも SiTCP は全てのプロジェクトで実装したほうがよいでしょう。

フロント差動入力

信号名	信号規格	
FIXED_SIGIN_U	LVCMOS33	フロントパネルの固定信号入力線、上側のコネクタです。
FIXED_SIGIN_D	LVCMOS33	フロントパネルの固定信号入力線、下側のコネクタです。

メザニンベース

信号名	信号規格	
MZN_SIG_Up/n	VCCO=1.8V までの差動・シングルエンド 双方向	メザニンコベースコネクタ上側に接続されて信号線です。この信号線の信号規格、入出力方向はメザニンカードによって決まることに注意しなければいけません。これらの信号線は HP バンクに配線されているので VCCO は 1.8V です。そのため、HP バンクで利用可能な差動信号、および 1.8V までのシングルエンド信号がサポートされますが、特別な理由がない限り LVDS 信号規格を利用してください。

		入力なのか出力なのか、はたまた双方向ポートを使うのか、浮き配線の処理などは Firmware 開発者が適切に行ってください。
MZN_SIG_Dp/n	VCCO=1.8V までの差動・シングルエンド 双方向	メザニンコベースコネクタ下側に接続されて信号線です。

J0 バス信号

信号名	信号規格	
J0RS	LVCMOS33	J0 バスの S1-7 番を読み出しポートとです。HUL controller が J0 バスに対してスレーブの場合使用します。この信号を使って TRM からのトリガーやタグ情報を受け取ります。
J0DS	LVCMOS33	J0 バスの S1-7 番を駆動するポートです。HUL controller が J0 バスに対してマスタである場合に使用します。
J0RC	LVCMOS33	J0 バスの C1-2 信号を読み出すポートです。J0C 線は BUSY 処理専用です。J0 バスでは Low で BUSY となります。C1 と 2 は全く同じ信号が流れているはずなので、モジュール内部では C1 と 2 を OR して使ってください。この線は HUL controller が J0 バスに対してバスマスタの場合に利用します。
J0DC	LVCMOD33	J0 バスの C1-2 信号を駆動するポートです。J0C 線はオープンコレクタ OR になっており、Low で BUSY を示します。C1 と 2 には同様の信号を入力してください。この信号線は HUL controller が J0 バスに対してスレーブである場合に利用します。

FPGA 内部の J0 バスの信号線と基板上の信号パターンは図 8 のような関係になっています。

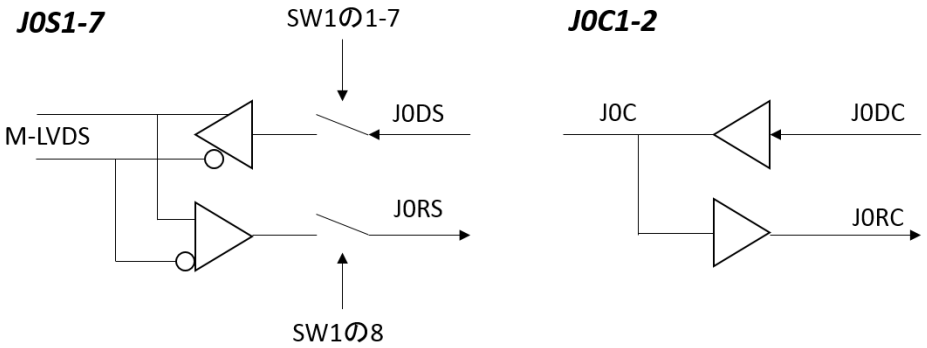


図 8：J0 バスに対する FPGA と基板間の関係。

2.2 SiTCP

SiTCP は KEK の内田さんによって開発された技術で、本来 CPU が必要な TCP/IP 通信をハードウェアのみで実現した技術になります。詳しい使い方や最新の IP コアは内田さんの web site や BBT で入手してください。

内田さんの web site　：http://research.kek.jp/people/uchida/technologies/SiTCP/

BBT の web site　：https://sitcp.bbtech.co.jp/

SiTCP 動作の概略は図 9 のようになります。SiTCP は起動時に PHY を初期化して、EEPROM から MAC アドレスと IP address をロードします。Force default で起動すると SiTCP が持つ Default IP address (192.168.10.16)と MAC アドレスがロードされます。Default 状態では他の機器が存在するネットワーク空間では使えないので PC と一対一で使うことになります。FPGA 内部の回路へ提供する通信機能は TCP と UDP になります。TCP は 100 Mbsp と 1 Gbps の両方の通信をサポートし、PHY からの信号により自動で切り替わるように設定することが可能ですが、HUL controller では 100 Mbps か 1Gbps かは固定、もしくは DIP による選択しかできません。TCP 通信はシステムクロックに同期した 8bit 送信と 8bit 受信を行うことができますが、TCP によるデータ受信は推奨されていません。レジスタの設定などは UDP 通信を使うことが望ましいです。UDP 通信は RBCP と言う独自のパケットを用いて制御コマンドの送付、アドレス送付、デー送受信を行う機能を提供します。RBCP は Acknowledge を返すことによって PC と SiTCP でハイドシェイク通信を行っています。UDP 通信はシステムクロックに同期して 32bit のアドレス送付、8bit のデータ送受信を行います。また、特定のアドレスを指定することで SiTCP の内部レジスタや EEPROM に直接アクセスすることが可能ですが、特殊な要求がない限り変更する必要はないと思います。**ただ、SiTCP はデフォルトだと keep alive (heart beat) しない設定となっているので、極端にトリガーレートが低い場合 keep alive packet の送信を行うように設定しないとセッションが閉じてしまいます。**また、直接 SiTCP のレジスタにアクセスすることで後述する BBT のソフトウェアを用いずとも IP address の変更を行うこともできます。

SiTCP のコアには 2 種類のクロックを供給する必要があります。1 つはシステムクロックでこのクロックに同期して SiTCP は TCP と UDP のデータを送受信します。**システムクロックには下限の周波数があり、100 Mbps 時は最低 25 MHz・30 MHz 以上推奨、GbE 時は最低 125MHz・130MHz 以上推奨となります。**もうひとつは PHY をデータ通信する際のクロックです。100 Mbps 時は PHY から送られてくる tx clk を接続してください。GbE 時は FPGA 内部などで生成した 125 MHz (gtx clk) のクロックを接続してください。**Gtx clk は良い精度で 125 MHz である必要があります。HUL controller で使用している PHY では 100 ppm しか猶予がないので、間違えずに設定してください。**

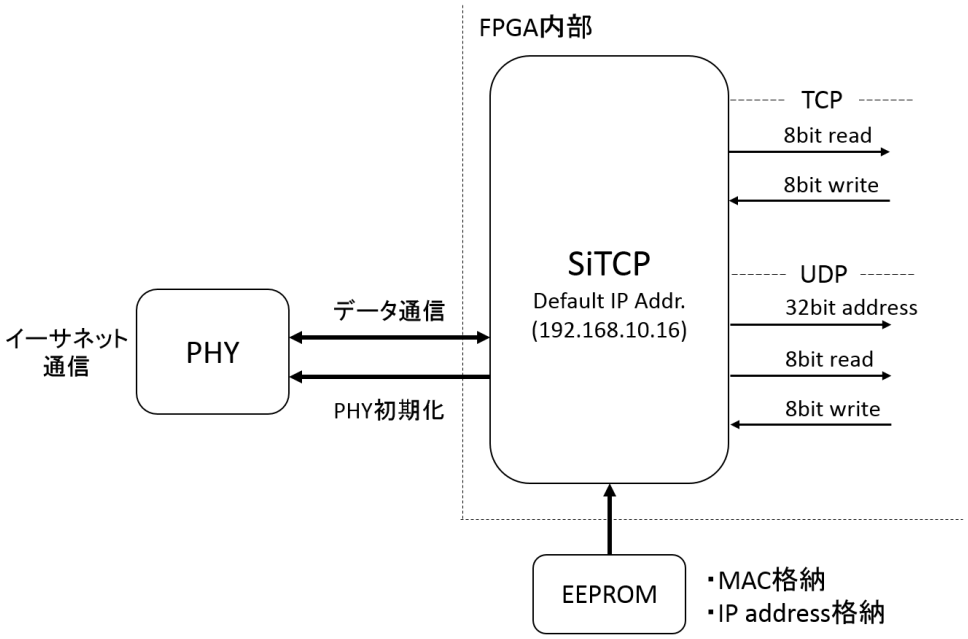


図 9：SiTCP 機能の概略図

IP address と MAC の設定方法

IP address と MAC は EEPROM 上に格納されており、SiTCP を経由して書き込むことが前提となっています。そのため、最低でも SiTCP が動く状態の Firmware を作らなければいけません。また、これらの書き込みは RBCP 通信を使って自分で行うこともできますが、BBT の要したツールを用いるのがいいでしょう。これらのツールは前述の BBT のサイトからユーザー登録をすることによって入手できます。

MAC は“SiTCPMpcWrite”を用いて書き込みます。現状（たぶん）Windows 版のバイナリしかないようです。ツールを立ち上げるたらジー・エヌ・ディーから納品された DVD ディスクにあるファイルを選択して書き込んでください。（MAC は直接 BBT から買うこともできますが、シリアル番号の管理等を発注者がやらないといけなくなる上に本多が把握できなくなるのでやめてください。）MAC は機器固有の Physical Address のため、衝突させると動かなくなります。どのボードにどの MAC を入れたかはログをとっておくのがいいでしょう。

IP address は SiTCP Utility を用いて書き換えを行います。このツールはマルチプラットフォーム向け（β版）に用意されているようです。この作業は MAC を入れた後行います。まず、図 10 に示す赤枠で囲んだ EEPROM にアクセスするにチェックを忘れずに入れてください。次に制御対象の IP address を入力して表示ボタンを押してください。（もし、IP address がわからない場合 force default で起動してください。）右上の目が動いて情報が表示されたら読めています。次に、書換情報の IP address を設定したい値にして書き換えを押してください。この段階では SiTCP の IP は変わっていません、EEPROM に格納しただけです。ちゃんと書けているかどうかはもう一度読み出してみるとわかります。ここで設定した IP は次回電源投入時から有効です。

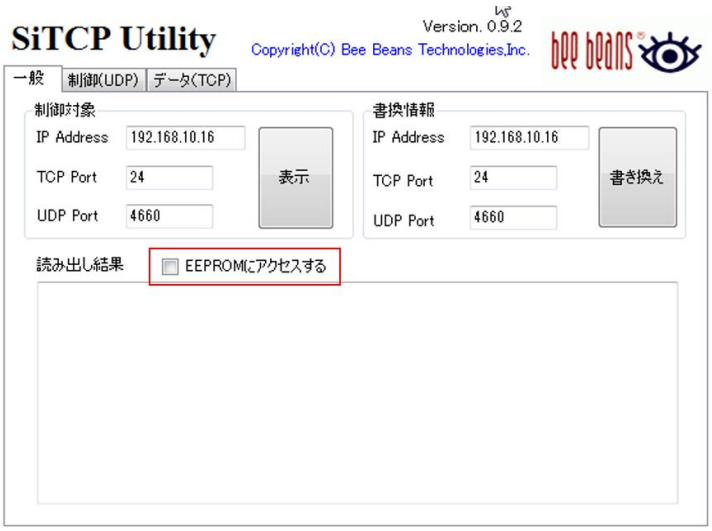


図 10：SiTCP Utility の画面

2.3 スケルトンプロジェクト

ここでは HUL_skelton プロジェクトについて説明します。このプロジェクトは SiTCP を実装しつつ何もしない firmware のほぼ最小構成になります。HUL 用の firmware を作る際のサンプルとして使ってください。機能は 2 つしかなく、入力信号を OR して NIM で出力する機能と、SiTCP もしくは DIP で LED を光らせる機能です。信号の OR では固定入力とメザニン入力（DCRv1 か v2 を想定）をコネクタ単位で OR（32 OR）して、それぞれ 4 つの NIMOUT から出力します。以下では SiTCP を通じて LED を光らせる部分を例にとって FPGA 内部の（Local) Bus Controller について詳しく説明します。

Bus Controller (BCT)

（Local）Bus Controller（BCT）は RCNP の味村さんが書いたソースコードを本多が再利用しています。BCT は外部インターフェース（External Bus）の生のタイミングやデータ線を隠蔽して、FPGA 内部のローカルモジュール管理 (Local Bus) を独立に行うためのバスインターフェースです。External Bus は通信方法によって様々なタイミングやデータ長を持つため、それらの違いを BCT が吸収することで Local Bus 以降のソースコードの再利用性が高まります。また、モジュール追加に対して少ないコードの変更で対応可能です。BCT と Linux ソフトウェアの FPGAModule クラスは対の関係になっており、本多の開発するシステムでは共通ライブラリになります。他者のプロジェクトでもこれらを再利用することを強くお勧めします。

BCT のバス通信シーケンスを図 11 に示します。BCT がバス通信サイクルを始める起点は External Bus（SiTCP UDP）の WE か RE が High となった時点です。この時点で External Bus には有効なアドレスとデータが流れているためそれらを BCT に格納します。この際に、External Bus から受け取った情報を Local Bus 用に再配置します。External Bus のアドレスとデータ、Local Bus のモジュール ID、

ローカルアドレス、およびデータの関係は図 12 の通りです。BCT は GetDest でモジュール ID からこの通信が BCT で内部処理しなければいけないのか、FPGA のほかのモジュールへ接続したいのかを判断します。BCT がターゲットの場合 FPGA の Re-config や User Reset の発行を行ったり、Firmware バージョンを獲得できたりします。他のモジュールがターゲットの場合モジュール ID から接続先のインデックスが決定されます。次に SetBus でアドレスとデータが Local Bus にブロードキャストされ、更に次の Connect において指定したインデックスの WE か RE のみが High 状態になります。ここで BCT は FPGA 内部の指定したモジュールを噛んだ状態になり、モジュールからの応答待ちになります。接続先のモジュールを自分の WE か RE が High になった段階でバスサイクルの処理を始めます。適切な処理を行った後 Ready を High にしてバスサイクルの終了を BCT へ要求します。BCT は Ready を受け取ると終了処理に入り、最後に External Bus に対して Acknowledge を返して 1 つの通信サイクルを終えます。

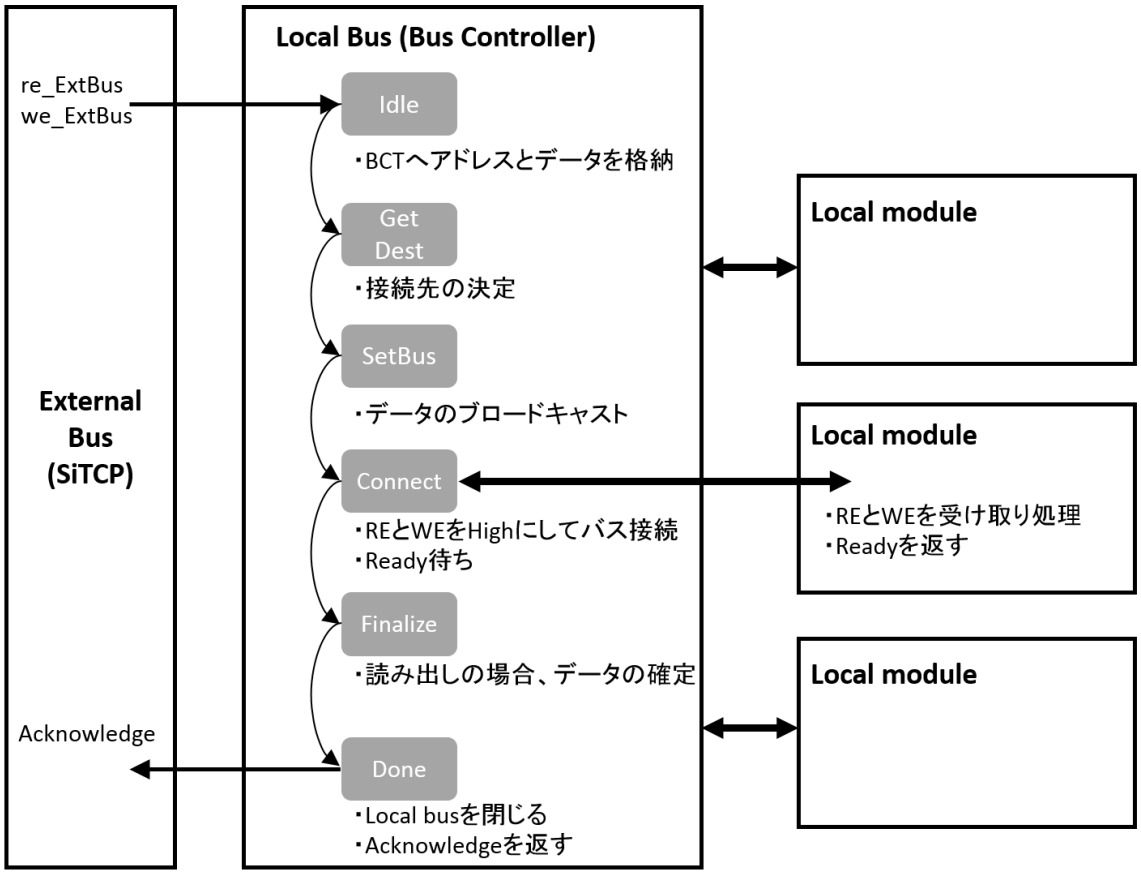
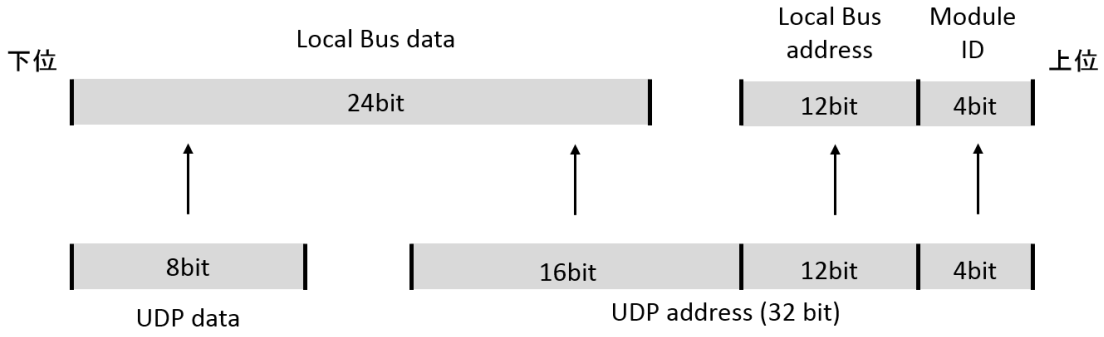


図 11：Local Bus Cycle

前述のように、External Bus と Local Bus の間ではデータとアドレスの図 12 ように再配置を行っています。UDP のアドレスは 32bit 長を持ちますが、そのままだと長すぎるため 3 つの用途に分けて利用します。上位 4bit は Local module のインデックスであるモジュール ID に振り分けられます。この ID の上限までモジュールを追加できますが、0xF が SiTCP、0xE が BCT の予約領域のため、実質的には 14 個までしか追加できません。ただし、そんなに多くのモジュールを 1 つの firmware に入れるようなことは殆どないと思います。次の 12bit はモジュール内部で使われる Local address に振り分けられま

す。最初のアドレスは 0x000 だと思いますが、次のアドレスは 0x010、0x020 と増やすことをお勧めします。下位 4bit をあけておくと、第 4 章で出てくる ReadModule の多バイト読み出し（8bit 以上のレジスタを読み出す際に使用）が利用できるためです。UDP address の下位 16bit は書き込みの場合に限り拡張データ領域として扱われます。書き込みでは 1 サイクルで 24bit まで書き込むことができますが、読み出しでは 8bit までしか 1 サイクルで読めません。

Local moduleへ書き込みの場合



Local moduleから読み出しの場合

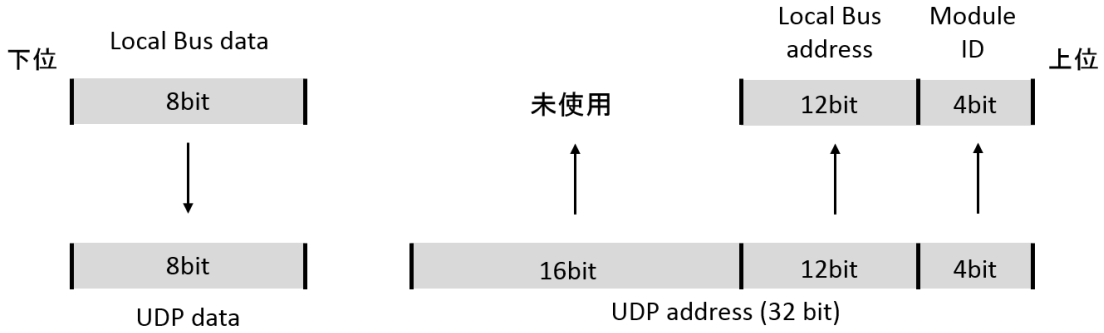


図 12：UDP と Local Bus の間でのデータの再配置。

多バイト読み出し（書き込み）

Local Bus を介した読み出し場合 8bit ずつしか読み出すことができない為、4 バイトのレジスタであれば 4 回 RBCP 通信を行う必要があります。これをなるべくソフトウェア的に簡潔に行うために、FPGAModule クラスには多バイト読み出しの機能が備わっています。その機能を使うためには Local module 内部のステートマシンでレジスタビットとアドレスが以下のような関係を満たしていなければいけません。

4 バイトのレジスタを多バイト読み出しで読む場合

```
when Read =>
  case addrLocalBus(11 downto 4) is
    when addr_register(11 downto 4) =>
      if( addr_register(1 downto 0) = “00” ) then
        dataLocalBusOut  <= reg_hoge(7 downto 0);
      elsif( addr_register(1 downto 0) = “01” ) then
        dataLocalBusOut  <= reg_hoge(15 downto 8);
      elsif( addr_register(1 downto 0) = “10” ) then
        dataLocalBusOut  <= reg_hoge(23 downto 16);
      else
        dataLocalBusOut  <= reg_hoge(31 downto 24);
      end if;
  case ...
  case ...
  end case;
end case;
```

このようにローカルアドレスのマッチングは上位 8bit までで行い、下位ビットは何バイト目かを指す数字として使います。このような for 文をまわすようにソフトウェア側では書かれているので、小さいほうが下位ビットになるようにしてください。同様の方式で 24bit を超える幅のレジスタへも書き込むことができますが、ソフトウェア側で多バイト書き込みはサポートしていないので独自関数をつくってください。

Local Bus へモジュールを追加する

モジュールを追加する場合、次の点を変更してください。

UserParamter.vhd 内部の変数

package AddressMap 以下

NofModules　　: LocalBus にぶら下がっているモジュールの数を変更。BCT は含みません。

mid_hoge　　: Module ID map 欄に追加するモジュールの ID を登録。

LocalAddress　　: モジュール内部で使用するローカルアドレス。3 文字のモジュール名を使って LED_hoge のように記述。

package AddressBook 以下

type binder の行の下に追加するモジュールのインデックスを追加。

constant AddressBook 以下の行に整数を変数に割り当てるコードを追加。

BusController.vhd 内部
ステーマシンで GetDest 状態の else 構文以下で、追加したモジュールの case 文を追加。

toplevel.vhd 内部
モジュールを実体化させて LocalBus を接続。配列のインデックスには AddressBook で生成した変数を代入。

3. 既存のファームウェア

3.0 各ファームウェア共通の事

リセットシーケンス

2017 年 12 月 19 日づけの更新分から各ファームウェアのリセットシーケンスを統一しました。これまでのファームウェアでは BCT がハングアップするとリセットする手段が電源 OFF 以外なかったり、MH-TDC では BCT リセットをかけても一部 FIFO の状態がリセットされず、DAQ がハングしている状態が解除されなかったり、という問題がありました。そこで、リセットのレベルを段階ごとに設定して対処できるようにしました。

リセットを行う方法。（下に行くほど強制力が強く非常性がある。）

- **BCT リセット：**BCT の Address::Reset に RBCP で書き込みを行う。BCT より下流のユーザーモジュールがリセットされる。通常のリセット。
- **SiTCP リセット：**sitcp_controller で Reset_SiTCP を呼び出す。BCT がハングしている場合でもリセット可能。BCT もデッドロック状態から復帰する。
- **ハードリセット：**基板上のボタンを押す。SiTCP を含む全回路がリセットされる。最も強制力がある。

通常は BCT リセットを利用し、操作を誤って BCT がデッドロックした場合は SiTCP リセットを、SiTCP がハングしてしまった場合はハードリセットを、という順番で利用してください。

サポートするネットワーク規格

現在公開しているファームウェアでは、SiTCP は GbE でしか動作しないようになっています。SiTCP 自体は 100 Mbps と GbE の両方をサポートし自動切り替えが可能なのですが、利用している PHY の都合で自動切り替えの機能を意図的に殺しています。そのため、100 Mbps で利用するためにはファームウェア書き換えが必要になり、そのような要求はほぼないだろうと思っているので 100 Mbps のバージョンは作成していません。よって、**100 Mbps でしか通信できないネットワークスイッチなどにつなぐと SiTCP が動かなくなります。**

3.1 HUL RM

HUL RM は Mezzanine HRM をマウントすることにより、J0 バスマスタとなることができ、なおかつ HRM が受信したデータを読み出す DAQ モジュールとして動作させることができます。このプロジェクトはソースコードを公開しているので DAQ モジュールを作る際に参考にしてください。その際に、HRM の機能を残すかどうかは各自判断してください。1 つのメザニンポートに 2 つの機能を割り当てることになるので、タイミング制約を追加でかけないといけない場合があります。

Firmware 固有名と現在の最新版

BCT に対して Version 読み出しを行うと 32bit のレジスタが返ってきます。このうち上位 16bit が Firmware 固有名で、下位 16bit がバージョン（メジャーバージョン 8bit + マイナーバージョン 8bit）となります。

現在の HUL RM の固有名とバージョンは

固有名　　0x0415

メジャーバージョン　　0x01

マイナーバージョン　　0x06

以下バージョンは v メジャーバージョン．マイナーバージョンの形式で表記します。

更新歴

バージョン	リリース日	変更点
v1.0	2016.12.23	初期版
v1.1	2017.01.15	RVM のデータヘッダを 0x9C から 0xF9 へ変更。
v1.2	2017.01.23	Vivado 更新 2016.2 => 2016.4 TRM の中間バッファを分散 RAM から BRAM へ変更。深さを 128 から 1024 に変更。Prog Full threshold 導入。 それにあわせて、RVM の中間バッファ深さを 1024 へ変更。 見かけの機能に変更は無し。
v1.3	2017.03.22	IOM の初期レジスタが正しく設定できない問題を解決。機能的には v1.2 と同等。トリガー管理モジュール名を Master Trigger Manager (MTM)から Trigger Manager (TRM)へ変更。
v1.4	2017.05.09	Clear に応答しない（BUSY が立ちっぱなしになる）問題を解決。
v1.5	未公開	未公開
v1.6	2017.12.19	3.0 章にあるようにリセットシーケンスを統一。Header3 の 24 ビット目に HRM が刺さっているかどうか（正確には DIP2 が ON かどうか）を示すビットを挿入。
v1.7	2018.2.2	J0 バスからやってくるイベントタグをラッチするタイミングが早すぎて、HRM 側のイベント番号と 1 ずれるバグを解決。

モジュール動作概要

HUL RM の信号フロー概念図を図 13 に示します。この図では一部モジュールは省略しているので気をつけてください。HUL RM の機能は Mezzanine HRM が受信、およびデコードした情報の処理です。そのため、Mezzanine Base U にのみ機能が実装されていて、ほかのポートには何をつないでも無意味です。Mezzanine HRM が受信した情報は 3 つの経路に分配されます。1 つは Receiver Module (RVM) で、lock bit、spill number increment、spill number (full 8bit)、event number (full 12bit) の情報を記憶し、Event Builder に渡します。この場合、HRM はトリガー受信機ではなく、トリガー情報を測定して

いる読み出し回路という位置づけになります。そのため、RVM の情報はヘッダではなくデータボディ扱いになります。2 つ目の経路は J0 Bus へのトリガー情報の配布です。この段階で event number は 3bit へ spill number は 1bit へ減らされます。どの情報が J0 Bus のどの線に現れるかは後述します。3 つ目は TRM への入力になります。

FPGA 内部にはトリガーを管理する Trigger Manager (TRM) というモジュールが存在します。TRM は 3 つのポート、Trig Ext (NIM IN)、J0 Bus (スレーブの場合)、および HRM (存在する場合) からトリガー信号を受け取ることができ、どのポートから受け取るかはレジスタで設定します。どの経路がどのレジスタによって選択されるのかは後述します。受信したトリガー情報は FPGA 内部で利用するために TrigDownType という型で配布されます。この中には非同期のままの L1 やクロック同期されたトリガー、更に one pulse 信号に整形されたトリガーなどが含まれます。また、TRM は level2 と clear の情報を覚えていて、N 番目のイベントはビルドして転送するべきなのか、それともクリアして破棄するべきなのかを Event Builder に命令します。更に TRM はイベント Tag 情報を FPGA 内部のモジュールに再配布します。HRM ポートと J0Bus ポートの情報を同等に取り扱わないといけないので、TRM が配布する Tag は 3bit の event number と 1bit の spill number に減らされます。当然ですが、Tag を受け取っていなければ何も配布されません。

Event Builder は RVM から受け取った情報を Build しヘッダ情報を付与して SiTCP へ渡します。

I/O Manager は NIM 入力ポートをどの FPGA 内部信号に割り当てるか、また FPGA 内部の信号を NIM 出力ポートに割り当てるかを制御するモジュールです。

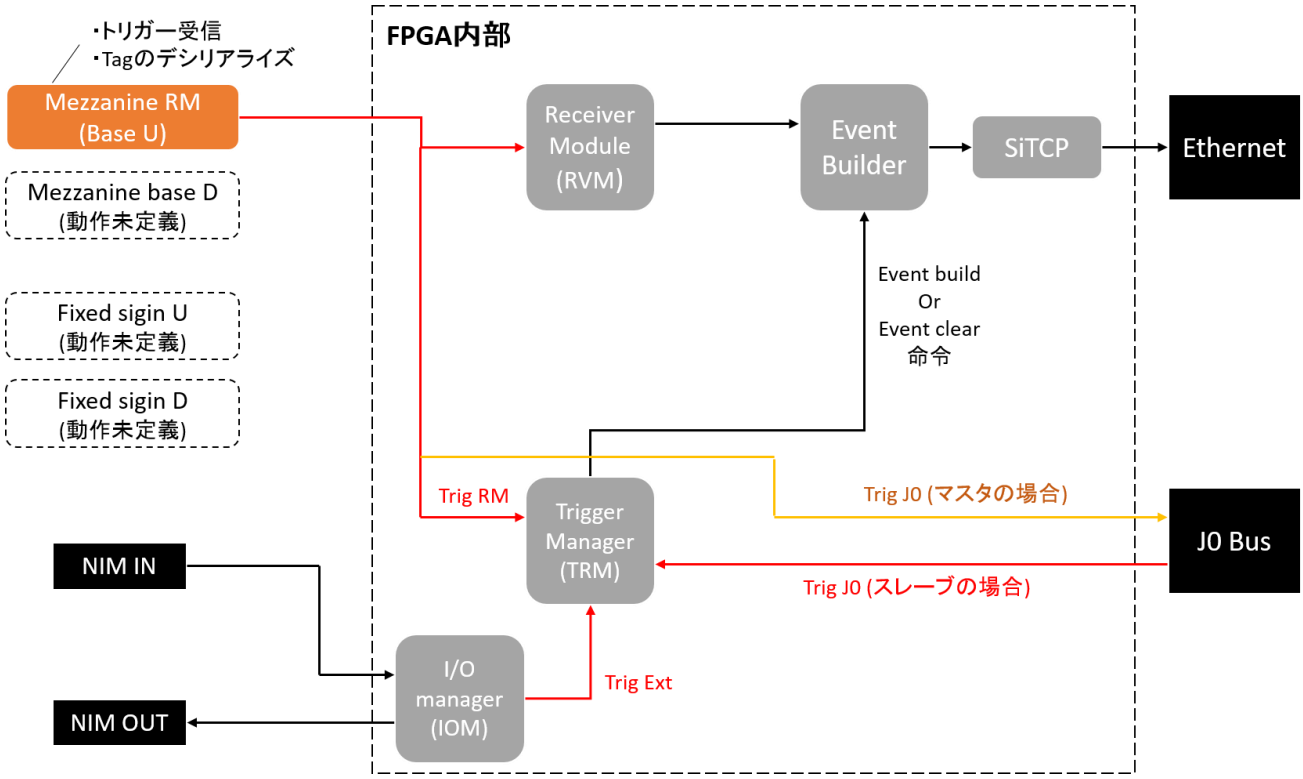


図 13：HUL RM のデータフロー外略図

レジスタマップ

FPGA 内部のモジュール ID、アドレス、レジスタについてまとめます。**これは HUL RM 専用のマップです。同名のモジュールやレジスタが他の firmware に存在しても同一の ID やアドレスであるとは限りません。必ずこのマップ（もしくは配布したソフトウェアの RegisterMap.hh 及び namespace）にしたがって設定してください。**

Trigger Manager : TRM (module ID = 0x0)				
レジスタ名	アドレス	Read/Write	ビット幅	備考
sel_trig	0x000	R/W	12	TRM 内部のトリガーポートの選択を行うレジスタ。詳細は後述。
DAQ controller : DCT (module ID = 0x1)				
gate	0x000	R/W	1	DAQ gate の 0/1 設定。DAQ gate が 0 だと TRM は trigger を出力できない。簡易なシステムなどでイベント同期を取る際などに利用する。
evb_reset	0x010	W	-	このアドレスへ書き込み要求することで Event Builder のソフトリセットがアサートされ、Event Builder 内部のセルフイベントカウンタが 0 になる。
I/O Manager : IOM (module ID = 0x2)				
nimout1	0x000	R/W	4	NIMOUT1 へ何を出力するかを設定する。詳細は後述。
nimout2	0x010	R/W	4	NIMOUT2 へ何を出力するかを設定する。
nimout3	0x020	R/W	4	NIMOUT3 へ何を出力するかを設定する。
nimout4	0x030	R/W	4	NIMOUT4 へ何を出力するかを設定する。
extL1	0x040	R/W	3	extL1 にどの NIMIN を接続するか設定。
extL2	0x050	R/W	3	extL2 にどの NIMIN を接続するか設定。
extClr	0x060	R/W	3	extClr にどの NIMIN を接続するか設定。
extBusy	0x070	R/W	3	extBusy にどの NIMIN を接続するか設定。
extRsv2	0x080	R/W	3	extRsv2 にどの NIMIN を接続するか設定。
Bus Controller : BCT (module ID = 0xE)				
Reset	0x000	W	-	Bus Controller からモジュールリセット信号をアサートし、SiTCP を除く全モジュールを初期化。
Version	0x010	R	32	Firmware の固有名とバージョンを読み出す。多バイト読み出しが必要。
ReConfig	0x020	W	-	PROG_B_ON を Low にして FPGA の再コ

				ンフィギュレーションを行う。一度通信が切れるので暫くしてから再接続。

Trigger Manager (TRM)

TRM はどの入力ポートからの信号をトリガーとして使うのかを決め、FPGA 内部用の L1、L2、Clear 信号を出力します。また、Tag を受けとっている場合 FPGA 内部で使うために再配布します。どのポートの信号を選択するかは 12bit のレジスタ、"sel_trig"で設定します。トリガー信号の経路と sel_trig の関係を図 14 にまとめます。L1 trigger をどのポートから受けるかは 3 つの bit で決まります。2 つ以上の bit をセットするとトリガーが OR で入ってきてしまうので注意してください。L1 の経路が決まると次に DAQ gate (DAQ controller:DCT が管理) との AND をとり、L1 trigger として配布されます。L2 trigger と Clear の選択も同様に 3 つの bit で行いますが、これらは経路決定後に EnL2 bit の影響を受けます。EnL2 が 0 の場合 L2 は L1 のコピーが入り、Clear は常に 0 です。TRM-RM システムを使わない簡易は DAQ を行う場合この bit を 0 にします。その後、L2 も DAQ gate との AND をとり配布されます。Tag 情報は J0 から受け取るのか HRM から受け取るのかを EnJ0 と EnRM bits で選択します。両方 1 にすると Tag が出力されないので注意してください。

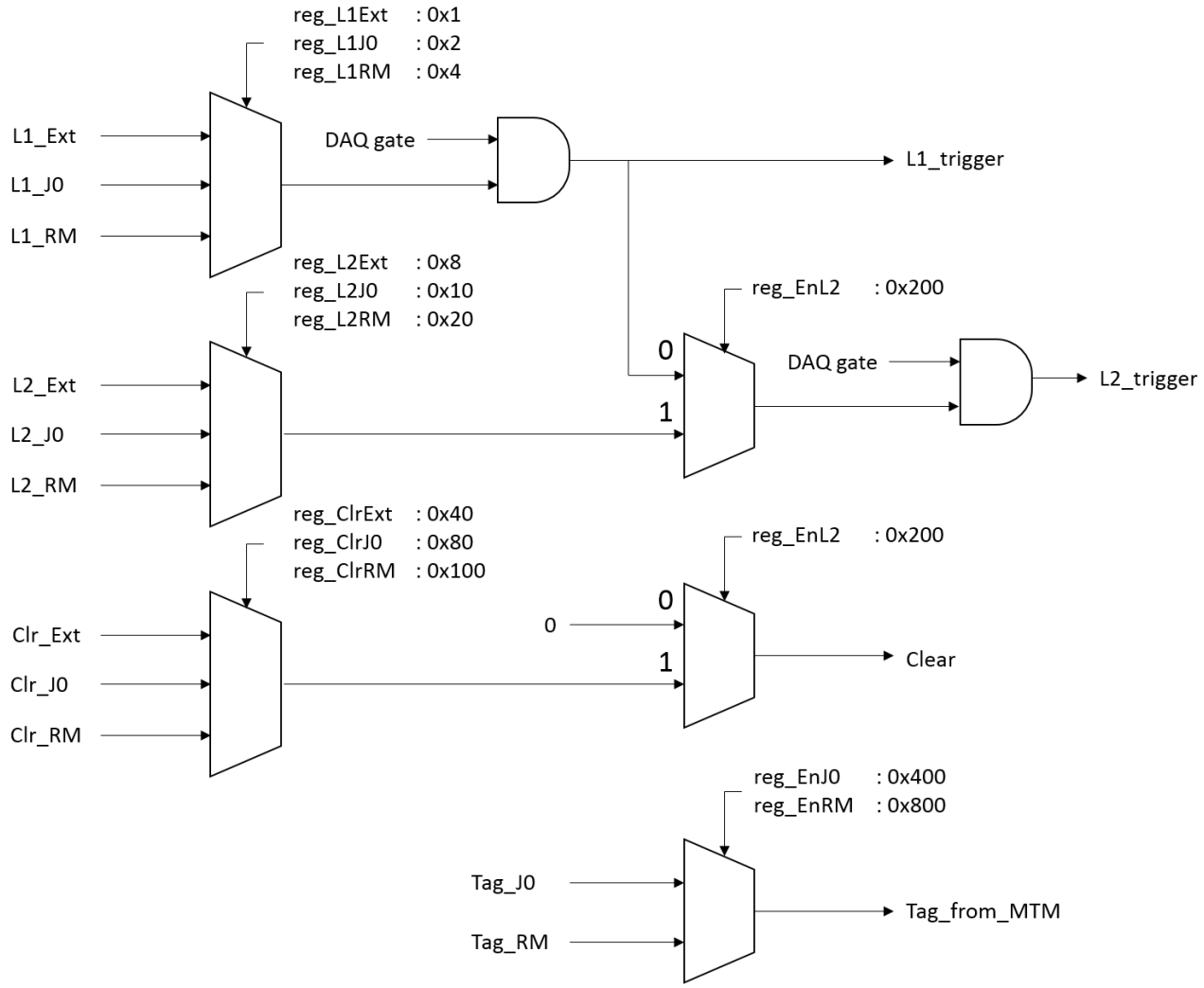


図 14：TRM 内部におけるレジスタとトリガー選択の関係

TRM::sel_trig アドレスに対するレジスタのリストです。それぞれのビットが該当するセレクタのスイッチになっているので、このレジスタは整数値ではなく 12bit のビット列になります。例外的に reg_EnJ0 のみ、TRM の外でも使われます。reg_EnJ0 が High であり、かつ DIP SW2 2 番（Mezzanine HRM）が Low の場合に限り、J0 bus へ module busy を流します。クレートにはモジュールさすが J0 bus へは永長を与えたくない場合はこのレジスタを Low にします。

レジスタラベル	レジスタ値	備考
reg_L1Ext	0x1	NIMIN から L1 trigger を選択。
reg_L1J0	0x2	J0 bus からの L1 trigger を選択。
reg_L1RM	0x4	Mezzanine HRM からの L1 trigger を選択。
reg_L2Ext	0x8	NIMIN からの L2 trigger を選択。
reg_L2J0	0x10	J0 bus からの L2 trigger を選択。
reg_L2RM	0x20	Mezzanine HRM からの L2 trigger を選択。
reg_ClrExt	0x40	NIMIN からの Clear を選択。
reg_ClrJ0	0x80	J0 bus からの Clear を選択。
reg_ClrRM	0x100	Mezzanine HRM からの Clear を選択。
reg_EnL2	0x200	0: L2=L1 trigger、1: L2=L2 入力
reg_EnJ0	0x400	Tag 情報に J0 bus の物を採用する。また、この bit が 1 だと J0 bus へ自身の module busy を流す。
reg_EnRM	0x800	Tag 情報に J0 bus の物を採用する。

I/O Manager (IOM)

IOM は FPGA 内部の信号を NIMOUT へ割り当てたり、NIMIN の信号を FPGA 内部の信号線へ接続したりする機能を持ちます。たとえば、以下で述べる nimout1 へ reg_o_ModuleBusy を設定した場合、フロントパネルの NIM 出力 1 番から BUSY 信号が出力されます。一方で、extL1 へ reg_i_nimin1 を設定した場合、TRM の L1Ext 線に NIM 入力 1 番が割り当てられます。割り当てることの出来るレジスタを以下にまとめます。NIM 出力は、出力する NIM ポートのアドレスに信号線のレジスタを設定する、NIM 入力は内部の信号線のアドレスへ NIM 入力ポートのレジスタを設定すると、逆になっているので注意してください。これらのレジスタ値は整数値として解釈します。

NIMOUT へ出力可能な信号線		
レジスタラベル	レジスタ値	備考
reg_o_ModuleBusy	0x0	Module busy です。Module busy は自身の内部 busy のみを指します。J0 bus の busy や ExtBusy は含まれません。
reg_o_CrateBusy	0x1	CrateBusy です。CrateBusy は module busy に加えて J0 bus の busy や ExtBusy を含みます。J0 bus マスタの場合に利用する信号になり、また HRM が Trigger Module へ返す busy と同等です。

reg_o_RML1	0x2	HRM が受信した L1 trigger を出力します。
reg_o_RML2	0x3	HRM が受信した L2 trigger を出力します。
reg_o_RMClr	0x4	HRM が受信した Clear を出力します。
reg_o_RMRsv1	0x5	HRM が受信した Reserve 1 を出力します。
reg_o_RMSnInc	0x6	HRM が Spill Number Increment を出力します。
reg_o_DaqGate	0x7	DCT の DAQ gate を出力します。
reg_o_DIP8	0x8	DIP SW2 8 番のレベルを出力します。
reg_o_clk1MHz	0x9	1 MHz のクロックを出力します。
reg_o_clk100kHz	0xA	100 kHz のクロックを出力します。
reg_o_clk10kHz	0xB	10 kHz のクロックを出力します。
reg_o_clk1kHz	0xC	1 kHz のクロックを出力します。
内部信号線へ割り当て可能な NIMIN ポート		
reg_i_nimin1	0x0	NIMIN1 番を信号線へアサインします。
reg_i_nimin2	0x1	NIMIN2 番を信号線へアサインします。
reg_i_nimin3	0x2	NIMIN3 番を信号線へアサインします。
reg_i_nimin4	0x3	NIMIN4 番を信号線へアサインします。
reg_i_default	0x7	このレジスタが設定された場合、指定のデフォルト値がそれぞれの内部信号線へ代入されます。

IOM には初期割り当てが存在します。その初期値を以下に列挙します。

NIM 出力ポート	**初期レジスタ**	
NIMOUT1	reg_o_ModuleBusy	
NIMOUT2	reg_o_DaqGate	
NIMOUT3	reg_o_clk1kHz	
NIMOUT4	reg_o_DIP8	
内部信号線	**初期レジスタ**	**デフォルト値**
ExtL1	reg_i_nimin1	NIMIN1
ExtL2	reg_i_default	0
ExtClear	reg_i_default	0
ExtBusy	reg_i_nimin3	NIMIN3
ExtRsv2	reg_i_nimin4	NIMIN4

DIP SW2 の機能

DIP SW2 に割り当てられている機能を列挙します。

スイッチ番号	機能	詳細

1	SiTCP force default	ON で SiTCP のデフォルトモードで起動します。電源投入前に設定している必要があります。
2	Mezzanine HRM	ON で Mezzanine HRM をマウントするためのモードになります。このビットにより切り替わる機能については後述します。 このビットがデータの Header3 に現れます。
3	Force BUSY	Crate Busy と Module Busy を強制的に High にします。接続チェックなどに使ってください。
4	Bus BUSY	ON で Crate Busy に J0 bus busy を含め、OFF で含めません。
5	LED	ON で LED4 番を光らせます。
6	NC	
7	NC	
8	Level	IOM の DIP8 から出力されるレベルです。

Mezzanine HRM (DIP SW2) の機能

このビットが ON になると Mezzanine HRM にかかわる複数の機能が切り替わります。

Mezzanine base U の入出力方向の変更

ON で Mezzanine HRM にあわせていくつかの信号線が LVDS 出力に切り替わります。OFF 状態では全ての線は LVDS 入力で接続されます。

J0 bus スレーブモードを OFF にする

このビットが ON だと、J0 bus から L1, L2, Clear, Tag を受け取らなくなります。また、BUSY 信号を J0 bus へ流しません。

J0 bus マスタモードを On にする

このビットが ON だと、J0 bus へ L1, L2, Clear, Tag 情報を流すようになり、なおかつ J0 bus に流れる BUSY 信号を受け取ることが出来るようになります。ただし、同時に DIP SW1 を全て ON にしている必要があります。

以下が J0 bus 信号線とトリガー信号の関係です。

J0 bus 信号線	**トリガー信号線**
S1	RM_Clear
S2	RM_Level2
S3	RM_SpillNumber(0)
S4	RM_Level1
S5	RM_EventNumber(0)

S6	RM_EventNumber(1)
S7	RM_EventNumber(2)

Event Builder が RVM をイベントパケットに含める

RVM の情報を Event Builder が読み出してイベントパケットに含めるようになります。

LED 点灯の機能

LED が点灯した際の意味です。

LED1	点灯中は TCP 接続が張られています。
LED2	点灯中は module busy が high です。
LED3	点灯中は DIP SW2 の Mezzanine HRM が ON であることを意味します。
LED4	点灯中は DIP SW2 の LED が ON です。

DAQ の動作

データフローと DAQ の動作について述べます。DAQ 機能は各計測用モジュール（以下計測ブロック）と Event Builder モジュールによって構成されます。データフローを図 15 に示します。トリガーを受信すると各計測ブロックは決められた動作に従いデータを処理してブロックバッファに保存します。計測ブロックは内部にブロックバッファを持ち、マルチイベントを一時的に保存できるようになっています。Event Builder は各計測ブロックからデータを読み、イベントバッファを Full でない限りイベントビルドを続けます。そのため、DAQ 機能がトリガーに対して同期して動くのはブロックバッファにデータを書き込むまでであり、それから後段の処理は外界の信号や状態に依存せず、帯域が許す限りデータを転送し続けるのみになります。

HUL RM の場合計測ブロックは RVM と TRM のみなります。ですが、TRM の情報はデータボディには含まれなず、ヘッダに含まれるため厳密には TRM は計測ブロックではありません。Event Build される情報は RVM データのみで、TRM の情報はデータの転送制御に使われます。TRM は N 番目のイベントに L2 trigger と Clear のどちらが着ているかを保存しており、この情報を使って Event Builder は N 番目のイベントパケットを SiTCP に送るか、それともそのまま破棄するかを決定します。そのため、**HUL の DAQ 機能には実質的には Fast Clear という動作がなく、Cear BUSY という物も存在しません。**L1 rigger によって発生した全イベントは一度デジタイズされ、ビルドされます。ただし、Event Builder が付与するセルフイベント番号は転送が行われない限りインクリメントされません。

RVM は L2 trigger のタイミングで図 15 に示した情報をラッチしてブロックバッファに保存します。

Module Busy となるタイミング

HUL RM の Module BUSY の定義は以下に列挙する BUSY 信号の OR になります。Block full と TCP full はネットワーク転送が負けない限り発生しないため、通常は 160 ns の固定長 BUSY のみとなります。現在 Self-busy に 160 ns が設定されていますが、若干長めのため将来的に短くする可能性があります。

BUSY	BUSY 長	備考
Self-busy	160 ns	L1 trigger を検出した瞬間から固定長でアサート。
Block full	—	ブロックバッファが Full になった段階で BUSY が出力されます。L1 trigger レートが後段の回路のデータ処理速度を上回るとアサートされます。つまり TCP 転送が負けていることを意味するので、実質的に TCP full と同等です。
TCP full	—	SiTCP の TCP バッファが Full になると出力されます。ネットワーク帯域に対して Event Builder が送信しようとするデータ量が多いとアサートされます。

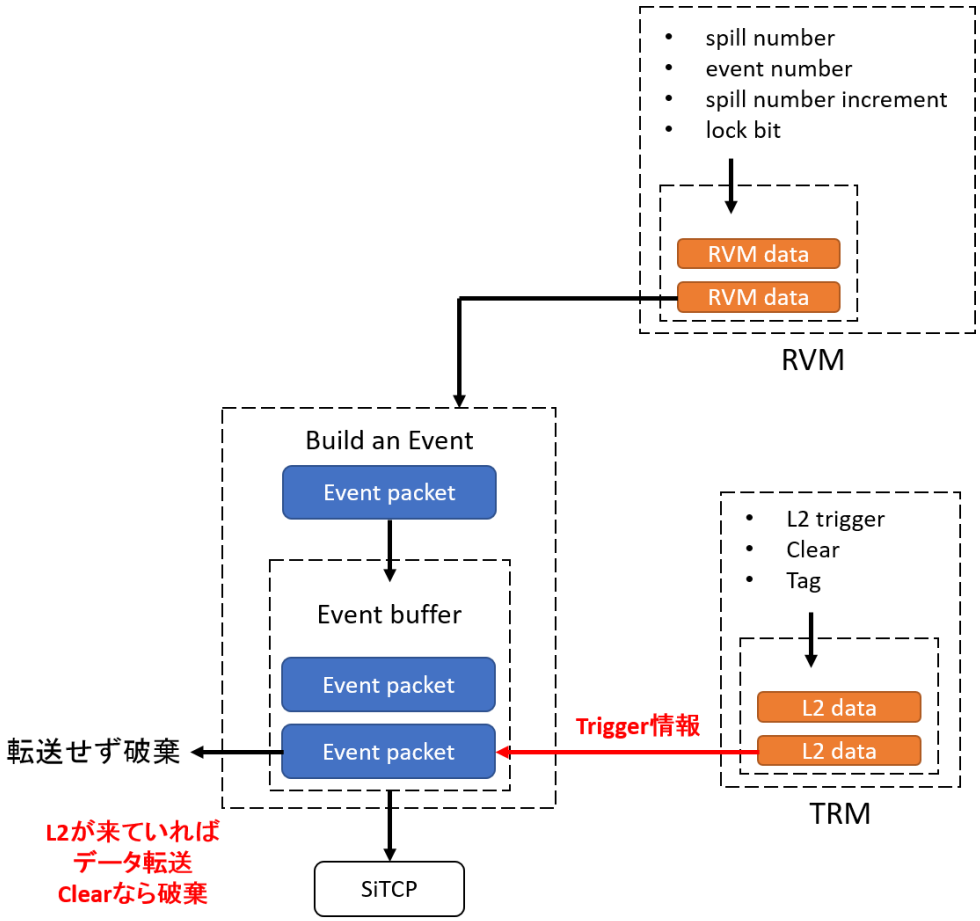


図 15：HUL RM のデータフロー

データ構造

HUL では 32bit を 1 ワードとして、3 ワードのヘッダと可変長のデータボディを 1 イベントのブロックとしています。

ヘッダワード

ヘッダ 1 (マジックワード)

上位ビット　　下位ビット

0xFFFF0415

ヘッダ 2 (イベントサイズ)

0xFF	0x00	“00000”	Number of Word (11bit)

Number Of Word はデータボディに含まれるワード数を示します。ヘッダ 3 ワードは含まないので注意してください。

ヘッダ 3 (イベント番号)

0xFF	HRM exist	“000”	Tag (4bit)	Self-counter (16bit)

HRM exist が 1 であれば、DIP の 2 ビット目が ON であり、デコード時に HRM がマウントされている事が分かります。つまり、data body に必ず RVM ワードが存在します。Tag は TRM から出力される 4bit の Tag 情報です。下位 3bit が RM Event Number の下位 3 ビット、4 ビット目が RM Spill Number の最下位ビットとなります。Self-counter はイベント転送を行うたびにインクリメントされる Local Event Number で、0 オリジンです。

RVM ワード

0xF9	“00”	Lock	SNI	Spill Num (8bit)	Event Num (12bit)

Lock は RM lock bit で 1 である必要があります。Spill Number Increment (SNI)はスピル番号の変わり目で 1 になるべき信号です。 (ただし本当にスピル変わり目がデータで見えるかは未検証) Spill Num.は HRM が受信したスピル番号です。Event Num は HRM が受信したイベント番号です。

3.2 HUL Scaler

HUL Scaler は HUL RM の機能にスケーラを追加したファームウェアです。実装したスケーラは 300 MHz でサンプリングを行う、28bit の同期カウンタです。HUL Scaler は HUL RM と多くの機能が共通のため、異なる点のみ述べます。

Firmware 固有名と現在の最新版

現在の HUL Scaler の固有名とバージョンは

固有名	**0x4ca1**
メジャーバージョン	**0x01**
マイナーバージョン	**0x07**

更新歴

バージョン	リリース日	変更点
v1.0	2016.12.23	初期版
v1.1	2017.01.15	RVM のデータヘッダを 0x9C から 0xF9 へ変更。
v1.2	2017.01.27	Vivado 更新 2016.2 => 2016.4 TRM の中間バッファを分散 RAM から BRAM へ変更。深さを 128 から 256 に変更。Prog Full threshold 導入。256 に設定した理由は SCR ブロックの深さを越えないようにするため。 それにあわせて、RVM の中間バッファ深さを 256 へ変更。 見かけの機能に変更は無し。
v1.3	2017.03.22	IOM の初期レジスタが正しく設定できていない問題を解決。電源投入後最初のイベントでヘッダ 2 に書かれているワード数が 0 になってしまう問題を解決。
v1.4	2017.05.09	Clear に応答しない（BUSY が立ちっぱなしになる）問題を解決。
v1.5		HRM を使っていて尚且つ Clear が入るとハングする問題を解決。（リリースしないまま v1.6 にとってかわった。）
v1.6	2017.08.22	トリガーが入って 2 us 程度以内にハードリセットが入ると DAQ がハングする問題を修正。ハードリセットへ応答するかどうか、ブロックごとにレジスタで設定できるように変更。新しいローカルアドレス追加。
v1.7	2017.12.19	3.0 章にあるようにリセットシーケンスを統一。Header3 の 24 ビット目に HRM が刺さっているかどうか（正確には DIP2 が ON かどうか）を示すビットを挿入。
v1.8	2018.2.2	J0 バスからやってくるイベントタグをラッチするタイミングが早すぎて、HRM 側のイベント番号と 1 ずれるバグを解決。

モジュール動作概要

HUL Scaler では HRM で動作未定義となっていた残りの Mezzanine base D と固定入力ポートに機能が割り当てられています。Mezzanine base には DCR v1/2 を実装することを想定しており、最大 128ch 分のスケーラ値を取得することができます。また、Mezzanine base U には DCR の代わりに HRM をマウントすることで HUL RM と同様 J0 bus マスタになることも可能です。この場合 base U に割り当てられている 32ch 分は強制的にデータから削除されます。

Scaler は 300 MHz、28bit 長のカウンタで構成されており、L1 trigger のタイミングでカウンタをラッチしてバッファへ書き込みます。HUL scaler では 2 つ新しい内部信号が IOM に接続されています。1 つは spill gate で、この信号が High の間のみスケーラはカウントアップします。もう 1 つは counter reset で、high になったタイミングでカウンタ値を全てリセットします。NIM 入力のカウンターリセットに応答するかどうかは、enable_hdrst で設定可能です（v1.6 以降）。

他の機能は HUL RM と同様になります。

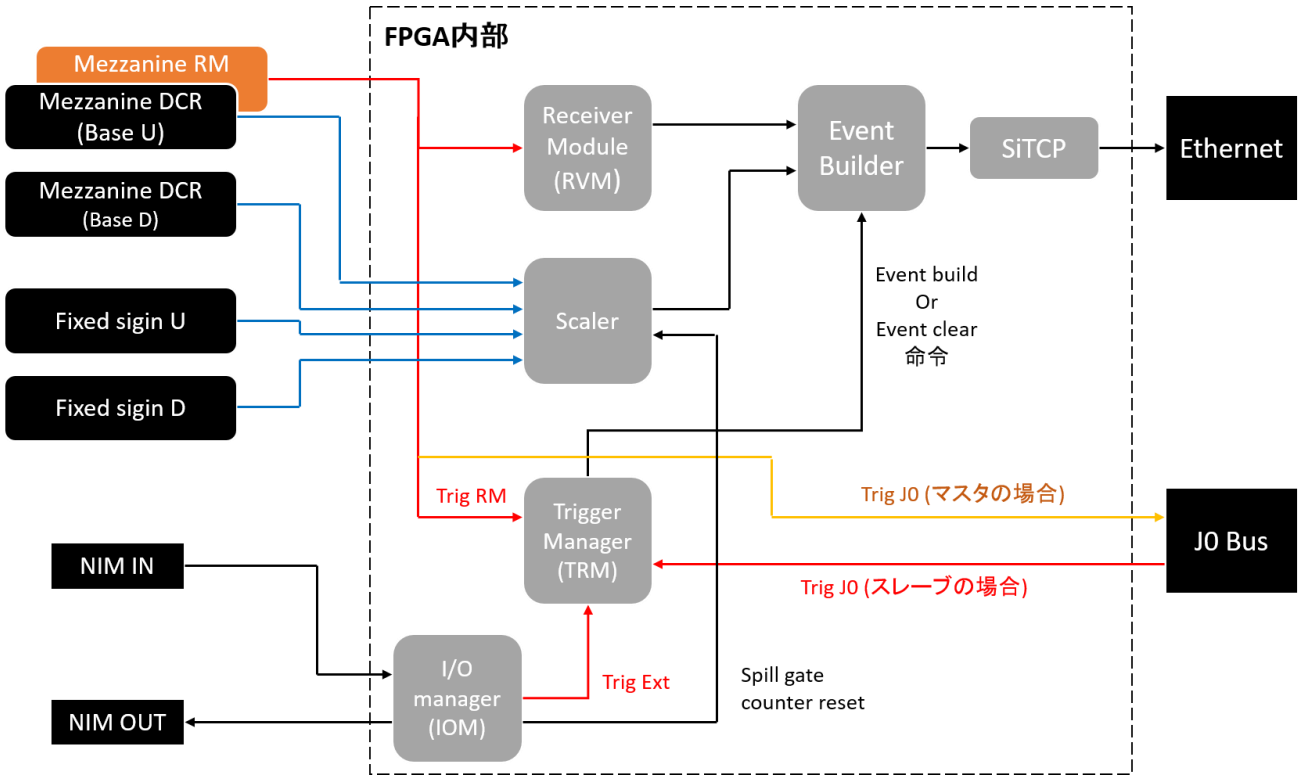


図 16：HUL Scaler データフロー概略図。

レジスタマップ

FPGA 内部のモジュール ID、アドレス、レジスタについてまとめます。これは HUL Scaler 専用のマップです。**同名のモジュールやレジスタが他の firmware に存在しても同一の ID やアドレスであるとは限りません。必ずこのマップ（もしくは配布したソフトウェアの RegisterMap.hh 及び namespace）にしたがって設定してください。**

Trigger Manager : TRM (module ID = 0x0)				
レジスタ名	アドレス	Read/Write	ビット幅	備考
sel_trig	0x000	R/W	12	TRM 内部のトリガーポートの選択を行うレジスタ。
DAQ controller : DCT (module ID = 0x1)				
gate	0x000	R/W	1	DAQ gate の 0/1 設定。DAQ gate が 0 だと TRM は trigger を出力できない。簡易なシステムなどでイベント同期を取る際などに利用する。
evb_reset	0x010	W	—	このアドレスへ書き込み要求することで Event Builder のソフトリセットがアサートされ、Event Builder 内部のセルフイベントカウンタが 0 になる。
Scaler : SCR (module ID = 0x2)				
counter_reset	0x000	W	—	Software counter reset をアサート。
enable_block	0x010	R/W	4	どのブロック（入力ポート毎）を利用するか設定。ビットを High にするとそのブロックのデータが取得できるようになります。 入力ポートと対応するビットは 1bit 目：Fixed Sigin U 2bit 目：Fixed Sigin D 3bit 目：Mezzanine U 4bit 目：Mezzanine D
enable_hdrst	0x020	R/W	4	NIM 入力の hardware counter reset に応答するかどうかをブロックごとに設定します。ビットが 1 でそのブロックは hardware counter reset のタイミングでカウンタが 0 に戻ります。 入力ポートと対応するビットは 1bit 目：Fixed Sigin U 2bit 目：Fixed Sigin D 3bit 目：Mezzanine U

				4bit 目：Mezzanine D
I/O Manager：IOM (module ID = 0x3)				
nimout1	0x000	R/W	4	NIMOUT1 へ何を出力するかを設定する。
nimout2	0x010	R/W	4	NIMOUT2 へ何を出力するかを設定する。
nimout3	0x020	R/W	4	NIMOUT3 へ何を出力するかを設定する。
nimout4	0x030	R/W	4	NIMOUT4 へ何を出力するかを設定する。
extL1	0x040	R/W	3	extL1 にどの NIMIN を接続するか設定。
extL2	0x050	R/W	3	extL2 にどの NIMIN を接続するか設定。
extClr	0x060	R/W	3	extClr にどの NIMIN を接続するか設定。
extSpillGate	0x070	R/W	3	ext spill gate にどの NIMIN を接続するか設定。
extCCRst	0x080	R/W	3	ext counter reset にどの NIMIN を接続するのか設定。
extBusy	0x090	R/W	3	extBusy にどの NIMIN を接続するか設定。
extRsv2	0x0A0	R/W	3	extRsv2 にどの NIMIN を接続するか設定。
Bus Controller：BCT (module ID = 0xE)				
Reset	0x000	W	—	Bus Controller からモジュールリセット信号をアサートし、SiTCP を除く全モジュールを初期化。
Version	0x010	R	32	Firmware の固有名とバージョンを読み出す。多バイト読み出しが必要。
ReConfig	0x020	W	—	PROG_B_ON を Low にして FPGA の再コンフィギュレーションを行う。一度通信が切れるので暫くしてから再接続。

Trigger Manager (TRM)

機能およびレジスタは HUL RM と同様です。

Scaler (SCR)

スケーラは HUL Scaler の主機能になります。各スケーラユニットは 300 MHz で入力信号を同期し、その後エッジ検出を行い、そのエッジのタイミングでカウンタを 1 つインクリメントします。そのため、検出可能なパルスの最小幅は 3.5～4.0 ns 程度で、なおかつ 2 つのパルスを分離するためにも同程度パルスが離れている必要があります。カウンタは 28bit 長で一周すると 0 に戻ります。スケーラは 32ch（入力ポート毎）に 4 つのブロックに分けられており、enable block レジスタで利用するブロックを設定できます。対応するビットが High であればそのブロック全体（32ch 分)のデータが返ってきて、Low であれば逆に 32ch 全て返ってきません。スケーラのリセットはハードリセットの ext counter reset (NIMIN、IOM で制御)か、ソフトリセットの counter reset アサートによって行うことが出来ます。ハードリセットに応答するかどうかは enable_hdrst でブロックごとに設定できます。このビットが 1 のブロックはハードリセットのタイミングでカウンタが 0 になります。**また、トリガーから前後 100 ns にリセットが入った場合の動作は不定とします。データは返ってきますが、意図しない値が入っているかもしれません。**

スケーラのインクリメントは ext spill gate (NIMIN、IOM で制御）によって制御できます。スケーラは spill gate が High の時だけインクリメントされます。Spill gate は NIM 入力ポートを割り当てると利用することが出来、もし未割り当ての場合デフォルトで常に High になるように設定されています。

I/O Manager (IOM)

IOM の機能は HUL RM と基本的に同様ですが、extSpillGate と extCCRst への NIMIN ポートの割り当てが追加されています。NIMOUT 側は変更ありません。

NIMOUT へ出力可能な信号線		
レジスタラベル	レジスタ値	備考
reg_o_ModuleBusy	0x0	Module busy です。Module busy は自身の内部 busy のみを指します。J0 bus の busy や ExtBusy は含まれません。
reg_o_CrateBusy	0x1	CrateBusy です。CrateBusy は module busy に加えて J0 bus の busy や ExtBusy を含みます。J0 bus マスタの場合に利用する信号になり、また HRM が Trigger Module へ返す busy と同等です。
reg_o_RML1	0x2	HRM が受信した L1 trigger を出力します。
reg_o_RML2	0x3	HRM が受信した L2 trigger を出力します。
reg_o_RMClr	0x4	HRM が受信した Clear を出力します。
reg_o_RMRsv1	0x5	HRM が受信した Reserve 1 を出力します。
reg_o_RMSnInc	0x6	HRM が Spill Number Increment を出力します。
reg_o_DaqGate	0x7	DCT の DAQ gate を出力します。
reg_o_DIP8	0x8	DIP SW2 8 番のレベルを出力します。
reg_o_clk1MHz	0x9	1 MHz のクロックを出力します。

reg_o_clk100kHz	0xA	100 kHz のクロックを出力します。
reg_o_clk10kHz	0xB	10 kHz のクロックを出力します。
reg_o_clk1kHz	0xC	1 kHz のクロックを出力します。
内部信号線へ割り当て可能な NIMIN ポート		
reg_i_nimin1	0x0	NIMIN1 番を信号線へアサインします。
reg_i_nimin2	0x1	NIMIN2 番を信号線へアサインします。
reg_i_nimin3	0x2	NIMIN3 番を信号線へアサインします。
reg_i_nimin4	0x3	NIMIN4 番を信号線へアサインします。
reg_i_default	0x7	このレジスタが設定された場合、指定のデフォルト値がそれぞれの内部信号線へ代入されます。

IOM には初期割り当てが存在します。その初期値を以下に列挙します。

NIM 出力ポート	**初期レジスタ**	
NIMOUT1	reg_o_ModuleBusy	
NIMOUT2	reg_o_DaqGate	
NIMOUT3	reg_o_clk1kHz	
NIMOUT4	reg_o_DIP8	
内部信号線	**初期レジスタ**	
ExtL1	reg_i_nimin1	NIMIN1
ExtL2	reg_i_default	0
ExtClear	reg_i_default	0
ExtSpillGate	reg_i_default	1
ExtCounterReset	reg_i_nimin2	NIMIN2
ExtBusy	reg_i_nimin3	NIMIN3
ExtRsv2	reg_i_nimin4	NIMIN4

DIP SW2 の機能

HUL RM と同様です。

LED 点灯の機能

HUL RM と同様です。

DAQ の動作

データフローを図 17 に示します。HUL RM に SCR が追加され、DIP SW2 の Mezzanine HRM が OFF であれば SCR のみからデータを集め、Mezzanine HRM が ON であれば SCR と RVM 両方からデータを集めて Event Build します。ヘッダ 2 の Number Of Word には SCR と RVM の合計ワード数が入ります。

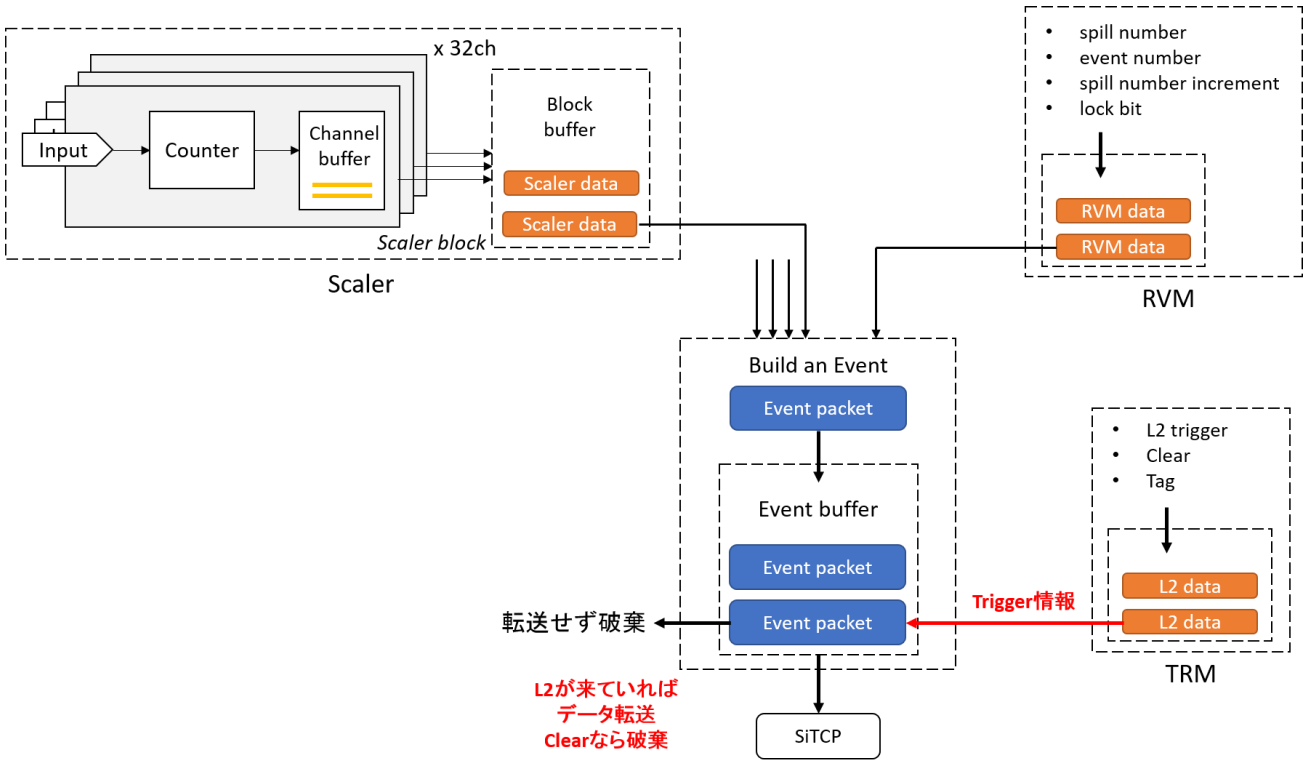


図 17：HUL Scaler のデータフロー

Module Busy となるタイミング

HUL Scaler の Module BUSY の定義は以下に列挙する BUSY 信号の OR になります。Block full と TCP full はネットワーク転送が負けない限り発生しないため、通常は 210 ns の固定長 BUSY のみとなります。現在 Self-busy に 210 ns が設定されていますが、若干長めのため将来的に短くする可能性があります。HUL RM と Self-busy の長さが異なるのは使っているシステムクロックの周波数が異なるためです。

BUSY	BUSY 長	備考
Self-busy	210 ns	L1 trigger を検出した瞬間から固定長でアサート。
Block full	—	SCR か RVM のブロックバッファが Full になった段階で BUSY が出力されます。L1 trigger レートが後段の回路のデータ処理速度を上回るとアサートされます。つまり TCP 転送が負けていることを意味するので、実質的に TCP full と同等です。
TCP full	—	SiTCP の TCP バッファが Full になると出力されます。ネットワーク帯域に対して Event Builder が送信しようとするデータ量が多いとアサートされます。

データ構造

ヘッダワード

ヘッダ 1 (マジックワード)

上位ビット　　下位ビット

0xFFFF4ca1

ヘッダ 2 (イベントサイズ)

0xFF	0x00	“00000”	Number of Word (11bit)

Number Of Word はデータボディに含まれるワード数を示します。ヘッダ 3 ワードは含まないので注意してください。

ヘッダ 3 (イベント番号)

0xFF	HRM exist	“000”	Tag (4bit)	Self-counter (16bit)

HRM exist が 1 であれば、DIP の 2 ビット目が ON であり、デコード時に HRM がマウントされている事が分かります。つまり、data body に必ず RVM ワードが存在します。Tag は TRM から出力される 4bit の Tag 情報です。下位 3bit が RM Event Number の下位 3 ビット、4 ビット目が RM Spill Number の最下位ビットとなります。Self-counter はイベント転送を行うたびにインクリメントされる Local Event Number で、0 オリジンです。

RVM ワード

0xF9	“00”	Lock	SNI	Spill Num (8bit)	Event Num (12bit)

Lock は RM lock bit で 1 である必要があります。Spill Number Increment (SNI)はスピル番号の変わり目で 1 になるべき信号です。 (ただし本当にスピル変わり目がデータで見えるかは未検証) Spill Num.は HRM が受信したスピル番号です。Event Num は HRM が受信したイベント番号です。このワードは必ず data body の先頭で返ってきます。

SCR ワード

SCR Block (4bit)	Counter (28bit)

スケーラはデータ領域に 28bit もの幅を使っているために制御ビットが少ないです。SCR Block はそのワードがどの入力ブロックに属するかを示します。また、SCR ワードにはチャンネルを表すビットが存在しません。若いワードから順番に ch0, ch1, ch2 の順番で並んでいるので、そのようにデコードしてください。

SCR Block bit	入力ポート
0x8	Fixed Sigin U
0x9	Fixed Sigin D
0xA	Mezzanine U
0xB	Mezzanine D

3.3 HUL MH-TDC

HUL MH-TDC は HUL RM の機能に Multi-Hit TDC を追加したファームウェアです。HUL RM と多くの機能が共通のため、異なる点のみ述べます。

Firmware 固有名と現在の最新版

現在の HUL MH-TDC の固有名とバージョンは

固有名	**0x30CC**
メジャーバージョン	**0x01**
マイナーバージョン	**0x08**

更新歴

バージョン	リリース日	変更点
v1.0	2016.12.23	初期版
v1.1	2017.01.15	RVM のデータヘッダを 0x9C から 0xF9 へ変更。
v1.2	2017.01.27	Vivado 更新 2016.2 => 2016.4 Block buffer が BuildIn FIFO だったので BRAM にして深さを 4096 にする。EventBuffer の深さを 2048 から 4096 にして pgfull を 4058 にする。 TDC ブロックの Channel buffer を分散 RAM から BRAM へ変更。 TRM の中間バッファを分散 RAM から BRAM へ変更。Prog Full threshold 導入。それにあわせて、RVM の中間バッファ深さを 128 へ変更。 **Ring buffer の write pointer increment にバグがあり、BUSY 明け直後の 14μs 分のデータがおかしくなる問題を解決。**
v1.4	2017.05.09	Clear に応答しない（BUSY が立ちっぱなしになる）問題を解決。
v1.5		HRM を使っていて尚且つ Clear が入るとハングする問題を解決。（未リリース）
v1.6		一度でも max multihit (16 hit/ch)を使い切ると、それいこうのイベントがずれる問題に対処。 （未リリース）
v1.7	2017.08.22	FPGA 内部のイベントシーケンスのバグを修正。**致命的なので v1.7 未満のファームウェアはしようしないこと。**
v1.8	2017.12.19	3.0 章にあるようにリセットシーケンスを統一。Header3 の 24 ビット目に HRM が刺さっているかどうか（正確には DIP2 が ON かどうか）を示すビットを挿入。 **高負荷な状況で稀にデータが壊れるバグを修正。ヘッダ 2 のワー**

		ド数のビット幅を 11 から 12bit へ拡張。
v1.9	2018.2.2	J0 バスからやってくるイベントタグをラッチするタイミングが早すぎて、HRM 側のイベント番号と 1 ずれるバグを解決。 Mezzanine HR-TDC で見られた、サーチウィンドウ外からデータが返ってくる、バグの原因を MH-TDC も内包するため、該当部分を改善した。

モジュール動作概要

HUL Scaler では RM で動作未定義となっていた残りの Mezzanine base D と固定入力ポートに機能が割り当てられています。Mezzanine base には DCR v1/2 を実装することを想定しており、最大 128ch 分の TDC を取得することができます。また、Mezzanine base U には DCR の代わりに HRM をマウントすることで HUL RM と同様 J0 bus マスタになることも可能です。この場合 base U に割り当てられている 32ch 分は強制的にデータから削除されます。

MH-TDC は 300 MHz 4 相の多相クロック式の TDC を実装しており、1bit の時間精度は 0.83 ns です。Leading と Trailing の両エッジを検出することができ、RingBuffer の長さは 13.7 us、時間分解能は 300 ps (σ)です。

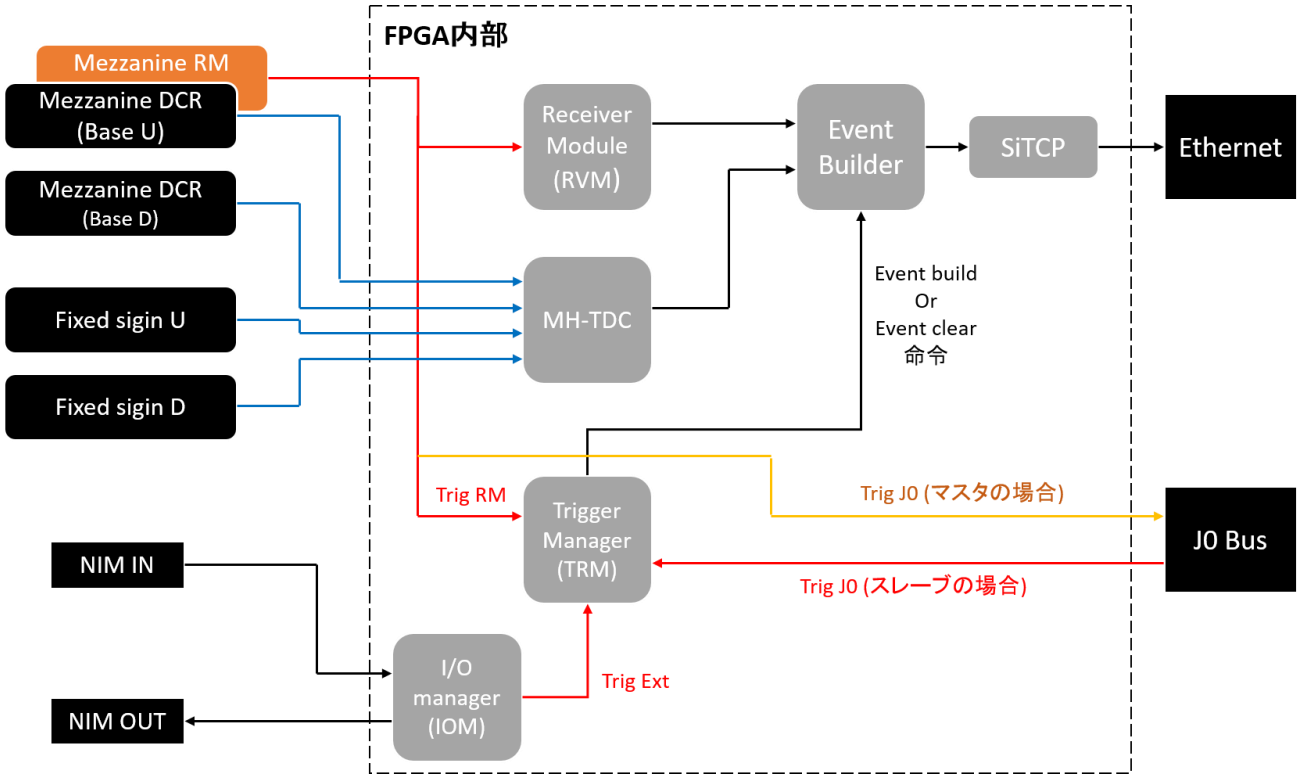


図 18：HUL MHTDC のデータフロー外略図

レジスタマップ

FPGA 内部のモジュール ID、アドレス、レジスタについてまとめます。これは HUL MH-TDC 専用のマップです。**同名のモジュールやレジスタが他の firmware に存在しても同一の ID やアドレスであるとは限りません。必ずこのマップ（もしくは配布したソフトウェアの RegisterMap.hh 及び namespace）にしたがって設定してください。**

Trigger Manager : TRM (module ID = 0x0)				
レジスタ名	アドレス	Read/Write	ビット幅	備考
sel_trig	0x000	R/W	12	TRM 内部のトリガーポートの選択を行うレジスタ。
DAQ controller : DCT (module ID = 0x1)				
gate	0x000	R/W	1	DAQ gate の 0/1 設定。DAQ gate が 0 だと TRM は trigger を出力できない。簡易なシステムなどでイベント同期を取る際などに利用する。
evb_reset	0x010	W	—	このアドレスへ書き込み要求することで Event Builder のソフトリセットがアサートされ、Event Builder 内部のセルフイベントカウンタが 0 になる。
TDC (module ID = 0x2)				
enable_block	0x000	R/W	4	どのブロック（入力ポート毎）を利用するか設定。ビットを High にするとそのブロックのデータが取得できるようになります。 入力ポートと対応するビットは 1bit 目：Fixed Sigin U 2bit 目：Fixed Sigin D 3bit 目：Mezzanine U 4bit 目：Mezzanine D
ptrofs	0x010	R/W	11	内部制御変数。ユーザーは触らない。
win_max	0x020	R/W	11	Ring buffer からヒットを探す時間窓の上限値。1bit が 6.666... ns に相当。詳細は後述。
win_min	0x030	R/W	11	Ring buffer からヒットを探す時間窓の下限値。1bit が 6.666...ns に相当。詳細は後述。
I/O Manager : IOM (module ID = 0x3)				
nimout1	0x000	R/W	4	NIMOUT1 へ何を出力するかを設定する。
nimout2	0x010	R/W	4	NIMOUT2 へ何を出力するかを設定する。
nimout3	0x020	R/W	4	NIMOUT3 へ何を出力するかを設定する。
nimout4	0x030	R/W	4	NIMOUT4 へ何を出力するかを設定する。

extL1	0x040	R/W	3	extL1 にどの NIMIN を接続するか設定。
extL2	0x050	R/W	3	extL2 にどの NIMIN を接続するか設定。
extClr	0x060	R/W	3	extClr にどの NIMIN を接続するか設定。
extBusy	0x070	R/W	3	extBusy にどの NIMIN を接続するか設定。
extRsv2	0x080	R/W	3	extRsv2 にどの NIMIN を接続するか設定。
Bus Controller : BCT (module ID = 0xE)				
Reset	0x000	W	—	Bus Controller からモジュールリセット信号をアサートし、SiTCP を除く全モジュールを初期化。
Version	0x010	R	32	Firmware の固有名とバージョンを読み出す。多バイト読み出しが必要。
ReConfig	0x020	W	—	PROG_B_ON を Low にして FPGA の再コンフィギュレーションを行う。一度通信が切れるので暫くしてから再接続。

Multi-Hit TDC (MH-TDC)

本ファームウェアの主機能です。本 MH-TDC は 4 相クロックを使うことにより、擬似的に 1.2 GHz を作り出しています。図 19 に示す Multi-Hit TDC ブロックには、TDC unit、Ring buffer、Channel buffer の 3 つのコンポーネントが存在します。TDC unit は擬似 1.2 GHz で時間を測定し、ヒット検出を行います。TDC unit の時間分解能は 300 ps (σ)、検出可能最小パルス幅はおよそ 4 ns です。検出したヒット情報は Ring buffer に保存されます。Ring buffer の長さは 13.7 us で、Ring buffer の書き込み・読み出しポインタがコースカウントに相当します。Ring buffer は 150 MHz のクロックで駆動されているため、コースカウントは 6.666...ns 精度です。

L1 trigger を検出すると Ring buffer からの読み出しを開始します。この時、ヒットを探す範囲を min_max レジスタおよび win_min レジスタによって設定可能です。これらのレジスタは 1bit がコースカウント精度の 11bit の整数値です。この範囲に入らないヒットは Channel buffer へ書き込まれません。Ring buffer からヒット情報を探している間＝サーチ窓幅分の BUSY が出力されます。

その後 channel buffer から block buffer へデータをまとめる際に、1ch/event に許される最大ヒット数が設定されます。1ch/event に許されるヒットは TDC ch の大きいほうから 16 ヒットまでで、それ以上のヒットが channel buffer に記録されていた場合、16 を超えるデータは破棄され、overflow ビットが立ちます。

Multi-Hit TDC 仕様	
TDC 精度	0.83... ns
コースカウント精度	6.66... ns
Ring buffer 長	13.8 us
時間分解能	300 ps (σ) *実測
最小パルス幅	～4 ns
ダブルヒット分解能	～7 ns
最大ヒット数/ch/event	16

I/O Manager (IOM)

IOM の機能は HUL RM と基本的に同様です。

NIMOUT へ出力可能な信号線		
レジスタラベル	レジスタ値	備考
reg_o_ModuleBusy	0x0	Module busy です。Module busy は自身の内部 busy のみを指します。J0 bus の busy や ExtBusy は含まれません。
reg_o_CrateBusy	0x1	CrateBusy です。CrateBusy は module busy に加えて J0 bus の busy や ExtBusy を含みます。J0 bus マスタの場合に利用する信号になり、また RM が Trigger Module へ返す busy と同等です。
reg_o_RML1	0x2	HRM が受信した L1 trigger を出力します。
reg_o_RML2	0x3	HRM が受信した L2 trigger を出力します。
reg_o_RMClr	0x4	HRM が受信した Clear を出力します。

reg_o_RMRsv1	0x5	HRM が受信した Reserve 1 を出力します。
reg_o_RMSnInc	0x6	HRM が Spill Number Increment を出力します。
reg_o_DaqGate	0x7	DCT の DAQ gate を出力します。
reg_o_DIP8	0x8	DIP SW2 8 番のレベルを出力します。
reg_o_clk1MHz	0x9	1 MHz のクロックを出力します。
reg_o_clk100kHz	0xA	100 kHz のクロックを出力します。
reg_o_clk10kHz	0xB	10 kHz のクロックを出力します。
reg_o_clk1kHz	0xC	1 kHz のクロックを出力します。
内部信号線へ割り当て可能な NIMIN ポート		
reg_i_nimin1	0x0	NIMIN1 番を信号線へアサインします。
reg_i_nimin2	0x1	NIMIN2 番を信号線へアサインします。
reg_i_nimin3	0x2	NIMIN3 番を信号線へアサインします。
reg_i_nimin4	0x3	NIMIN4 番を信号線へアサインします。
reg_i_default	0x7	このレジスタが設定された場合、指定のデフォルト値がそれぞれの内部信号線へ代入されます。

IOM には初期割り当てが存在します。その初期値を以下に列挙します。

NIM 出力ポート	**初期レジスタ**	
NIMOUT1	reg_o_ModuleBusy	
NIMOUT2	reg_o_DaqGate	
NIMOUT3	reg_o_clk1kHz	
NIMOUT4	reg_o_DIP8	
内部信号線	**初期レジスタ**	**デフォルト値**
ExtL1	reg_i_nimin1	NIMIN1
ExtL2	reg_i_default	0
ExtClear	reg_i_default	0
ExtBusy	reg_i_nimin3	NIMIN3
ExtRsv2	reg_i_nimin4	NIMIN4

DIP SW2 の機能

HUL RM と同様です。

LED 点灯の機能

HUL RM と同様です。

DAQ の動作

データフローを図 19 に示します。HUL RM に MH-TDC が追加され、DIP SW2 の Mezzanine HRM が OFF であれば TDC のみからデータを集め、Mezzanine HRM が ON であれば TDC と RVM 両方からデータを集めて Event Build します。ヘッダ 2 の Number Of Word には TDC と RVM の合計ワード数が入ります。

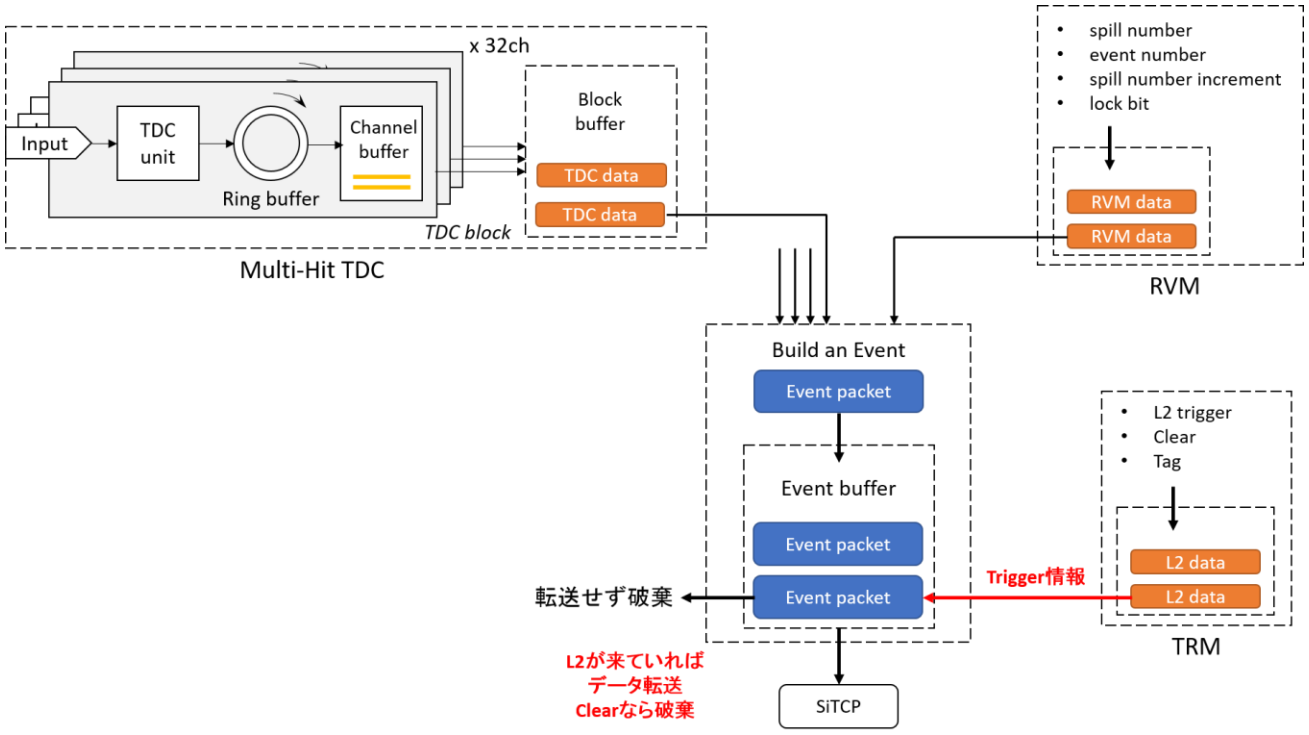


図 19：HUL MH-TDC のデータフロー

Module Busy となるタイミング

HUL MH-TDC の Module BUSY の定義は以下に列挙する BUSY 信号の OR になります。普段は Sequence busy 分の BUSY 長となるはずです。現在 Self-busy に 210 ns が設定されていますが、若干長めのため将来的に短くする可能性があります。HUL RM と Self-busy の長さが異なるのは使っているシステムクロックの周波数が異なるためです。

BUSY	BUSY 長	備考
Self-busy	210 ns	L1 trigger を検出した瞬間から固定長でアサート。
Sequence busy	サーチ幅に依存	Ring buffer からヒット情報を探すている間、つまり win_max – min_min 分の BUSY が出力されます。
Block full	—	SCR か RVM のブロックバッファが Full になった段階で BUSY が出力されます。L1 trigger レートが後段の回路のデータ処理速度を上回るとアサートされます。つまり TCP 転送が負けていることを意味するので、実質的に TCP full と同等です。

TCP full	—	SiTCP の TCP バッファが Full になると出力されます。ネットワーク帯域に対して Event Builder が送信しようとするデータ量が多いとアサートされます。

データ構造

ヘッダワード

ヘッダ 1 (マジックワード)

上位ビット 下位ビット

0xFFFF30CC

ヘッダ 2 (イベントサイズ)

0xFF	0x00	OverFlow	“000”	Number of Word (12bit)

Number Of Word はデータボディに含まれるワード数を示します。**(2017.12.19, v1.8 より 11bit から 12bit へ変更。)** ヘッダ 3 ワードは含まないので注意してください。MH-TDC では可変長になります。また、16bit 目に Over flow ビットが存在します。このビットが 1 だとそのイベントでは 128ch の中のどこかに 16hit/ch/event の上限を超えたチャンネルが存在します。(どこかはわかりません。)

ヘッダ 3 (イベント番号)

0xFF	HRM exist	“000”	Tag (4bit)	Self-counter (16bit)

HRM exist が 1 であれば、DIP の 2 ビット目が ON であり、デコード時に HRM がマウントされている事が分かります。つまり、data body に必ず RVM ワードが存在します。Tag は TRM から出力される 4bit の Tag 情報です。下位 3bit が RM Event Number の下位 3 ビット、4 ビット目が RM Spill Number の最下位ビットとなります。Self-counter はイベント転送を行うたびにインクリメントされる Local Event Number で、0 オリジンです。

RVM ワード

0xF9	“00”	Lock	SNI	Spill Num (8bit)	Event Num (12bit)

Lock は RM lock bit で 1 である必要があります。Spill Number Increment (SNI)はスピル番号の変わり目で 1 になるべき信号です。(ただし本当にスピル変わり目がデータで見えるかは未検証) Spill Num.は HRM が受信したスピル番号です。Event Num は HRM が受信したイベント番号です。

TDC ワード

Magic word (8bit)	‘0’+ Ch (7bit)	“00”	TDC (14bit)

Magic word は 0xCC で Leading、0xCD で Trailing を示します。Ch は 128ch 連番のチャンネル番号です (0 origin)。どのポートがどのチャンネルに当たるかは 1 章を参照してください。下位 14 ビットが TDC の値になります。

3.4 Mezzanine HR-TDC 及び HUL HR-TDC BASE

Mezzanine HR-TDC 内部のファームウェアとそれを制御するためのファームウェアの説明です。Mezzanine HR-TDC は HUL MH-TDC でいう Block buffer までの機能を別 FPGA へ実装したファームウェアになり、HUL HR-TDC BASE はそれ以降の Event builder や Trigger manager といった DAQ 全体を管理する機能を実装したファームウェアになります。よって、Mezzanine HR-TDC は複雑な動作はできません。Trigger（Common stop）に応答して測定データを HUL へ転送するだけがその機能になります。一方で、その制御系は FPGA が 2 つあるせいで若干複雑です。また、Mezzanine HR-TDC は Tapped delay line の Calibration LUT を持っていたり、データ転送のために DDR 通信を行っていたりと他のファームウェアにない特徴を持っています。ここではこれらの機能と制御方法について説明します。

Mezzanine HR-TDC 固有名と現在の最新版

Mezzanine HR-TDC は 2 つの異なった系統が存在します。Version 2 は Leading edge のみを測定し、Version 3 は Leading/Trailing の両方を検出します。Version 3 では必ず Trailing edge のデータも返ってきます。レジスタでオフにすることはできませんので、目的に合わせて利用してください。

固有名	**0x80CC**
メジャーバージョン	**0x02, 0x03**

更新歴

バージョン	リリース日	変更点
v2.5	2017.12.19	Leading 版の初期バージョン。
v2.6	2018.2.2	サーチウィンドウ外からデータが返ってくるバグを解決。
v3.2	2017.12.19	Leading/Trailing 版の初期バージョン。
v3.3	2018.2.2	サーチウィンドウ外からデータが返ってくるバグを解決。

モジュール動作概要

Mezzanine HR-TDC を HUL HR-TDC BASE を接続した状態のブロック図を図 20 に示します。制御系が他のファームウェアよりも複雑なため、多少詳しく描いています。HR-TDC のシステムはそれぞれの FPGA に BCT が存在し、Mezzanine HR-TDC 側は BASE 側の MIF を通じて 2 段階アクセスでレジスタを制御します。MIF を通じた Mezzanine 側の制御については専用の C++関数が用意されており、ソフトウェアの節でもう一度詳しく説明します。

データ取集側は時間測定のみを行う Mezzanine HR-TDC と、Event build、Fast clear/Level2 への応答、各 IO の管理を行う BASE 側に分かれます。特に Trigger や Event TAG の管理は他のモジュール同様に Trigger Manager (TRM)が行います。Mezzanine HR-TDC へは TRM を出た Level 1 trigger のみが送信されます。この Lvel 1 が Common stop となり HR-TDC へ入力されます。TDC としての動作は HUL MH-TDC と同等です。時間精度だけがよりよくなったと思ってください。ヒットを記録するための Ring buffer は 15.7 us、時間分解能は 25 ps (σ) (common stop に対して)、20 ps (σ) (チャンネル間の

引き算)です。

DAQ データを Mezzanine から BASE へ高速で転送するために、5 線の 128 MHz (256 Mbps) DDR を使ってデータを送っています。制御ビットを含んで転送しているため 5 線の帯域を全て使えているわけではなく、1 ワード (32 bit) 転送するのにかかる時間は 30 ns (1 Gbps) です。BASE 側の DDR receiver は電源投入後に一度初期化する必要があります。この方法についてもソフトウェアの節で説明します。TDC Base は送られてきたデータを見て、イベント event builder に渡す準備をします。この部分が他のファームウェアの計測ブロックの模擬をしていると思ってください。

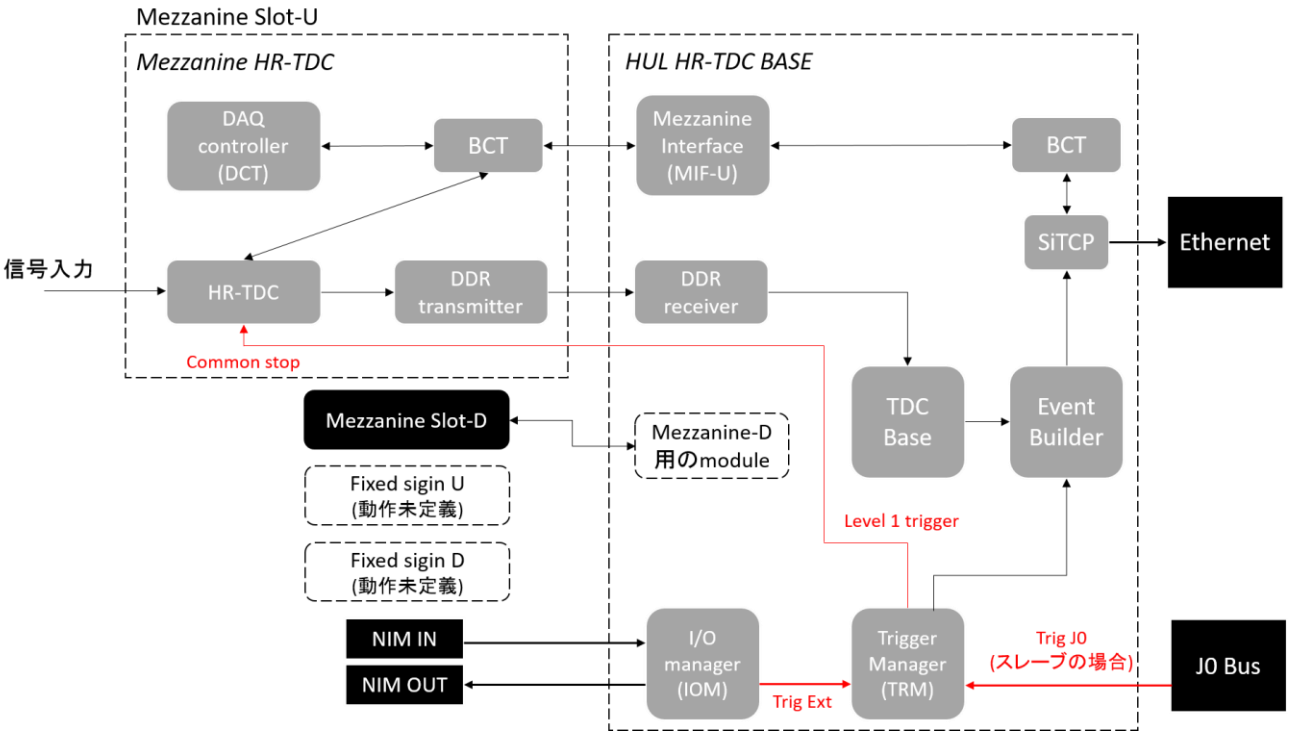


図 20：Mezzanine HRTDC と HUL HR-TDC BASE のブロック図。Mezzanine Slot-D については Slot-U と同様のため省略。

3.4.1 Mezzanine HR-TDC についての詳細説明

高分解能時間測定の原理

本ファームウェアは Tapped delay line（TDL）方式という時間測定方法をとっています。TDL の概念を図 21 に示します。TDL はごく細かい遅延量を持つ素子を直列につないだ遅延ライン上を、入力信号がどこまで走ったかをフリップフロップで記録することにより、クロックエッジ間の時間情報の補間を行う技術です。図 21 の灰色の四角形は遅延素子を示し、各遅延素子の間の情報を D-FF アレイが毎クロックスナップショットを取っています。遅延素子を以後 Tap と呼ぶことにします。スナップショットを取るためのクロックは 520 MHz (1.92 ns)のため、1.92 ns の間にパルスが到達可能な最大の Tap 番号が分かれば Tap 当たりの遅延量 dT が分かります。これは十分な数の統計があれば、Tap 番号のエンドポイントがそれに相当します。普通の TDC ではここで「1920 ps/最大 Tap 番号」が Tap あたりの dT であり、TDC 1bit 分の時間になります。ところが、FPGA HR-TDC では Tap の遅延量はすべて異なるので、

静的な校正ではなく、全ての Tap 番号を時間に変換する動的な校正が必要になります。ここで、Tap 番号を fine count、fine count から時間に変換された値を esitimator と呼びます。

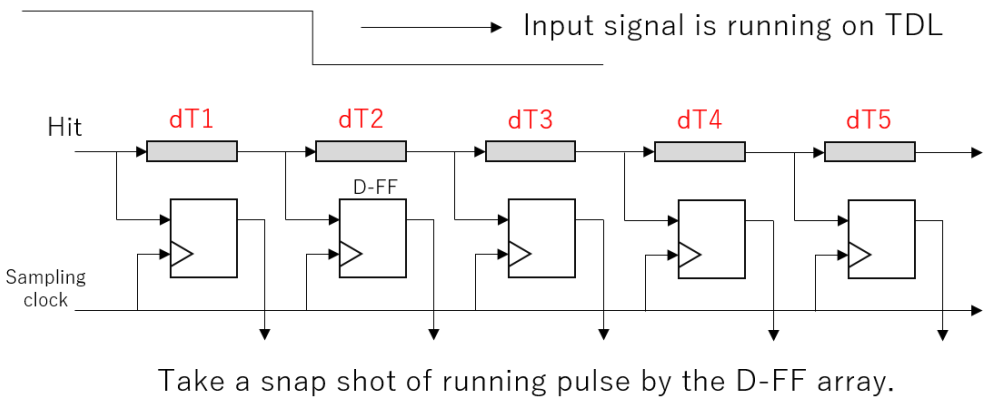


図 21：TDL の概念図。灰色の四角形が遅延素子であり、この遅延素子を直列につないだ線上を入力信号が走る。その様子を D-FF アレイで記録することにより、パルスがどこまで走ったかが分かる。これにより、クロックエッジ間の時間情報の補間が行える。

時間校正

Estimator を生成するための手順を図 22 に示します。まず、無作為に入力信号をサンプリングし fine count のヒストグラムを生成する必要があります。ここで histogram の総エントリー数は固定値であり、ある値（0x7ffff)に達するまで histogram へのデータフィルを続けます。Histogram が用意出来たら各ビンカウントを積分し Estimator を生成します。i 番目のビンのエントリー数を w_i とすると、N 番目の Tap でパルスの先頭が見つかった場合の estimator は

$$E_n = w_n/2 + \Sigma^{n-1}_0 (w_i)$$

として計算されます。そのため、FPGA 内部には fine count histogram と estimator を生成するための機構、更に毎ヒット fine count から estimator へ変換して出力する回路が必要になります。また、histogram を生成するための機構が必要になります。最も分かりやすい方法は検出器の信号を使って fine count histogram を生成する方法です。（検出信号はランダムである必要があります。）Histogram 生成は DAQ とは無関係であるため、入力信号は全て自動的に histogram へフィルされていきます。しかし、この方法では 0x7ffff までイベント溜まるまで待たないといけなく、また、検出器が繋がっているチャンネルでしか利用できません。そのため、クロックを使って histogram を生成する方法を用意しています。FPGA 内部で校正専用のクロックを全入力ラインに接続します。このクロックは DTL をサンプリングするクロックに対して毎エッジ少しずつ位相がずれていくように調整されています。原理的には 1ps ずつ異なった場所に校正クロックのエッジを立てることが出来ます。校正クロックを使った方法では数十 ms で histogram を生成することが出来ます。電源投入後のモジュールの初期化や RUN の最初に構成することを想定しています。

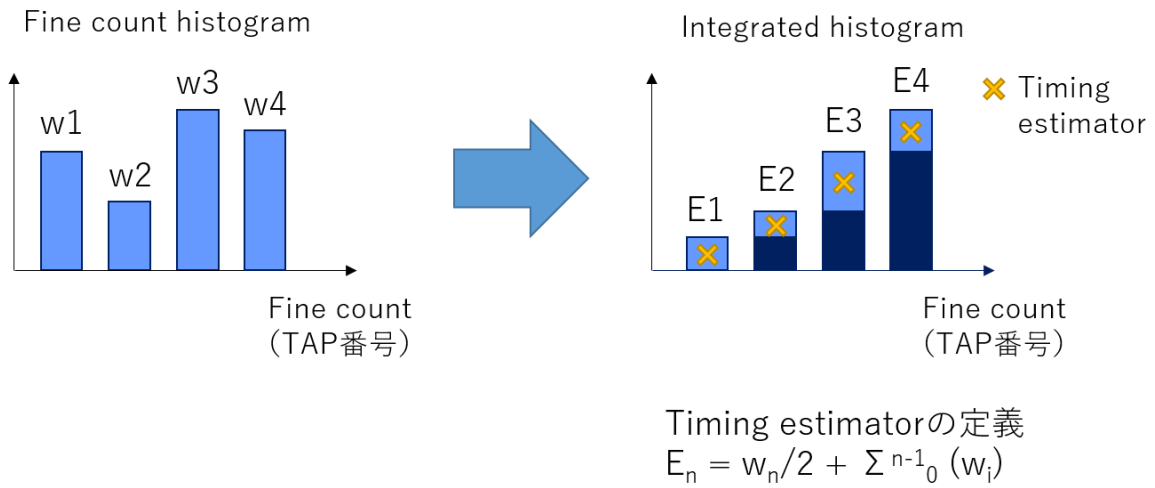


図 22：Fine count histogram から Esitomator を生成する手順。

実装されている校正ブロックと動作

時間校正のシステムは 2 つの RAM によって実装されます。図 23 に示すように fine count は 2 つの RAM へ同時に入力されます。片方の RAM は fine count から estimator へ変換する仕事をします。生の estimator は 19bit 幅でそのままでは過剰なので、下位 8bit を捨てて 11bit 幅にしてから出力します。対してもう一方の RAM では histogram の生成を行っています。0x7ffff イベント溜まった時点で estimator へ変換し、RAM のスワップ待ちの状態になります。RAM スワップは片方が準備できたら自動的に切り替えるか、手動で気切り替えるかを選ぶことが出来ます。これをつかさどっているのが、reg_autosw と reg_switch です。autosw が 1 であれば自動切り替え、一方で autosw が 0 の時に reg_switch へ書き込み動作をすると RAM が主導で切り替わります。自動切り替えは RUN 中であっても検出器の信号を使って RAM を常に更新したい場合に利用します。

また、estimator へ変換せずに fine count をそのまま取り出したい場合もあると思います。その場合は、reg_through を 1 にすることで fine count が直接現れます。

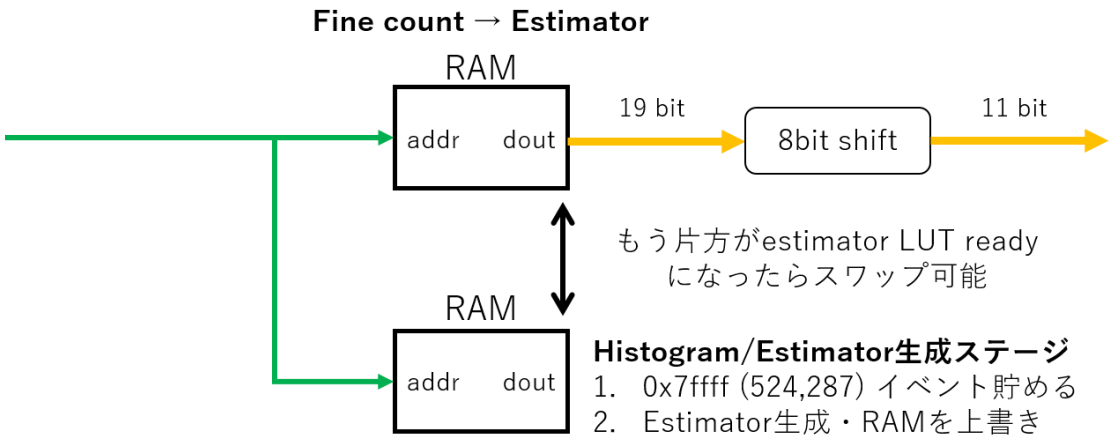


図 23：時間校正システムの概念図。

HR-TDC system

Mezzanine HR-TDC 内部の HR-TDC 機能についてまとめます。本ファームウェアでは TDL の長さは 192 taps ですが、そのままだと遅延時間が細かすぎるため、3 つの tap を 1 つにまとめて 64 の有効 tap に変換しています。そのため、fine count の最大値は 63 となります。測定した結果では、1.92 ns でパルスが走る距離はおおよそ 55 taps です。出力された fine count は 130 MHz のクロック領域に渡され、estimator へ変換されたのち、ring buffer へ hit bit と一緒に書き込まれます。図 24 にあるように estimator (11bit) + semi-coarse count (2bit) + coarse count (11bit)で 24bit のデータ長になります。その後、Mezzanine HR-TDC の中で一度 event build され、HUL 側へ転送されます。あんちょこに時間に直すには estimator の最大値 2048 と 520 MHz のクロックの値を使って、

時間 = TDC value/2048/0.52 (GHz)

とすることで時間 (ns)に直ります。もっと正確に時間に直した場合は TDC calibrator を使ってください。

Ring buffer 以降の実装は MH-TDC と同一です。L1 trigger (common stop) を検出すると Ring buffer からの読み出しを開始します。この時、ヒットを探す範囲を min_max レジスタおよび win_min レジスタによって設定可能です。これらのレジスタは 1bit がコースカウント精度の 11bit の整数値です。この範囲に入らないヒットは Channel buffer へ書き込まれません。Ring buffer からヒット情報を探している間＝サーチ窓幅分の BUSY が出力されます。**MH-TDC とは使っているシステムクロックが異なるため、coarse count 精度が違うことに注意してください。**

その後 channel buffer から block buffer へデータをまとめる際に、1ch/event に許される最大ヒット数が設定されます。1ch/event に許されるヒットは TDC ch の大きいほうから 16 ヒットまでで、それ以上のヒットが channel buffer に記録されていた場合、16 を超えるデータは破棄され、overflow ビットが立ちます。

High-Resolution-Multi-Hit TDC 仕様	
TDC 精度	~30 ps
コースカウント精度	7.69… ns
Ring buffer 長	15.8 us
時間分解能	20 ps (σ) *実測
最小パルス幅	～2 ns
ダブルヒット分解能	～4 ns
最大ヒット数/ch/event	16

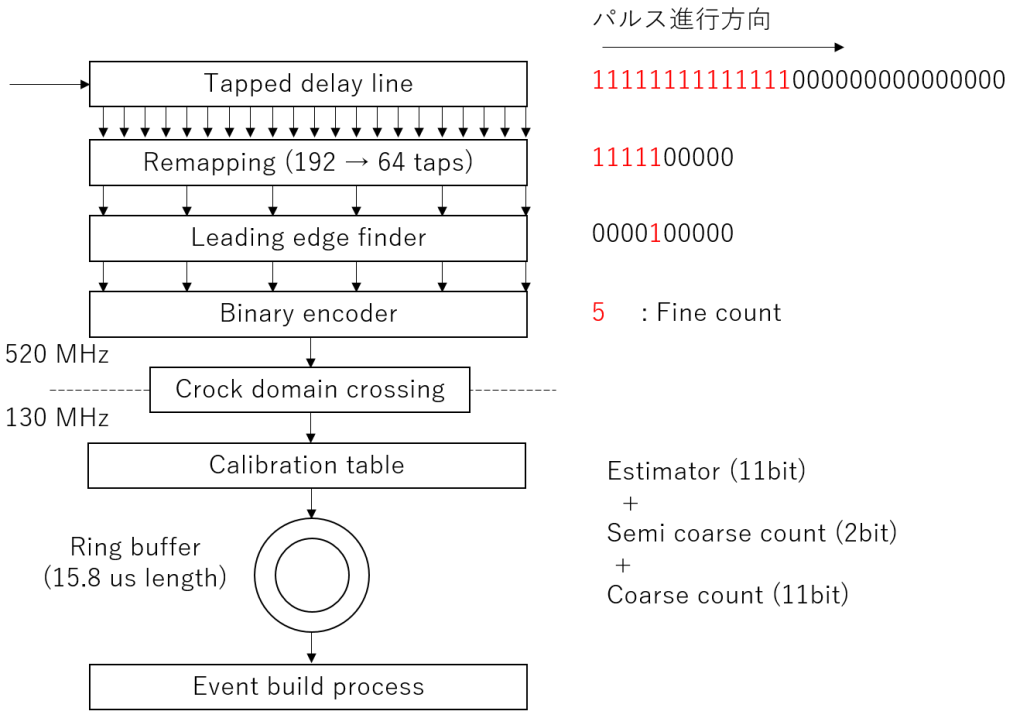


図 24：HR-TDC のブロック構成。

レジスタマップ

FPGA 内部のモジュール ID、アドレス、レジスタについてまとめます。**ここに記述しているレジスタは Mezzanine HR-TDC の物であり、RegisterMap.hh 内で namespace HRTDC_MZN に属するものです。**Mezzanine HR-TDC では配線数が限られていることから local address 幅が 8bit に縮められています。

DAQ controller：DCT (module ID = 0x0)				
レジスタ名	**アドレス**	**Read/Write**	**ビット幅**	**備考**
test_mode	0x00	R/W	1	HUL 側の DDR receiver を初期化するために DDR transmitter からテストパターンを出力するモード。電源投入時のモジュール初期化に必要で、配布している C++ソフトウェアの ddr_initialize 内部で使用する。
extra_path	0x10	R/W	1	このビットを立てると信号入力経路が基板上の入力ポートから校正クロックに切り替わる。前述の estimator を校正クロックによって生成するために使用。
gate	0x20	R/W	1	DAQ gate です。1 だと common stop が HR-TDC へ入力される。

TDC : TDC (module ID = 0x1)				
control	0x10	R/W	3	HR-TDC の動作を変えるためのレジスタ。以下の 3 つのビットが存在。 ● through (0x1) ● autosw (0x2) ● stop_dout (0x4) through が 1 だと fine count は estimator へ変換されずにそのまま出力される。 autosw が 1 だと前述の estimator RAM が準備出来次第 RAM のスワップを行う。 stop_dout が 1 だと FPGA 内部で common stop との引き算を取らずに、stop のデータも 1 ワードとして転送する。
switch	0x20	W	-	autosw が 0 の場合、このレジスタへ書き込みを行おうとすると Estimator RAM のスワップを行う。
status	0x30	R	1	1 であれば次の Estimator RAM の準備が出来ていることを表す。
ptrofs	0x40	R/W	11	内部制御変数。ユーザーは触らない。
win_max	0x50	R/W	11	Ring buffer からヒットを探す時間窓の上限値。1bit が 7.69... ns に相当。詳細は MH-TDC を参照。
win_min	0x60	R/W	11	Ring buffer からヒットを探す時間窓の下限値。1bit が 7.69...ns に相当。詳細は MH-TDC を参照。
Bus Controller : BCT (module ID = 0xE)				
Reset	0x00	W	—	Bus Controller からモジュールリセット信号をアサートし､SiTCP を除く全モジュールを初期化。
Version	0x10	R	32	Firmware の固有名とバージョンを読み出す。多バイト読み出しが必要。
ReConfig	0x20	W	—	PROG_B_ON を Low にして FPGA の再コンフィギュレーションを行う。一度通信が切れるので暫くしてから再接続。

DIP SW の機能

基板上の 4bit DIP スイッチに割り当てられている機能です。

スイッチ番号	機能	詳細
1	HUL clock	Mezzanine HR-TDC のクロック入力を選択します。Off で基板上の発振器を使い、On で HUL から送られてくるクロックを使います。
2	未実装	
3	未実装	
4	未実装	

LED 点灯の機能

Mezzanine HR-TDC にはユーザーが利用できる LED はありません。赤い LED は FPGA がコンフィグされていることを示す LED です。

Module Busy となるタイミング

Mezzanine HR-TDC の BUSY の定義は以下に列挙する BUSY 信号の OR になります。実際には更に HUL HR-TDC BASE の BUSY も OR したものがシステムの BUSY となります。普段は Sequence busy 分の BUSY 長となるはずです。

BUSY	BUSY 長	備考
Sequence busy	サーチ幅に依存	Ring buffer からヒット情報を探している間、つまり win_max – min_min 分の BUSY が出力されます。
Block full	—	ブロックバッファが Full になった段階で BUSY が出力されます。L1 trigger レートが後段の回路のデータ処理速度を上回るとアサートされます。この BUSY が立つようであれば HUL BASE の方も何かしら BUSY が立っていると思われます。

3.4.2 HUL HR-TDC BASE についての詳細説明

HUL HR-TDC BASE は DDR receiver と MIF を除けばその機能は殆ど MH-TDC と同じです。ですが、**MH-TDC と違い HRM を実装することはできません。**そのため、J0 bus master になることはできません。

DDR receiver は電源投入後一度初期化する必要がある以外はユーザーが能動的にアクセスすることはありません。MIF は Mezzanine slot up と down にそれぞれ独立で用意されています。MIF は BASE 側の BCT から制御されつつ Mezzanine 側の BCT へアクセスを行わないといけないため、MIF 自体のア

ドレスと Mezzanine 側のアドレス両方を指定する必要があります。この動作を簡単にするために一度 Mezzanine へアクセスるためには 2 度のアクションを行う必要があります。1 回目のアクションで、MIF に対してアドレス MIF::reg_mif を指定して、Mezzanine 側の module ID、local address、および書き込むべきレジスタ値（書き込みを行う予定の場合）を MIF 内部へ格納します。2 回目のアクションで MIF::connect に書き込み、もしくは読み出しを試みます。MIF::connect は特殊なアドレスで、これが指定されると MIF は Mezzanine 側の BCT 制御のサブシーケンスを動かします。MIF::connect へ書き込みモードでアクセスすれば、1 回目のアクションで指定したアドレスへ指定したレジスタを書き込みます。一方読み出しモードでアクセスすれば、指定したアドレスから返ってきた Mezzanine 側のレジスタ値が MIF から返ってきます。MIF::connect はサブシーケンスの処理が正しく終わるまで HUL 側の BCT へ応答しません。**そのため、例えば Mezzanine HR-TDC が刺さっていないのに MIF::connect を呼ぶと HUL 側の BCT がデッドロックします。**こうなると BCT::Reset を呼ぶことが出来なくなるので SiTCP Reset で対応する必要があります。MIF 制御のための C++関数は mif_func.cc にまとめられています。詳細は 4 章で述べます。

HR-TDC BASE 固有名と現在の最新版

Mezzanine HR-TDC を制御するためのファームウェアです。Mezzanine HR-TDC の v2 も v3 もこのファームウェアで制御できます。

固有名	**0x80EB**
メジャーバージョン	**0x01**

更新歴

バージョン	リリース日	変更点
v1.5	2017.12.19	初期バージョン。
v1.6	未公開	
v1.7	2018.2.2	J0 バスからやってくるイベントタグをラッチするタイミングが早すぎて、HRM 側のイベント番号と 1 ずれるバグを解決。 BCT::Reset を呼ぶと BCT がハングするバグを解決。

レジスタマップ

これは HUL HR-TDC BASE 側のレジスタマップであり、RegisterMap.hh 内部で namespace HRTDC_BASE に属します。Mezzanine 側と名前が被るものもあるので、必ず namespace で指定してください。IOM において HRM をサポートしなくなったことによっていくつかのレジスタが機能しなくなっています。機能しないものは取り消し線が引いてあります。

加えて本ファームウェア用の RegisterMap.hh には先頭に en_up と en_down というグローバル変数が記述されています。これはどのスロットに Mezzanine HR-TDC が刺さっているかを示すフラグです。**Mezzanine HR-TDC 刺さっていないスロットは false にしてください。**

Trigger Manager : TRM (module ID = 0x0)				
レジスタ名	アドレス	Read/Write	ビット幅	備考
sel_trig	0x000	R/W	12	TRM 内部のトリガーポートの選択を行うレジスタ。
DAQ controller : DCT (module ID = 0x1)				
gate	0x000	R/W	1	DAQ gate の 0/1 設定。DAQ gate が 0 だと TRM は trigger を出力できない。簡易なシステムなどでイベント同期を取る際などに利用する。
evb_reset	0x010	W	—	このアドレスへ書き込み要求することで Event Builder のソフトリセットがアサートされ、Event Builder 内部のセルフイベントカウンタが 0 になる。
init_ddr	0x020	W	—	DDR receiver へ向けて初期化要求を行う。
ctrl_reg	0x030	R/W	4	DDR receiver を制御するためのレジスタ。詳細は後述。
status	0x040	R	4	DDR receiver のステータス。詳細は後述。
I/O Manager : IOM (module ID = 0x2)				
nimout1	0x000	R/W	4	NIMOUT1 へ何を出力するかを設定する。
nimout2	0x010	R/W	4	NIMOUT2 へ何を出力するかを設定する。
nimout3	0x020	R/W	4	NIMOUT3 へ何を出力するかを設定する。
nimout4	0x030	R/W	4	NIMOUT4 へ何を出力するかを設定する。
extL1	0x040	R/W	3	extL1 にどの NIMIN を接続するか設定。
extL2	0x050	R/W	3	extL2 にどの NIMIN を接続するか設定。
extClr	0x060	R/W	3	extClr にどの NIMIN を接続するか設定。
extBusy	0x070	R/W	3	extBusy にどの NIMIN を接続するか設定。
~~extRsv2~~	~~0x080~~	~~R/W~~	~~3~~	~~extRsv2 にどの NIMIN を接続するか設定。~~
cntRst	0x090	R/W	3	Mezzanine HR-TDC 内部の coarse count をリセットするハードリセット信号。複数台の HR-TDC を同期したい場合に使用する。 この線にどの NIMIN を接続するか設定。
Mezzanine Interface U : MIF-U (module ID = 0x3)				
connect	0x000	R/W	—	Mezzanine HR-TDC の BCT 制御を行う。アクセスする際のモードが書き込みなのか読み出しなのかで、Mezzanine の BCT へのアクセス方法が切り替わる仕様となっている。
reg_mif	0x010	W	20	Mezzanine HR-TDC 用の local address と書き込み用レジスタ値を MIF に一時的に保

				存する。 [19:16] Module ID [15:8] Local address [7:0] Register value
frst	0x020	W	—	Mezzanine HR-TDC へ強制リセット信号をアサート。DAQ や BCT がハングした場合に使用する。
Mezzanine Interface D : MIF-D (module ID = 0x4)				
MIF-U と同様のため省略				
Bus Controller : BCT (module ID = 0xE)				
Reset	0x000	W	—	Bus Controller からモジュールリセット信号をアサートし、SiTCP を除く全モジュールを初期化。
Version	0x010	R	32	Firmware の固有名とバージョンを読み出す。多バイト読み出しが必要。
ReConfig	0x020	W	—	PROG_B_ON を Low にして FPGA の再コンフィギュレーションを行う。一度通信が切れるので暫くしてから再接続。

Trigger Manager (TRM)

HUL HR-TDC BASE は HRM をマウントすることが出来ないため、TRM の RM に関するレジスタは機能しません。以下のレジスタマップは HUL RM のものと同一ですが、機能しないレジスタは取り消線を引きました。

レジスタラベル	レジスタ値	備考
reg_L1Ext	0x1	NIMIN から L1 trigger を選択。
reg_L1J0	0x2	J0 bus からの L1 trigger を選択。
~~reg_L1RM~~	~~0x4~~	~~Mezzanine HRM からの L1 trigger を選択。~~
reg_L2Ext	0x8	NIMIN からの L2 trigger を選択。
reg_L2J0	0x10	J0 bus からの L2 trigger を選択。
~~reg_L2RM~~	~~0x20~~	~~Mezzanine HRM からの L2 trigger を選択。~~
reg_ClrExt	0x40	NIMIN からの Clear を選択。
reg_ClrJ0	0x80	J0 bus からの Clear を選択。
~~reg_ClrRM~~	~~0x100~~	~~Mezzanine HRM からの Clear を選択。~~
reg_EnL2	0x200	0: L2=L1 trigger、1: L2=L2 入力
reg_EnJ0	0x400	Tag 情報に J0 bus の物を採用する。また、この bit が 1 だと J0 bus へ自身の module busy を流す。

~~reg_EnRM~~	~~0x800~~	~~Tag 情報に J0 bus の物を採用する。~~

DAQ controller (DCT)

DCT 内部の ctrl と status が示すビットの詳細を述べます。

CTRL レジスタの内訳		
レジスタラベル	Bit 番号	備考
reg_test_mode_u	1st bit (0x1)	このビットを立てると、DDR receiver (slot-U)を初期化するために、テストパターンを受信するモードへ切り替える。
reg_test_mode_d	2nd bit (0x2)	このビットを立てると、DDR receiver (slot-D)を初期化するために、テストパターンを受信するモードへ切り替える。
enable_u	3rd bit (0x4)	このビットを立てると DDR receiver (slot-U)が利用可能になる。
enable_d	4th bit (0x8)	このビットを立てると DDR receiver (slot-D)が利用可能になる。
Staus レジスタの内訳		
reg_bit_aligned_u	1st bit (0x1)	DDR receiver (slot-U)の bit slip が終了し、データ読み出しが可能になった事を示す。
reg_bit_aligned_d	2nd bit (0x2)	DDR receiver (slot-D)の bit slip が終了し、データ読み出しが可能になった事を示す。
reg_bit_error_u	3rd bit (0x4)	DDR receiver (slot-U)の bit slip を一定回数施行したが、正しい結果が返ってこなかった事を示す。初期化失敗。
reg_bit_error_d	4th bit (0x8)	DDR receiver (slot-D)の bit slip を一定回数施行したが、正しい結果が返ってこなかった事を示す。初期化失敗。

I/O Manager (IOM)

HRM をサポートしないため複数のレジスタが機能しません。

NIMOUT へ出力可能な信号線		
レジスタラベル	レジスタ値	備考
reg_o_ModuleBusy	0x0	Module busy です。Module busy は自身の内部 busy のみを指します。J0 bus の busy や ExtBusy は含まれません。
~~reg_o_CrateBusy~~	~~0x1~~	~~CrateBusy です。CrateBusy は module busy に加えて J0 bus の busy や ExtBusy を含みます。J0 bus マスタの場合に利用する信号になり、また RM が Trigger Module へ返す busy と同等です。~~
~~reg_o_RML1~~	~~0x2~~	~~HRM が受信した L1 trigger を出力します。~~
~~reg_o_RML2~~	~~0x3~~	~~HRM が受信した L2 trigger を出力します。~~

~~reg_o_RMClr~~	~~0x4~~	~~HRM が受信した Clear を出力します。~~
~~reg_o_RMRsv1~~	~~0x5~~	~~HRM が受信した Reserve 1 を出力します。~~
~~reg_o_RMSnInc~~	~~0x6~~	~~HRM が Spill Number Increment を出力します。~~
reg_o_DaqGate	0x7	DCT の DAQ gate を出力します。
reg_o_DIP8	0x8	DIP SW2 8 番のレベルを出力します。
reg_o_clk1MHz	0x9	1 MHz のクロックを出力します。
reg_o_clk100kHz	0xA	100 kHz のクロックを出力します。
reg_o_clk10kHz	0xB	10 kHz のクロックを出力します。
reg_o_clk1kHz	0xC	1 kHz のクロックを出力します。
内部信号線へ割り当て可能な NIMIN ポート		
reg_i_nimin1	0x0	NIMIN1 番を信号線へアサインします。
reg_i_nimin2	0x1	NIMIN2 番を信号線へアサインします。
reg_i_nimin3	0x2	NIMIN3 番を信号線へアサインします。
reg_i_nimin4	0x3	NIMIN4 番を信号線へアサインします。
reg_i_default	0x7	このレジスタが設定された場合、指定のデフォルト値がそれぞれの内部信号線へ代入されます。

IOM には初期割り当てが存在します。その初期値を以下に列挙します。

NIM 出力ポート	**初期レジスタ**	
NIMOUT1	reg_o_ModuleBusy	
NIMOUT2	reg_o_DaqGate	
NIMOUT3	reg_o_clk1kHz	
NIMOUT4	reg_o_DIP8	
内部信号線	**初期レジスタ**	**デフォルト値**
ExtL1	reg_i_nimin1	NIMIN1
ExtL2	reg_i_default	0
ExtClear	reg_i_default	0
ExtBusy	reg_i_nimin3	NIMIN3
~~ExtRsv2~~	~~reg_i_nimin4~~	~~NIMIN4~~
cntRst	reg_i_default	0

DIP SW2 の機能

DIP SW2 に割り当てられている機能を列挙します。

スイッチ番号	機能	詳細
1	SiTCP force default	ON で SiTCP のデフォルトモードで起動します。電源投入前に設定している必要があります。

2	NC	
3	Force BUSY	Crate Busy と Module Busy を強制的に High にします。接続チェックなどに使ってください。
4	Bus BUSY	ON で Crate Busy に J0 bus busy を含め、OFF で含めません。
5	LED	ON で LED4 番を光らせます。
6	NC	
7	NC	
8	Level	IOM の DIP8 から出力されるレベルです。

LED 点灯の機能

LED が点灯した際の意味です。

LED1	点灯中は TCP 接続が張られています。
LED2	点灯中は module busy が high です。
LED3	DAQ gate が ON であると点灯。
LED4	常に消灯。

DAQ の動作

データフローを図 25 に示します。Mezzanine HR-TDC と BASE が 2 つの FPGA へ分かれていますが、見かけの動作は HUL MH-TDC と変わりません。各 Mezzanine では Block buffer に相当する部分までの partial event build が行われデータが BASE へ転送されてきます。BASE 側では受け取ったデータをまとめて Event build します。

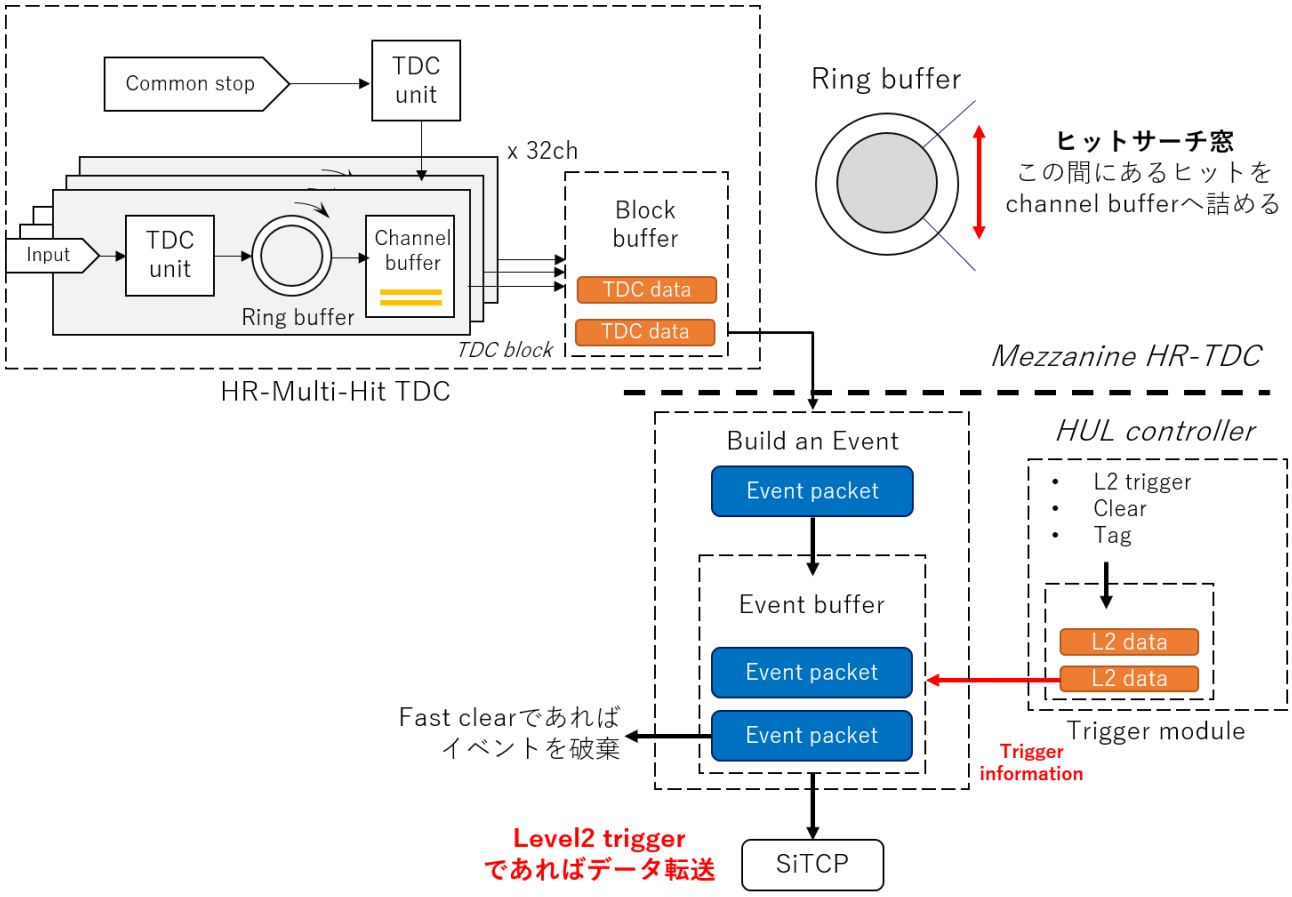


図 25：Mezzanine HR-TDC と HUL HR-TDC BASE を合わせた DAQ のデータフローブロック図

Module Busy となるタイミング

システム全体の BUSY 条件をのべます。Module busy はこれらの OR です。

BUSY	BUSY 長	備考
Self-busy	210 ns	L1 trigger を検出した瞬間から固定長でアサート。
Mezzanine	Mezzanine の状態に依存	Mezzanine HR-TDC が出力している BUSY。通常はサーチ窓幅の BUSY が返ってきます。
Block full	—	TDC Base のブロックバッファが Full になった段階で BUSY が出力されます。L1 trigger レートが後段の回路のデータ処理速度を上回るとアサートされます。つまり TCP 転送が負けていることを意味するので、実質的に TCP full と同等です。
TCP full	—	SiTCP の TCP バッファが Full になると出力されます。ネットワーク帯域に対して Event Builder が送信しようとするデータ量が多いとアサートされます。

データ構造

ヘッダワード

ヘッダ 1 (マジックワード)

上位ビット				下位ビット
0xFFFF800b				

ヘッダ 2 (イベントサイズ)

0xFF	0x00	OverFlow	“0000”	Number of Word (11bit)

Number Of Word はデータボディに含まれるワード数を示します。Number of Word は Sub-header 分の 2 ワード分を含みます。なので最低値が 2 です。Over flow は HUL 全体で 1ch でも over flow チャンネルがあると立ちます。

ヘッダ 3 (イベント番号)

0xFF	“0000”	Tag (4bit)	Self-counter (16bit)

Tag は TRM から出力される 4bit の Tag 情報です。下位 3bit が RM Event Number の下位 3 ビット、4 ビット目が RM Spill Number の最下位ビットとなります。Self-counter はイベント転送を行うたびにインクリメントされる Local Event Number で、0 オリジンです。

サブヘッダ A (Slot-U)

0xFA00	“00”	OverFlow	Stop dout	Through	Number of Word (11bit)

Slot-U にマウントされている Mezzanine HR-TDC のヘッダです。Over flow は Slot-U の中での over flow の存在を示しています。Stop dout と Through はそれぞれ HRTDC_MZN::TDC::controll における stop_dout と through の状態を示します。Number of Word は Slot-U におけるワード数を示します。これはサブヘッダ A からサブヘッダ B までの間にくるデータ数です。

サブヘッダ B (Slot-D)

0xFB00	“00”	OverFlow	Stop dout	Through	Number of Word (11bit)

Slot-D にマウントされている Mezzanine HR-TDC のヘッダです。Over flow は Slot-D の中での over flow の存在を示しています。Stop dout と Through はそれぞれ HRTDC_MZN::TDC::controll における stop_dout と through の状態を示します。Number of Word は Slot-D におけるワード数を示します。これはサブヘッダ B から最後までのデータ数です。

TDC ワード

Magic word (3bit)	Ch (5bit)	TDC (24bit)

Magic word は

6　　Leading

5　　Trailing

4　　Common stop

となります。Ch は 5 bit しかない事が示すように 31ch までしか数えることが出来ません。サブヘッダ A・B に属するデータであるかを見てデコードして、サブヘッダ B に属するのであれば 32ch 足してください。TDC は HR-TDC の節で述べたように Estimator (11bit) + semi-coarse count (2bit) + coarse count (11 bit)で合計 24 bit です。Through が ON の場合 fine count は estimator の位置に現れます。

3.5 HUL SSM (Spill Structure Monitor)

HUL Spill Structure Monitor (SSM)は入力信号をサンプリングしタイムスタンプを付与するfirmware です。SSM は以上 3 つの firmware とは大きく動作が異なり、トリガーやイベントという概念がなく、ゲート信号中の全入力信号にタイムスタンプを付与し、全データを転送するストリーミング型のシステムになります。

Firmware 固有名と現在の最新版

現在の HUL SSM の固有名とバージョンは

固有名	**0x4480**
メジャーバージョン	**0x01**
マイナーバージョン	**0x02**

更新歴

バージョン	リリース日	変更点
v1.0		初期版 （16ch 版、未公開）
v1.1		128ch 版、未公開
v1.2	2017.11.15	64ch 版。サンプリング周波数 300MHz、時間精度 10ns。

モジュール動作概要

HUL SSM の DAQ ブロックはデータストリーミング経路とその制御信号で構成されます（図 26)。信号入力は HUL の fixed input から行い、メザニンスロットは使用しません。入力信号は最初 STM ブロックで信号サンプリング（300 MHz)、およびタイムスタンプの付与（100 MHz）を行う。この段階では並列動作しているため、データ転送を行うためにデータマージを行う。最終的に heart beat module (HBM)が flame packet を挿入し SiTCP へ転送を行う。Flame packet と Heart beat については後述する

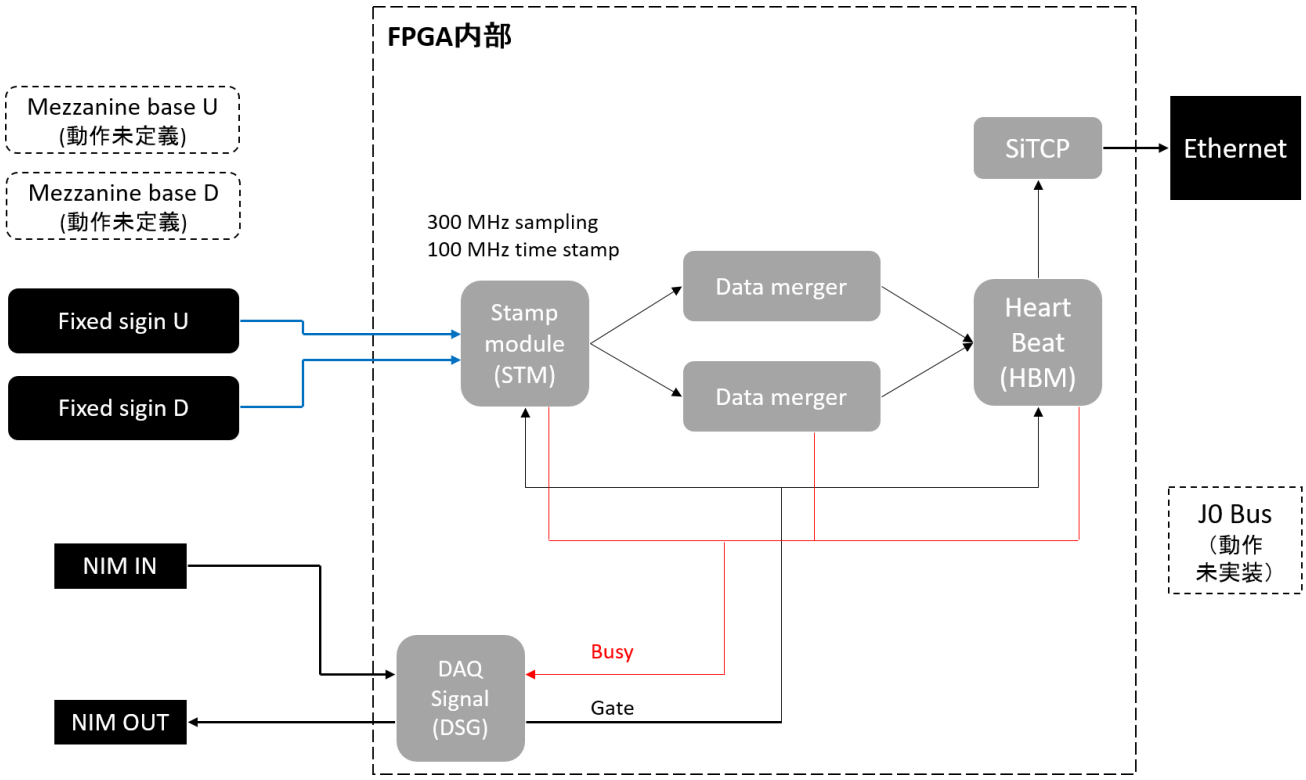


図 26：HUL SSM の DAQ ブロック図。

4.0 ソフトウェア

この章では制御用の Linux ソフトウェアについて述べます。

開発環境

Scientific Linux 6

gcc version 4.4.7

ソースファイル

ソースファイルには他の DAQ システムへ移植する際にそのままコピーするべきなライブラリ的なファイルと、同じように動作さえすればどのように書いてもいい物、そもそもデバッグ用で移植の必要がない物が存在します。

ライブラリ的なソースファイル

network.hh

ただのヘッダ集なのでわざわざ移植する必要もないのですが、ここに udp_port が書かれています。そのまま移植しないのであれば udp_port だけ他の場所に書いてください。

rbcp.h

RBCP パケットの構造体です。そのまま移植してください。

myString

内田さんのソースコードの隠蔽です。同じ機能を他の方法で実現することは可能ですが、既に動いているコードなのでそのまま使っています。ユーザーが直接この関数やクラス呼ぶことはないです。

Uncopyable.hh

自身のクラスのコピーコンストラクタと代入演算子を禁止するメソッドです。これを継承だけでそれらメソッドが禁止されるので楽です。まあ無くてもいいんですが…。

BidDump

整数値を渡して bit 列を標準出力するためのクラスです。元々は Unpacker の中で使われていたものを移植しました。UDPRBCP の中で使っています。デバッグに便利です。

UDPRBCP

SiTCP の UDP 通信を行うためのクラスで、重要なソースコードです。コンストラクタは IP address、upd port、RBCP 構造体の 3 つを必要として、最後の引数は表示モードで、UDPRBCP.hh 内に dbcp_debug_mode という列挙体で定義されています。disp_no など通信時に何も表示しません。

disp_interactive だと何をしている程度の情報を出力し、disp_debug だと全ての情報を標準出力します。通常使う際は disp_no でいいでしょう。UDPRBCP は SiTCP の生の UDP 通信レベルの機能を制御しているので、FPGA 内部で BusController が生の UDP を隠蔽している HUL では UDPRBCP を直接ユーザーが触ることはありません。

FPGAModule

UDPRBCP を隠蔽して BusController レベルの通信機能を提供する最も重要なクラスです。内部で UDPRBCP の実体が生成されるので、コンストラクタは UDPRBCP と同じ引数を必要とします。

int WriteModule(uint module_id, uint, local_addr, uint register)

レジスタを書き込むメソッドです。第一引数は FPGA 内部モジュールのモジュール ID、第二引数はモジュール内部の Local Address です。3 章のレジスタマップ、および後述する RegisterMap.hh を参照してください。第三引数が書き込むレジスタです。**第三引数は変数の型は unsigned int ですが、BCT が扱える書き込みレジスタの幅は 24bit までの事に注意してください。** 戻り値は UDCPRBCP::DoRBCP の結果です。

uint ReadModule(uint module_id, uint, local_addr, int n_cycle)

1 つのレジスタ (1 ワード)を読み出すためのメソッドです。第一、第二引数は WriteModule と同様です。第三引数は多バイト読み出しの回数を設定します。前述の通り BCT 読み出しは 8bit までしか一度に取り扱えないので、8bit 幅を超えるレジスタの場合数回アクセスしなければいけません。その手間を省き一回の関数呼び出しで 1 レジスタを読めるようにした機能です。n_cycle=4 とすれば Local Address の下位ビットをインクリメントしながら 4 回アクセスします。この時、FPGA モジュール内部では多バイト読み出しに対応したコードが書かれている必要があります。読み出し結果は戻り値に現れます。**ただし、レジスタが戻り値に現れることが示しているように、ReadModule では 32bit までしか読めません。そのため、n_cycle は 1-4 の間でしか有効でないので注意してください。**

int ReadModule_nByte(uint module_id, uint, local_addr, int n_byte)

ReadModule は 1 つレジスタを読み出す機能でしたが、ReadModule_nByte は同一のアドレスから連続した値を読み出すための機能です。例えば、ReadModule はアクセス先が DFF を想定していますが、こちらでは FIFO の用に深さ方向が存在するような物から読み出すことを想定しています。第三引数で何バイト読み出すかを指定しますが、UDP バッファサイズを越えて読むことはできません。読み出し結果は ud_data_という内部コンテナに格納されます。戻り値は UDPRBCP::DoRBCP の結果です。

sitcp_controller (2017.05.09 のバージョンより追加)

SiTCP の予約領域に直接アクセスするための関数です。Erase_EEPROM は EEPROM を全消去するための関数です。EEPROM の領域に読み書きする関数は危険なため作っていません。SiTCP Utility を使うことをお勧めします。

void Reset_SiTCP(const char* ip, rbcp_header* header)

SiTCP をソフトリセットします。

void Write_SiTCP(const char* ip, rbcp_header* header, unsigned int addr_ofs, unsigned int reg)

UDP RBCP の SiTCP 予約領域アドレスに 1byte のレジスタを書き込むための関数です。SiTCP の予約領域は 0xffffff00 から始まっており、そこからのオフセットを addr_ofs で指定します。この関数では EEPROM 領域にはアクセスできません。

void Read_SiTCP(const char* ip, rbcp_header* header, unsigned int addr_ofs)

UDP RBCP の SiTCP 予約領域から 1byte のレジスタを読み出すための関数です。addr_ofs の意味は Write_SiTCP と同様です。

void Erase_EEPROM(const char* ip, rbcp_header* header)

EEPROM を全消去するための特殊な関数です。IP アドレスや MAC やライセンス情報など全部消えるため、SiTCP のライセンスファイルは再書き込みする必要があります。この関数は BBT の SiTCP コミュニティに報告が挙がっている“一度 TCP コネクションを張るとしばらく再接続することができなくなる“問題の解決に使います。詳しくは該当スレッドを見てください。この関数では UDP アクセスを行うたびに sleep(1)しているため、非常に時間がかかります。(多分コメントアウトしても大丈夫。)

mif_func (HUL HR-TDC BASE のみに付属)

void WriteMIFModule(FPGAModule& fModule, unsinged int mid_base,
unsigned int mid_mif, unsigned int addr_mif, unsigned int wd,
int n_cycle)

MIF を通じて Mezzanine HR-TDC へ書き込みを行うための関数です。第一引数は生成済みの FPGAModule の実体を渡してください。第二引数は BASE 側の module ID です。MIFU か MIFD が入ります。第 3 引数第 5 引数までは Mezzanine HR-TDC 側の module ID, Local address, 書き込みレジスタです。MIF は 8bit 毎しかデータのやり取りができないので、多バイト書き込みに対応しています。8bit 以上書き込む場合は n_cycle を 1 から増やしてください。

```
void ReadMIFModule(FPGAModule& fModule, unsinged int mid_base,
                   unsigned int mid_mif, unsigned int addr_mif,
                   int n_cycle)
```

MIF を通じて Mezzanine HR-TDC から読み出しを行うための関数です。第一引数は生成済みの FPGAModule の実体を渡してください。第二引数は BASE 側の module ID です。MIFU か MIFD が入ります。第 3 引数第 4 引数までは Mezzanine HR-TDC 側の module ID, Local address です。MIF は 8bit 毎しかデータのやり取りができないので、多バイト読み出し対応しています。8bit 以上読み出す場合は n_cycle を 1 から増やしてください。

RegisterMap.hh

FPGA 内部のアドレスなどを格納したヘッダです。3 章で述べたように同一のモジュール名やレジスタ名であっても ID やアドレス値は異なります。複数のファームウェアに 1 つのソースコードからアクセスする場合 namespace を使って明確に区別してください。

べた移植する必要は無いが同様の機能を提供しないといけないコード

daq_func

DAQ を行うための関数をまとめたソースコードです。ベタ移植する必要は無いですが、ConnectSocket、Event_Cycle、receive 関数はそのまま移植したほうがいいでしょう。また、MH-TDC の set_tdc_window もそのまま移植してください。基本的には計測ブロックの設定を行い、DCT で gate を開いて、Event_Cycle を呼び続ける、RUN が終わったら DCT で gate を閉じる、という流れになります。

Gate を閉じた後タイムアウトするまで Event_Cycle を呼びつづける事で HUL 内部のバッファを空にしているので、while(-1 != Even_Cycle)の処理は必ず行ってください。これをやらないと次の RUN の先頭に前の RUN で読まれなかったイベントが返ってくる可能性があります。

~~gzfilter~~

~~basic_streambuffer を継承してストリーム処理に zlib 圧縮・展開機能をつけたストリームクラスです。別にこの方法でなくても zlib 圧縮は行えます。HDDAQ ではいらないクラスです。~~

2017.12.19 のバージョンから削除しました。コンパイルできるかどうかが gcc バージョンに強く依存し、問題をたびたび引き起こすためです。。

移植する必要の無いソースコード

CommandMan

対話モードで RBCP を行うためのクラスです。完全にデバッグ用なので FPGA ファームウェアを開発する人以外はいらないでしょう。

各メイン関数

制御の例題として使ってください。

5.0 実践的な使い方

この章では多分使っていく上で知らないと困ることを何点かあげます。

MCS の生成方法

Vivado を使った MCS の生成方法を説明します。iMpact は持っていない人もいると思うので省略します。この操作を行うためには Vivado 2016.1 以上である必要があります。

まず Vivado でプロジェクトを開き、tool → Generate Memory Configuration File を選択します。出てきた画面を図 27 の様に埋めて OK を押すと MCS が生成されます。

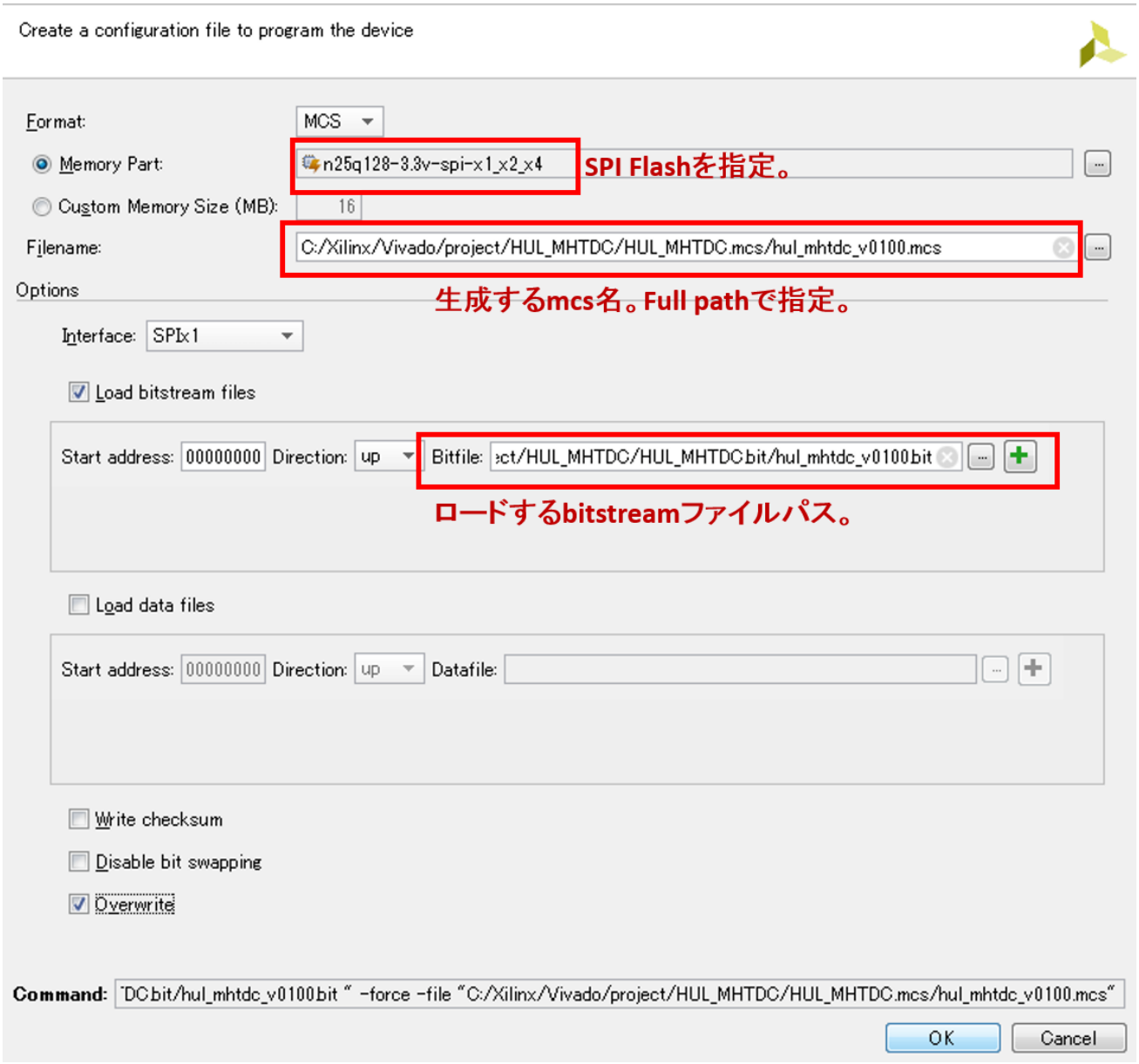


図 27：MCS 生成画面。

Hardware manager で MCS をダウンロードする

HUL に電源が入った状態で JTAG ケーブルを接続します。この時 Xilinx 純正であれば LED が緑色になります。仮想マシンなど立ち上がっているとクライアントに USB 接続が行くことがあるのでオレンジのままならそれらであることが多いです。Vivado で Open Hardware manager → Open target で FPGA にアクセスします。この段階で Kintex7 が見えていると思います。Bit stream ファイルを書き込むならこの段階で FPGA を右クリックして Bit stream ファイルをアサイン、書き込みを行えば OK です。ところが、MCS の場合 SPI に入れないのいけないのですが画面上に出ていません。これは SPI flash が JTAG チェイン上に存在しないためです。

まず configuration memory device を追加します。FPGA を右クリックし、Add configuration memory device を選択します。図 28 のような画面が出るので Memory device に MCS を作った際に選択した Micron の SPI flash を選択します。その下に書き込む MCS と PRM 両方を選択して、他は全てデフォルトにしておいてください。後は書き込みを行うだけです。大体書き込むのに 10 分くらいかかります。

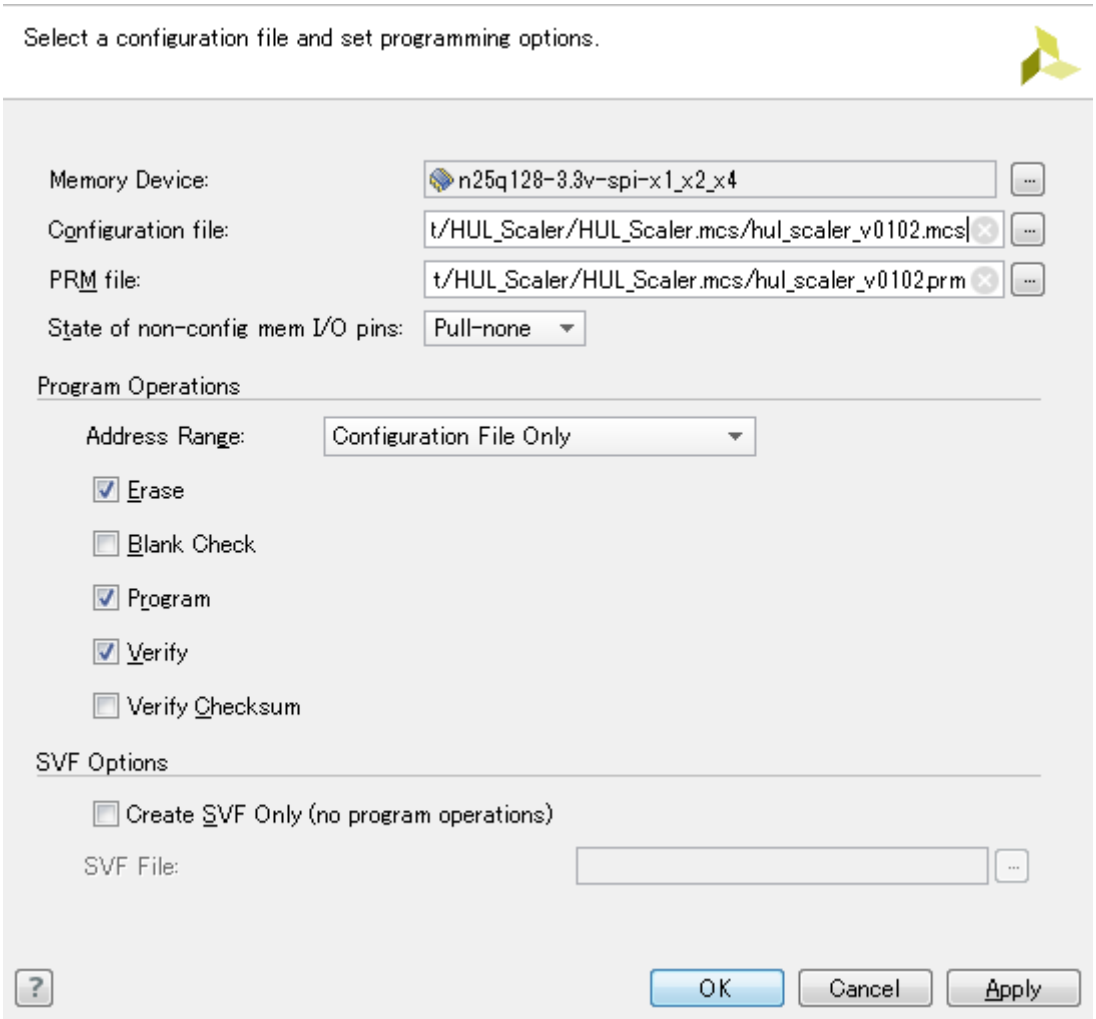


図 28：Configuration Memory Device 追加画面。

JTAG ケーブルをつないで Hardware manager を立ち上げたまま HUL に電源を入れると FPGA の初期化が終わらないことがある。

~~タイトルの通りです。再現性が微妙でうまくいくときもあります。なぜなのか分かったら教えてください。~~

Hardware manager が起動した状態、かつ JTAG サーバが Open の状態でダウンロードケーブルをつないだまま電源を投入すると MCU からの初期化が終わらないことが分かりました。
Hardware manager が立ち上がっていてもサーバを Close していればこの問題は回避できます。

初回電源投入時には Bit stream ファイルを書くことができない。

これは PROG_B が FPGA のユーザーポートにつながっている事が原因です。何も入ってない状態の FPGA は足がプルダウンされるのかもしれません。一度 MCS を SPI Flash に書き込んで、電源を On/Off すれば FPGA が特定のファームウェアで初期化されて足の電位が決まるので、Bit stream ファイルを書き込むことができるようになります。

ヒートシンクを取り付ける

HUL は大量のチャンネルを取り扱うこともあり、ファームウェアによっては消費電力が 2W に迫ります。FPGA が熱を持って嬉しいことは何も無いので（というか熱を持つとリーク電流が急激に増えて大変なことになる)、可能な限り空冷するようにしてください。ただ、HUL はクレートにささないことも想定しているので強い風を当てられないこともあると思います。弱い風でも効率的に冷やすためにヒートシンクを取り付けることをお勧めします。適合ヒートシンクは Advanced Thermal Solutions Inc.社の ATS-51270D-C1-R0 です。図 29 のように取り付けてください。赤い台を取り付けるのがちょっと難しいです。FPGA の足をはがさないように注意してください。

また、この FPGA にはパッケージ上面にパスコンが載っています。ヒートシンクを取り付けると金属面がこのパスコンのすぐそばにくるため、安全のためにもカプトンをパスコンにはってから取り付けることをお勧めします。

図 29：ヒートシンクを取り付けた状態の FPGA

ジー・エヌ・ディーから購入したい

有限会社ジー・エヌ・ディーにメールを送ってジーエヌディー管理番号のモジュールを何台作るのか書いてください。HUL controller ではれば SiTCP ライセンスはをつけるように指示し、FPGA を向こうで手配するように言ってください。

少数注文すると非常に割高になります。これは単純に基板製造のイニシャルが全部跳ね返ってくるからです。また、PCB 基板の面付けの都合で、HUL controller の PCB は一度の製造で 2 枚、Mezzanine card は 4 枚単位でしか生産できません。そのため、その倍数で頼まないと無駄な PCB 基板が生まれるのでその分割高になります。可能な限り共同購入するようにしましょう。

N25Q128A の取り扱いについて

この SPI flash memory は HUL を設計した時に採用した Micron の石ですが、生産終了となっています。そのため、すでに最新版の Vivado では正式サポートから外れています。後継版の石である MT25 系と内部的には互換らしいので、N25 用の MCS を生成する際には MT25QL128 を選択して MT25QL として作ってください。また、ダウンロードする際には同じように MT25QL としてダウンロードしてください。

SPI flash を MT25QL512 に変更した基板について。

2017 年後期の共同購入以降、SPI flash memory が MT25QL に変更になっています。ソースコードから合成して MCS ファイルを作る場合、図 27 に書かれている SPI メモリから **mt25ql512_spi-x1_x2_x4** 変更してください。同様に Hardware manager でダウンロードする場合も図 28 に示す memory device から **mt25ql512-spi-x1_x2_x4** へ変更してください。

SPI flash を S25FL256SAGNFI001 に変更した基板について。

2018 年度から購入可能であった HUL には Spansion の S25FL256SAGNFI001 が搭載されています。この石はこれまで使っていた Micron とメーカーが異なるため、まったく互換性がないと思ってください。ソースコードから合成して MCS ファイルを作る場合、図 27 に書かれている SPI メモリから **s25fl256sxxxxxxx1-spi-x1_x2_x4** を選択してください。同様に Hardware manager でダウンロードする場合も図 28 に示す memory device から **s25fl256sxxxxxxx1-spi-x1_x2_x4** へ変更してください。

簡単なテスト方法

HUL RM、HUL Scaler、HUL MH-TDC の簡単なテスト方法を述べます。これらのファームウェアは VME クレートに挿さずに動かしてもとりあえず動きます。

まず、daq_func.cc で TRM::sel_trig に設定するレジスタを TRM::reg_L1Ext のみにします。その状態で NIMIN 1 へトリガー信号をつなぎます。とりあえずデータが返ってくる様子が見たい場合はこの状態で./bin/daq を実行すればデータが返ってくるはずです。

クレートにたくさん挿して使用する方法。

クレートに複数の HUL を挿す場合 J0 bus master が 1 台必要になります。逆にその他は正しく slave

である必要があります。**Master が 2 台あると J0 がショートします。**（KEK VME クレートを使わない場合はこの限りではありません。）この説明では L2 を利用するか？L1 や L2 はどこから入力するか？は考慮の外に置いています。J0 bus を利用して信号をやり取りする場合、フロントパネルから信号を配る場合に比べ信号クオリティは低くなると思われます。また、J0 を通る信号のタイミングは恐らく挿しているモジュールの数に依存します。なので、リコメンドは L1 だけフロントパネルから入れる、です。

J0 bus は M-LVDS ですが、本来 100 Ωの終端が 1 つ必要です。（何台挿していても 1 つで OK）現在 HUL には終端抵抗を取り付ける場所がありません。そのため、M-LVDS の信号クオリティはあまりよくないと思われます。万全を期す場合は終端を取るためだけの基板を作って J0 へ挿すことが望ましいです。

Master の設定方法

- HRM を Slot-U にマウントする。
- DIP SW1 を全て ON にする。（これが ON だと J0 bus に対してドライバモードになります。）
- DIP SW2 の 2 bit 目（Mezzanine HRM）と 4 bit 目（Bus BUSY）を ON にする。（当然ファームウェアが HRM をサポートしている必要があります。）
- TRM::sel_trig で EnRM を立てる。（EnJ0 は立てない）

Slave の設定方法

- DIP SW1 を全て **OFF** にする。
- TRM::sel_trig で EnJ0 を立てる。（EnRM は立てない）

この状態で Slave 側の Force busy (DIP SW2 3 bit 目)を ON して HRM の Busy LED が光れば正しくつながっています。

KEK VME クレート J0 から Level2 を受け取る場合、モジュールリセットで問題が発生する

KEK VME クレートに複数台さして J0 バスマスタと連動させて使う場合に発生するトラブルです。現状 K1.8 ビームラインでは L1, L2 両方を送信しており、L1 は NIM でフロントパネルから、L2 は J0 バスから入力しています。この時、BCT::Reset を使ってモジュールリセットを行うと、スレーブ側のモジュールが L2 を受け取ることが出来ず DAQ がハングアップするという問題が発生することがあります。発生するモジュール、タイミング等に再現性がなく、何度か BCT::Reset を行うと問題なく DAQ が走りだします。Master と Slave の間のリセットタイミングなどが原因か？と考えていますが、現状未解決問題です。

イベントスリップが発生した場合

J0 や HRM のタグを使ってイベントスリップを監視することが出来ます。これらがモジュール間で一致しない場合、確実にイベントが滑っています。HUL のファームウェアは multi-event buffer を持っている都合上、一度イベントが滑ると RUN の Start/Stop では解決しません。BCT::Reset を呼び出して内部のバッファをすべてクリアする必要があります。

HR-TDC の使い方。

HR-TDC は FPGA が 2 つあり、なおかつ初期化手順があるため使い方を間違えるとすぐにハングアップして動かなくなります。最初はここに書いた方法で動かしてみることをお勧めします。

＊必ず VME クレートに挿してテストしてください。

1. HUL へ Mezzanine HR-TDC を挿してそれぞれの FPGA に MCS をダウンロードします。電源投入後に Bit stream で入れると BCT がハングすることがあるので、MCS から始めることをお勧めします。**Mezzanine HR-TDC に MCS をダウンロードする際に指定する memory device は n25q128-1.8V-spi-x1_x2_x4 です。**
2. 一度クレート電源を On/Off します。
3. ソフトウェアの準備をします。Mezzanine HR-TDC が 2 枚とも刺さっている場合は配布した C++コードがそのまま使えます。もし片方がだけだけ刺さっている場合、RegisterMap.hh の en_up/un_down を false にしてください。また、debug_main.cc の CalibLUT のどちらかをコメントアウトしてください。
4. ./bin/debug を実行します。DDR の初期化、バージョンの読み出し、校正クロックを用いた Estimator 生成を行います。3 つとも成功していることを確認してください。
5. 配布した C++コードでは NIMIN 1 に Common stop を入れる（TRM::sel_trig が L1Ext のみ）になっています。NIMIN 1 へトリガーをつないで./bin/daq を実行します。これでデータが返ってくる、はずです。

6.0 今後の開発予定

プライオリティ順に開発予定を述べます。

~~**Mezzanine HR-TDC を開発して FPGA HR-TDC を実装する。**~~

~~今年度中くらいに動くレベルまで開発したい。~~

正式リリース。

SPI flash へ SiTCP 経由で MCS をダウンロードできるようにする

塩崎君の作ったモジュールと Ruby ソフトを移植する必要がある。誰でもできることなので誰かやってほしい。その場合 HUL Skelton から出発してほしい。

~~**LVDS 出力できる Mezzanine を開発する**~~

~~超簡単なのでこれも誰か作ってほしい。~~

正式リリース。

拡張 NIM Mezzanine card を開発する

これはそんなに簡単じゃないかも。要望があれば作ります。

放射線対策を行う

Configuration Memory Block の SEU 対策で SEM を導入する。

BRAM のデータ損傷に対応するため ECC を導入する。