Hadron Universal Logic module User Guide

2023.08.19

Ryotaro Honda

Open-It HUL project

Table of contents

1. Introduction

1.1 Update history

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。（4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HRMメザニンカードが刺さっている事を示すビットを追加。（4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。
- 4章にHR-TDCの項を追加。
- 7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

- HRM、 Scaler、 MH-TDC、 HRTDC_BASEファームウェアにおいて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をかけるとと Local busがハングするバグを解決。
- 7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。(解決済み)
- 7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。

2021.11.10

- Firmwareをversion 3へメジャーアップデート。
- User guideの記述を見直し大幅書き換え。

2022.01.13

- 英語版のユーザーガイド作成時に見つかった間違いを修正。

2022.03.18

- Mezzanine HR-TDCのsub-header構造の記述に間違いがあったので修正。

2023.01.17

- Mezzanine HR-TDC v5.0とHUL HRTDC BASE v4.0をリリース。変更した部分に関する記述を更新。
- ソフトウェアからMifFunc.ccを廃止。代わりにBctBusBridge.ccを導入。
- HUL HRTDC BASEのセルフビジー長が短くて、ビジー信号に谷間が出来るバグを修正。

2023.02.24

- HUL HRTDC BASE v4.1をリリース。v4.0はバグありで正しく動作しない。

2023.08.19

- Mezzanine HR-TDCのクロック周波数の記述が間違っていたのを修正。

1.2 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Xilinx Kintex 7に切り替え、より複雑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スロットを有しており、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 (32x2) の固定入力線、および64ペア (32x2) の差動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ペアの差動線はメザニンベースコネクタ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 (GbE) とトリガー入出力バス (KEK-VME J0) を有します。PCとの通信はSiTCPによって提供されるUDPとTCPのプロトコルを利用して行います。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通信はSiTCPのRBCP (remote-bus control protocol) によって拡張され、アドレス指定によるスローコントロールもサポートされます。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。 Open-It

2. Hardware

2.1 HUL controller module

Hadron Universal Logic (HUL) controller moduleはVME 6U規格の回路基板であり、HULと呼ぶ場合このモジュールを指します。購入の際には以下の管理番号で有限会社ジー・エヌ・ディーへ問い合わせてください。 **ジーエヌディー管理番号：GN-1573-1**

2.1.1 仕様詳細

- 入力ポート
  - 差動入力64ペア (KEL 8831Eコネクタ)
  - LVDS、ECL、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
- メザニンスロット
  - 2スロット
  - 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
- 通信インターフェース
  - RJ45: GbE (1000BASE-T)
  - VME J1は通信機能を有さない
- FPGA
  - Xilinx Kintex7 (XC7K160T-1FBG676C)
- コンフィギュレーションメモリ記録媒体
  - SPI flash 3.3V用 (製造時期に依存し以下3種のいずれか)
  - N25Q128A13EF840E
  - MT25QL512ABB1EW9-0SIT
  - S25FL256SAGNFI001
  - 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
- 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

HULの写真とブロック図を以下に示します。

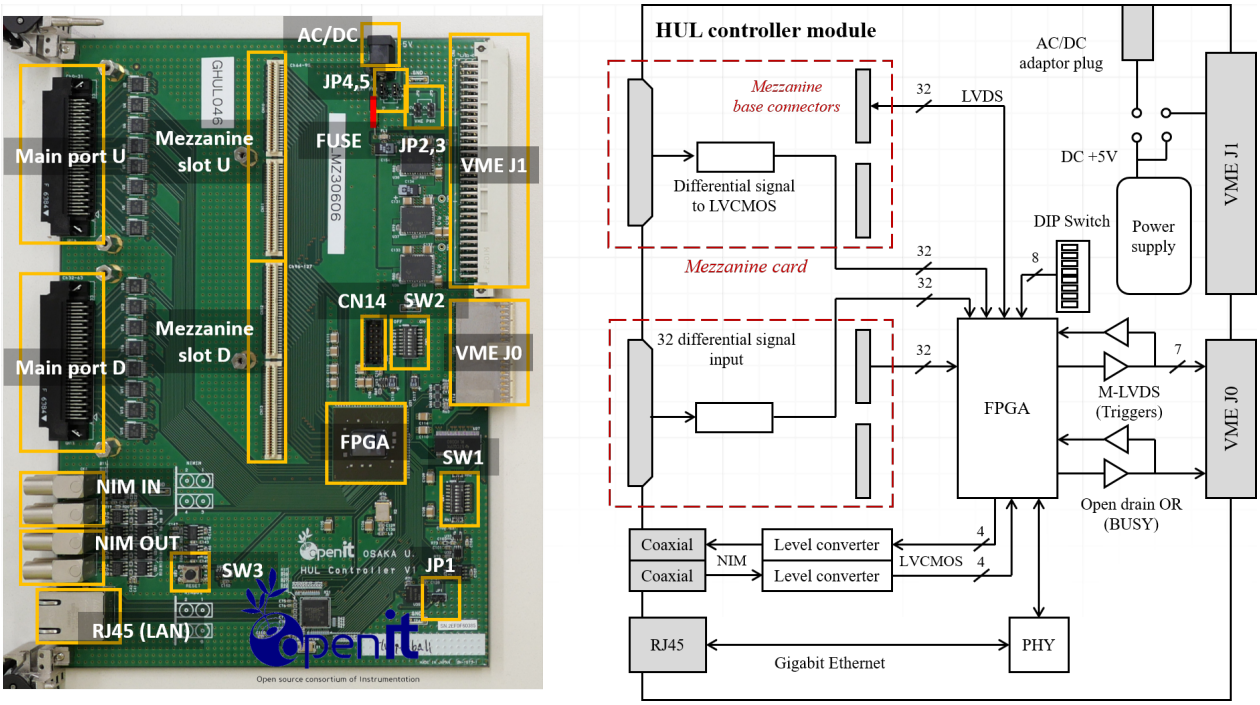

Figure 1: HUL controllerの写真、およびブロック図。

Main port U (D)

フロントの固定入力ポートです。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。チャンネルアサインはUおよびDに対してそれぞれ0-31、 32-63 chであり、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およびPECL信号規格を利用する場合、HUL側に電流制御用の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。最高繰り返しスピードは560 Mbpsです。それよりも高速な信号を入力する場合メザニンカードの使用を検討してください。ワイヤーチェンバーなどのASD出力を入力するような用途ではこの制限は問題になりません。

NIM IN

NIMレベルの4入力ポートです。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。**ポート番号は1から始まります**

NIM OUT

NIMレベルの4出力ポートです。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。**ポート番号は1から始まります**

LAN

RJ45のイーサネットコネクタです。Gigabit Ethernet PHYが接続されています。SiTCPを通じてFPGAとPCが通信するために利用します。

Mezzanine slot U(D)

上側のメザニンカードの取り付けスロットです。1つのスロットで2つのMOLEX-071439-0464コネクタを利用します。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と最も安定します。

信号線は32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。スロットUおよびDに入力ポートを割り当てた場合、メインポートから数えてチャンネルアサインはそれぞれ64-95および96-127 chになります。メザニンベースコネクタと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+3.3 Vです。HUL controller上の+3.3 V電源容量は6 Aであり、HUL上で1-2 A程度消費しているため、猶予は4-5 A (13-16 W) 程度です。

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。bitstreamやMCSのダウンロード方法については7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つドライバが存在するとショートします。

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に依存しますが著者 (本多) が開発したFWでは原則最も上位のリセット信号です。詳しくは後述します。押した際のレベル遷移は1→0です。(負論理)

JP1

利用しません。開放にしてください。

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。**必要ない場合は注文時にCN9を外すように指示してください。

2.2 HUL Mezzanine cards

2.2.1 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Amp Shaper Discriminator (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。機能的には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、繰り返しスピードの点で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。ただし微妙な差であるためサポート信号の種類が多いver.2が上位互換です。 **ジーエヌディー管理番号：GN-1573-S1**

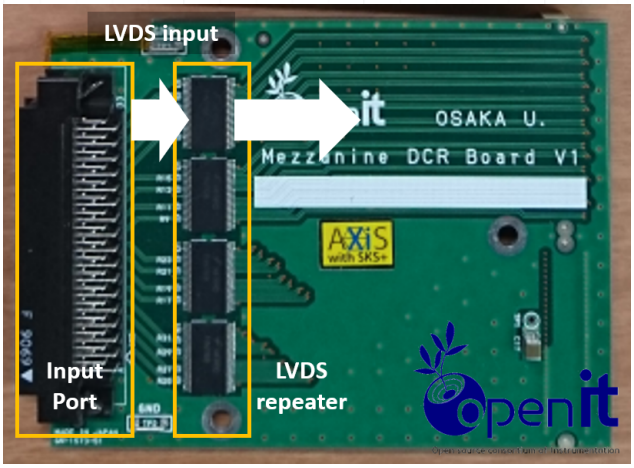

Figure 2: メザニンDCR v1

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、下図に示すようにDCR_NetAssign.vhdを通す必要があります。

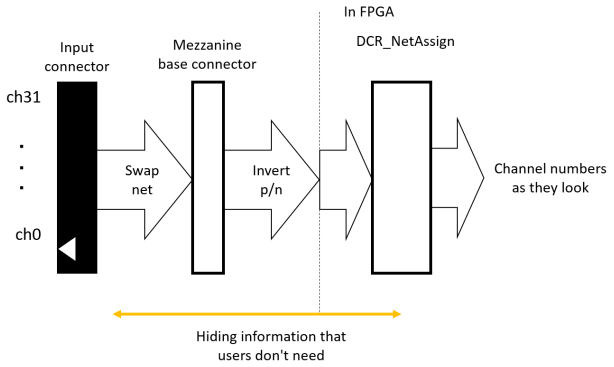

Figure 3: DCRのネットスワップの概念図

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保護のために一度信号を中継します。

2.2.2 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。HULの入力ポート拡張用途であれば、特別な理由がない限りver.2の方を購入してください。 **ジーエヌディー管理番号：GN-1626-1**

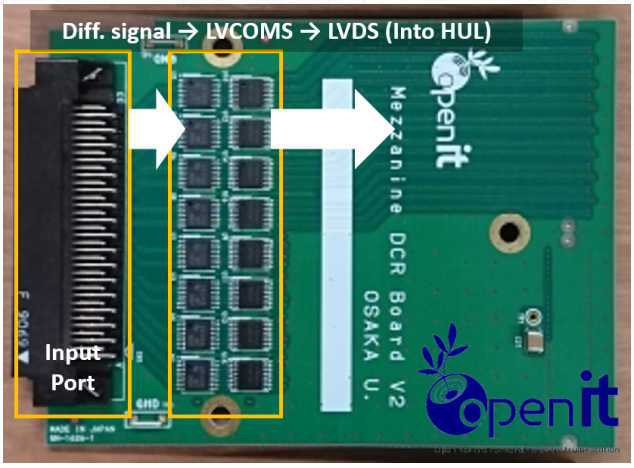

Figure 4: メザニンDCR v2

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。Main portで使われているICと同じものを用いて差動信号を一度LVCOMSへ変換します。その後LVDSへ再変換してFPGAへ入力します。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。

2.2.3 HUL Receiver Module (HRM)

Receiver moduleはJ-PARCハドロン実験等で利用されている Master Triger Module (MTM: GNN-570) 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、MTMへBUSY信号などを返送する回路基板になります。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MTM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へ送信する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**

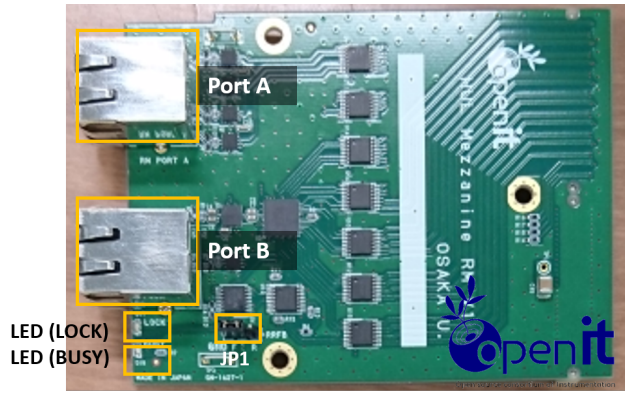

Figure 5: HUL receiver module (HRM)

Port A

トリガー入力ポートAです。MTM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。MTM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元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)

MTM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。HRMは直接クレートバスに繋がっていないため、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

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RCLKの立ち上がりに同期するか、立下りに同期するかを決めるジャンパです。デフォルトはRです。

Mezzanine HRMへの入出力信号

信号名	入出力方向	コメント
RCLK	IN	受信したイベントタグ情報から復元したクロックです。
LEVEL1	IN	Level1 trigger信号です。
LEVEL2	IN	Level2 trigger信号です。
Clear	IN	Fast clear信号です。
RSV1	IN	MTMから送られてくるReserve 1信号です。
LOCK	IN	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	IN	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	IN	MTMから送られてくる12ビットのイベント番号です。
Spill counter	IN	MTMから送られてくる8ビットのスピル番号です。
RRFB	OUT	タグ情報が出力されるタイミングを決めます。HighだとRCLKの立ち上り、lowだと立下りで出力されます。基板上のJP1の設定が優先されます。
BUSY	OUT	MTMへ送るBUSY信号です。
RSV2	OUT	MTMへ送るReserve 2信号です。

2.2.4 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と同一であり、LVDSリピータの入出力の向きが逆になった回路です。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**

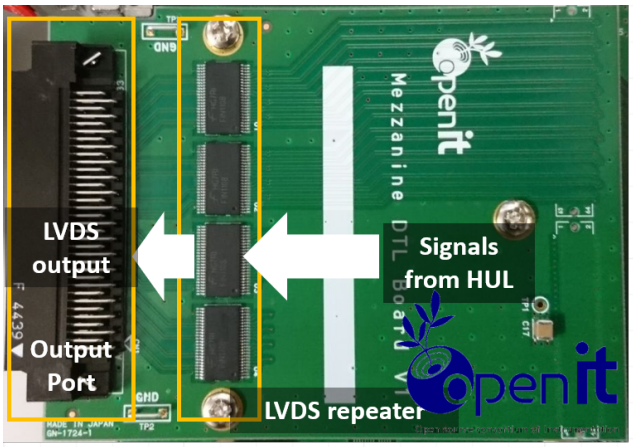

Figure 6: Mezzanine DTL

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DCR v1と同一です。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ります。

2.2.5 HUL Mezzanine NIM extension (NIM-Ex)

Mezzanine NIM-ExはHUL controllerに備わっているNIM入出力を拡張するための回路です。LEMOコネクタが合計12 ch備わっており、2 chずつ入出力の方向をスイッチで切り替えることが出来ます。HUL controllerのNIM IOと同様のICを採用しており、同等の速度の入出力が行えます。

NIM-Exあたりの消費電力は5.5Wと大きいです。 メザニンカード上で-3.3Vを生成しており基板自体が熱くなるため、空冷できない状況での使用は控えてください。熱により壊す可能性が十分にあります。

回路図に間違いが多くあります。 回路図ネットから正しいXDCとHDLコードを推測することが難しいため、参考用にskeletonプロジェクトにNIM-Ex用のexample designを同梱しました。

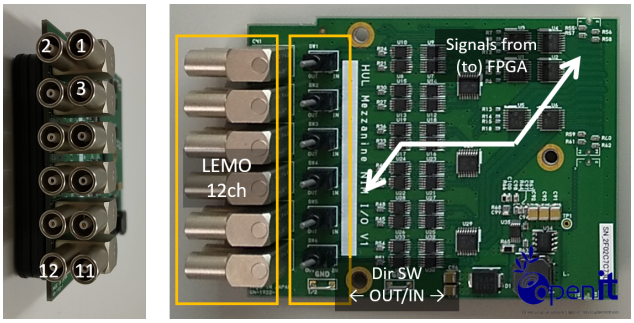

Figure 7: Mezzanine NIM-Ex

LEMO

LEMOコネクタ12 chです。Example designで提供されているNimEx_NetAssign.vhdにおけるネット番号とは上図のように対応します。

Dir SW

信号入出力方向を2 ch毎に設定するためのスイッチです。LEMOコネクタ側に倒すと出力、HUL側へ倒すと入力となります。

2.2.6 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。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へデータ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、TOF 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4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32 ch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はサポートしていません。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。メザニンあたりの消費電力は5W程度です。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。発熱が大きい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。（グランドが安定しない環境では性能が出ません。）
ジーエヌディー管理番号：GN-1644-1

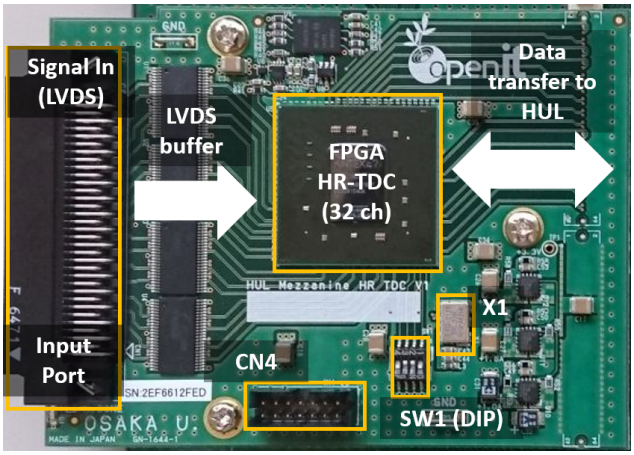

Figure 8: Mezzanine HR-TDC

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、もしくはMT25QU256ABA1EW9-0SITです。HUL本体のICとは異なります。

SW1

基板上のFPGAを制御するためのdip SWです。現状ではFPGAが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。

Flash memoryについて

Mezzanine HR-TDCで使用しているflash memoryには以下の2つのケースがあります。買った時期によって異なり、どのflash memoryが搭載されているかはVivadoを使わないと調べられないです。シールなどで見分けられるようにしてください。

- N25Q128A11EF840E（初期版）
- MT25QU256ABA1EW9-0SIT（2020年度製造版より）

3. 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です。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
- BBTのweb site

SiTCP動作の概略は下図のようになります。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、現状のHULのファームウェアでは1 Gbps固定です。TCP通信はシステムクロックに同期した8-bit送信と8-bit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、データ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32-bitアドレス送付、8-bit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。

SiTCPはデフォルトではkeep alive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では（1 trigger/30min程度）keep alive packetの送信を行うように設定しないとコネク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。

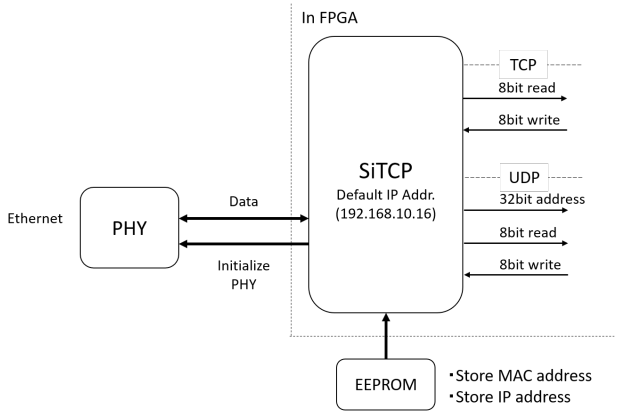

Figure 1: SiTCP関連回路のブロック図

3.1 IP addressとMACの設定方法

IP addressとMACは基板上のEEPROM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FWがFPGAにダウンロードされている必要があります。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の用意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。ツールを立ち上げたらジー・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、ジーエヌディーへ任せてください）MACは機器固有の物理アドレスのため、衝突させると動かなくなります。MPC書き込みツールについてはBBTのサイトを参照してください。

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。下図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。正しく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。

Figure 2: SiTCP Utilityの操作画面

4. Firmware

4.1 Version.1系とVersion.3系の違い

HULのリリース当時から2021年7月まで保守を続けてきたファームウェア（version.1）とそれ以降のファームウェア（version.3）ではBCTの仕様変更に伴いソフトウェアに互換性がありません。 Version.2系はver.3系へアップデートする際の内部開発系統なので公開していません。 Version.1ではBCTでRBCP addressの一部をデータ線として使っていましたが、version.3ではRBCP address 32 bitとRBCP data 8 bitをそのまま利用するようにしています。ソフトウェアでもModuleIDとLocalAddressによる表記をやめ、RBCP addressでローカルモジュールを指定するように変更しました。 ModuleIDやLocalAddressはlocal bus controllerの構造を知らないユーザーにとっては不必要な情報であるためです。

Version.3系では全てのファームウェアの共通機能として2つのローカルモジュールが追加されました。SiTCP経由でSPI flash memoryを書き換えるFlash Memory Programmer (FMP)、およびFPGAや基板の状態を自己診断するSelf Diagnosis System (SDS)です。FMPのおかげでUSBブラスターケーブルを使って書き込んでいたMCSファイルをネットワーク経由でダウンロードすることが出来るようになりました。これにより遠隔でのファームウェア更新が可能となりました。SDSは放射線によるソフトエラー修正回数の監視、および修復不可能な状態に落ちった事の検出が可能です。また、FPGAの温度、およびVCCINT、VCCAUX、VCCBRAM電位の監視が可能です。

4.2 ファームウェアの基本構造

HULのファームウェアには大別するとデータ収集を担うTDCのようなタイプと、トリガー回路のようなデータをPCへ転送しないタイプが存在します。 後者は機能によってブロック構成が異なりますが、データ収集タイプのファームウェアは統一されたブロック構造を持ちます。 以下にデータ収集タイプのブロック図を示します。 図中オレンジで示した構成要素はどのファームウェアにも実装されている機能です。 各ローカルモジュールを制御するlocal busはlocal bus controller (BCT)によって制御されており、BCTはSiTCPのRBCP (UDP)と接続されています。 FMPとSDSは全てのversion.3系のファームウェアに実装されているローカルモジュールであり、機能は前述の通りです。

データ収集に関わる部分は図中の白い四角で表されています。 TDCやスケーラといった計測に関わるブロックは計測ブロックと呼ばれます。 それらの機能は主に*user circuit*と書かれたブロックに実装されます。 これら*user circuit*からのデータはbuilder busを介してevent builderブロックに集められ、イベントヘッダを付与した後SiTCPを通じてPCへ転送されます。 Receiver block、trigger manager、およびJ0 interfaceはトリガー入出力に関わるブロックです。 HULはMTM (GNN-570)から送信されてくるトリガーとイベントタグ情報を受け取るように設計されています。 Receiver blockはHRMメザニンカードが実装されている場合有効になるブロックであり、トリガーを上流の回路から受け取る役割を持ちます。 このブロックはトリガー受信と同時に受信したトリガー情報をPCへ転送するためにbuilder busにも接続されています。 Receiver blockを利用するHULはVMEクレート内のJ0バスマスタであると思われます。 J0 interfaceはHULがJ0バスマスタである場合、receiver blockが受け取ったトリガーとタグ情報をJ0バスへ流します。 一方J0バスに対してスレイブである場合、J0バスからやってきたトリガー情報をtrigger managerへ渡します。 また、HULは汎用モジュールであるため、MTMが存在しない状況でも利用できるようにNIMポートからのトリガー入力をサポートしています。 Trigger managerはファームウェア内部のトリガー配布を担っており、HRMメザニン (receiver block)、NIM入力、J0バス入力からトリガー入力経路を選択できます。 Trigger managerが送信する*LEVEL2*、*Fast clear*、*TAG*の情報はevent builder内部でデータ送信の決定やイベントヘッダ情報の生成に利用されます。

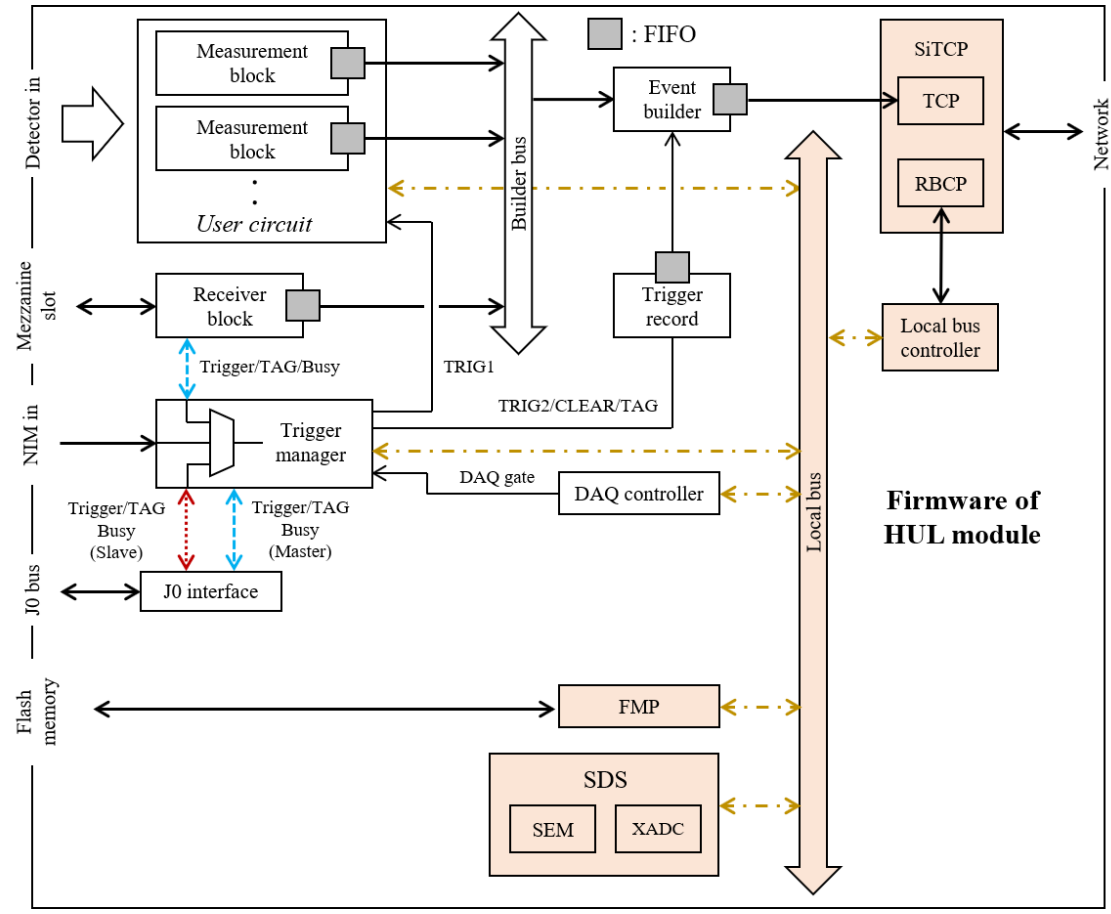

Figure 1: データ収集タイプファームウェアのブロック図

4.3 各ファームウェア共通の事柄

hul_softwareは共通部分であるBCT、FMP、およびSDSのアドレスを`common/src/RegisterMapCommon.hh`で管理しています。

4.3.1 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。

リセット方法（下に行くほど影響範囲が広いです）

- BCTリセット：BCTのResetにRBCPで書き込みを行う。BCT管理下にある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.cc`の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SW3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。

4.3.2 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利用不可にしています。 100 Mbpsで利用する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。 100 Mbpsでしか通信できないネットワークスイッチにつなぐとSiTCPが動かなくなります。

4.3.3 Bus controller

BCTにはローカルモジュールにアクセスする機能のほかに、BCTリセットを発行する、バージョンを取得する、FPGAの再コンフィグを行う、という機能が実装されています。 これらの機能はRBCP addressで以下のアドレスを実行することで利用可能です。 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。 以下の表においてビット幅が `-` と表記されているレジスタについては、送信レジスタ値の内容を気にしません。 書き込み動作を行うことで効力が発揮されます。

RBCP address of BCT (Module ID: 0xE)

レジスタ名	アドレス	読み書き	ビット幅	備考
Reset	0xE000'0000	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	0xE010'0000	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	0xE020'0000	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。

4.3.4 Flash Memory Programmer

FMPはPCから指定したSPIコマンドを送信することでメモリがサポートしているコマンドの実行、ページデータの読み書きが可能です。 メモリの消去、書き込み、読み出しをFMPを通じて行う事が可能であり、Vivadoを通じてメモリへ書き込む動作 (消去、書き込み、ベリファイ) と同じことがネットワークを通じて実行可能です。

FMPには既知のバグがあります。ページ書き込みをする際に書き込みコマンドを実行してからそのコマンドが終了する前に次のページ書き込みを要求できてしまいます。 現状はファームウェア側で対処せずにソフトウェア側で十分な時間待って次のページ書き込みコマンドを送信するようにしていますが、あまり良い対処方法ではありません。 そのうちファームウェア側で根本対処するかもしれません。 このような対処を行っているため、ネットワークで書き込みを行っている割には時間がかかります。 書き込みに失敗する事象が頻発する場合、著者へ連絡をしてください。

以下にRBCPアドレスを記載しますが、 `hul_software` に同梱されている `FlashMemoryProgrammer.cc` 以外からFMPを制御するのはお勧めしません。 間違って書き込みロックビットを立ててしまうと、そのメモリには二度と書き込みが出来なくなります。

RBCP address of FMP (Module ID: 0xD)

レジスタ名	アドレス	読み書き	ビット幅	備考
Status	0xD000'0000	R	8	FMPモジュールのステータスビットを取得します。 現状下位1ビットにSPIコマンドサイクルbusyが割り当てられています。
Status	0xD000'0000	R/W	8	SPIシーケンスのモード切替を行います。 1~2-bit目: SPIシーケンスモード • 0x0: Read mode • 0x1: Write mode • 0x2: Instruction mode 3-bit目: Dummy mode Chip selectをOFFにし、SPIシーケンスを回してもフラッシュメモリが反応しないようにします。
Register	0xD020'0000	R/W	64	SPIコマンドのビット列を与えます。
InstLength	0xD030'0000	R/W	3	SPIコマンドのビット長を与えます。
ReadLength	0xD040'0000	R/W	10	ページ読み出し長を与えます。
WriteLength	0xD050'0000	R/W	10	ページ書き込み長を与えます。
ReadCountFIFO	0xD060'0000	R	10	ページ読み出ししたデータが格納されているFIFOのread countを取得します。
ReadFIFO	0xD070'0000	R	8	8-bitずつページ読み出しFIFOからデータ読み出します。
WriteCountFIFO	0xD080'0000	R	10	ページ書き込み用のデータを格納するFIFOのwrite countを取得します。
WriteFIFO	0xD090'0000	W	8	8-bitずつページ書き込み用FIFOへデータを書き込みます。
Execute	0xD100'0000	W	-	SPIコマンドシーケンスを実行します。

4.3.5 Self Diagnosis System

SDSは自己診断プログラムです。 Xilinx FPGAのIPであるSoft Error Mitigation (SEM)とXADCをそれぞれ実体化し監視しています。 SEMはsingle event upset (SEU)を検知、訂正、分類するIPです。 SDSでは訂正可能なエラーを訂正した回数の取得と、訂正不可能なエラーが発生したことの検知が行えます。 訂正不可能な状態に陥った場合フラッシュメモリからFPGAを再コンフィグするか、パワーサイクルを行う必要があります。 また、意図的にSEUをインジェクトすることが可能ですが、SDSの高度な利用になるのでSEMの利用方法をXilinxのUGで調べて使用してください。

XADCはXilinx FPGAのビルトインADCです。 HULではXADCを利用してFPGAの温度、およびVCCINT、VCCAUX、VCCBRAM電位を取得しています。

SEMには未解決のトラブルがあります。 FPGAの個体によっては電源投入後にuncorrectable errorステータスが1になってしまいます。 原因は分かっていませんが一度SEMをリセットすることで正常に使う事が出来ます。 そのような個体に遭遇したら電源投入後に`hul_software`のreset_semを実行してください。

RBCP address of SDS (Module ID: 0xC)

レジスタ名	アドレス	読み書き	ビット幅	備考
SdsStatus	0xC000'0000	R	8	SDSモジュールのステータスを取得します。
XadcDrpMode	0xC010'0000	R/W	1	XADCのDRPモード選択をします。 • 0x0: Read mode • 0x1: Write mode
XadcDrpAddr	0xC020'0000	R/W	7	XADCのDRP addressを与えます。
XadcDrpDin	0xC030'0000	R/W	16	XADCのDRPのデータ入力を与えます。
XadcDrpDout	0xC040'0000	R	16	XADCのDRPのデータ出力を取得します。
XadcExecute	0xC0F0'0000	W	-	XADCのDRPアクセスを実行します。
SemCorCount	0xC100'0000	R	16	訂正可能なSEUをSEMが訂正した回数を取得します。
SemRstCorCount	0xC200'0000	W	-	SemCorCountをリセットします。
SemErroAddr	0xC300'0000	W	40	SEMのinject_addressポートに入力するアドレスを与えます。
SemErroStrobe	0xC400'0000	W	-	SEMのinject_strobeポートにパルスを入力します。

SdsStatusの内容

ビット番号	ステータス名	備考
1	Over temp	FPGA温度が125℃を超えたことを示します。重大な冷却不足なので直ぐにHULの電源を切り冷やしてください。
2	Temp alarm	FPGA温度が85℃を超えたことを示します。冷却能力が不足していることが考えられます。
3	VCCINT alarm	VCCINTの電圧が正常範囲（0.97-1.03V）を超えたことを示しています。何らかのトラブルが基板に起きています。
4	未使用	
5	Watchdog alarm	SEMのheartbeat信号が出ていない事を示します。何らかのトラブルがSEMに起きています。
6	Uncorrectable error	SEMのが訂正不可能な放射線エラーを検出したことを示します。FPGAの再コンフィグが必要です。
7	未使用	
8	未使用	

4.4 HUL Skeleton

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（32OR）して、それぞれ4つのNIMOUTから出力します。

Skeletonの綴りが間違っているのですが直すと影響範囲が広いのでこのままにします。

NIM-Exメザニンカードを利用する際のイグザンプルが `sources_1/example_design/` 以下に同梱してあります。 同梱されたtoplevel.vhdを参考にしてください。

Firmware固有名と現在の最新版

現在のHUL Skeletonの固有名とバージョンは以下のようになります。

固有名	0x0000
メジャーバージョン	0x03
マイナーバージョン	0x02

以下バージョンはv3.2のような形式で表記します。

更新歴

バージョン	リリース日	変更点
-	-	v2.xまでの履歴は無し
v3.2	2021.08.01	Version.3系へ変更。

4.4.1 レジスタマップ

RBCP address of LED (Module ID: 0x0)

LED点灯のパターン
LED1-3はLEDレジスタかもしくはSW2の2-4ビット目で点灯させることが出来ます。 LED4番目はSDSのover temperatureフラグが接続されています。

4.5 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。MH-TDCやscalerファームウェアの元となる構成をしており、新しくDAQタイプのファームウェアを開発する際の出発点として利用してください。

Trigger入力に対しる応答はこのファームウェアが基本となっているため、ここではtrigger系統とそれに対するevent builderの応答について詳しく説明します。

Firmware固有名と現在の最新版

固有名	0x0415
メジャーバージョン	0x03
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	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	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	2018.02.02	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。
v2.x	-	未公開
v3.2	2021.08.01	FMPとSDSを追加。Builder busの導入。Local busの構造変更。

モジュール動作概要

HUL RMのデータ収集に関わるブロック図を以下に示します。この図ではデータ収集に関わるモジュール以外は省略しています。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 slot Uにのみ機能が実装されています。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) であり、lock bit、spill number increment、spill number (full 8bit)、event number (full 12bit) の情報を記憶し、event builder (EVB) に渡します。Event builderへはbuilder busを介してデータが渡されるため、RVMの情報はデータボディに含まれます。すなわち、RVMはEVBに取って計測ブロックの一部です。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3-bitへspill numberは1-bitへ減らされます。3つ目はTRMへの入力になります。

HULのファームウェアには内部のトリガー配布を管理する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。

TRMはlevel1 trigger, level2 trigger, clear, およびevent number (3-bit)、spill number (1-bit)の情報を各計測ブロックとEVBへ配布します。EVBにとってはこれらはDAQデータではないので、event hearder内にTRMからのタグは埋め込まれます。HULがJ0 busに対してマスターである場合とスレーブである場合とで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event numberとspill numberは3-bitと1-bitに減らされています。Tagを受け取っていない場合event numberもspill numberも0になります。

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。

HULには3種類のbusy信号が存在します。 1つ目はmodule busyであり、これはファームウェア内部のブロックが出力するbusy信号の論理和です。 2つ目はj0 bus busy信号であり、これはJ0 busに対してスレーブなHULのmodule busy信号の論理和です。通常J0 busに対してマスターであるHULが取り扱います。 最後はcrarte busy信号であり、これはmodule busy、J0 bus busy、およびNIM-INポートから入力される外部busy信号の論理和です。これも通常J0 busに対してマスターであるHULが取り扱います。

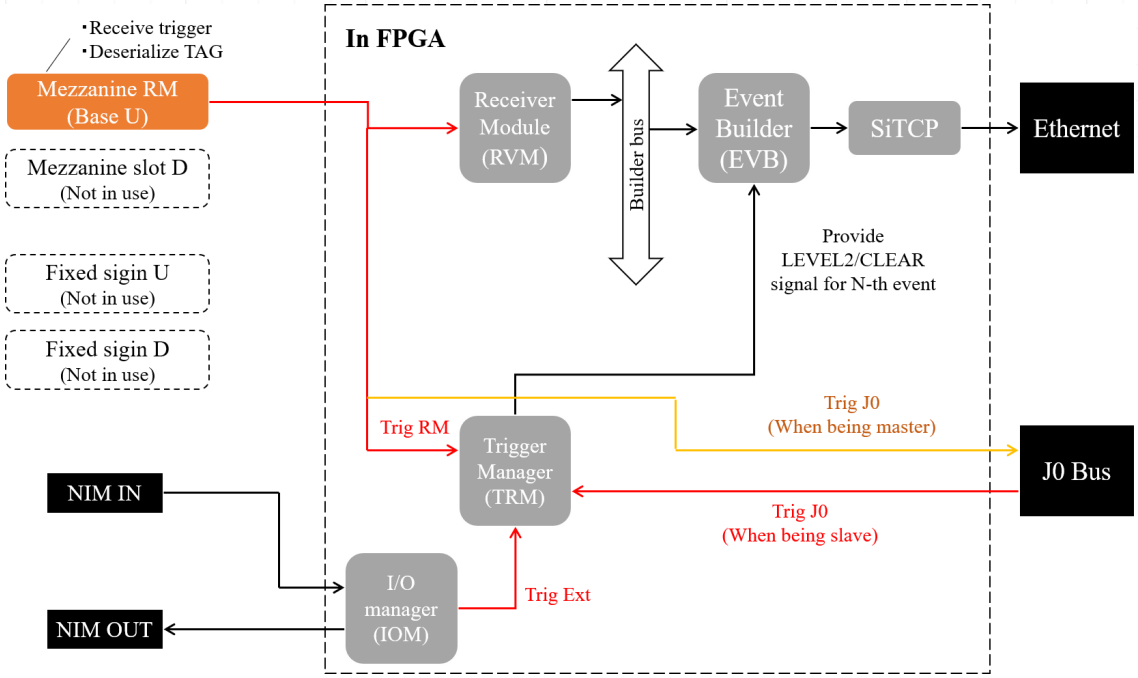

Figure 2: HUL RMファームウェアのブロック図

4.5.1 レジスタマップ

以下は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ファームウェアに存在しても同一の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。

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: TRM (module ID = 0x00)				
SelectTrigger	0x0000'0000	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 = 0x01)				
DaqGate	0x1000'0000	R/W	1	DAQ gateのON/OFF。DAQ gateが0だとTRMはtriggerを出力できない。
EvbReset	0x1010'0000	W	-	このアドレスへ書き込み要求することでEVB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。
IO Manager: IOM (module ID = 0x02)				
NimOut1	0x2000'0000	R/W	4	NIMOUT1へ何を出力するかを設定する。
NimOut2	0x2010'0000	R/W	4	NIMOUT2へ何を出力するかを設定する。
NimOut3	0x2020'0000	R/W	4	NIMOUT3へ何を出力するかを設定する。
NimOut4	0x2030'0000	R/W	4	NIMOUT4へ何を出力するかを設定する。
ExtL1	0x2040'0000	R/W	3	extL1にどのNIMINを接続するか設定。
ExtL2	0x2050'0000	R/W	3	extL2にどのNIMINを接続するか設定。
ExtClr	0x2060'0000	R/W	3	Ext clearにどのNIMINを接続するか設定。
ExtBusy	0x2070'0000	R/W	3	Ext busy入力にどのNIMINを接続するか設定。
ExtRsv2	0x2080'0000	R/W	3	Ext rsv2入力にどのNIMINを接続するか設定。

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12-bitのレジスタ、SelectTriggerで設定します。トリガー信号の経路とSelectTriggerの関係を図にまとめます。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で入ってきてしまいます。L1の経路が決まると次にDAQ gate (DAQ controller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にはL1のコピーが送信され、Clearは常に0です。MTM-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が出力されません。

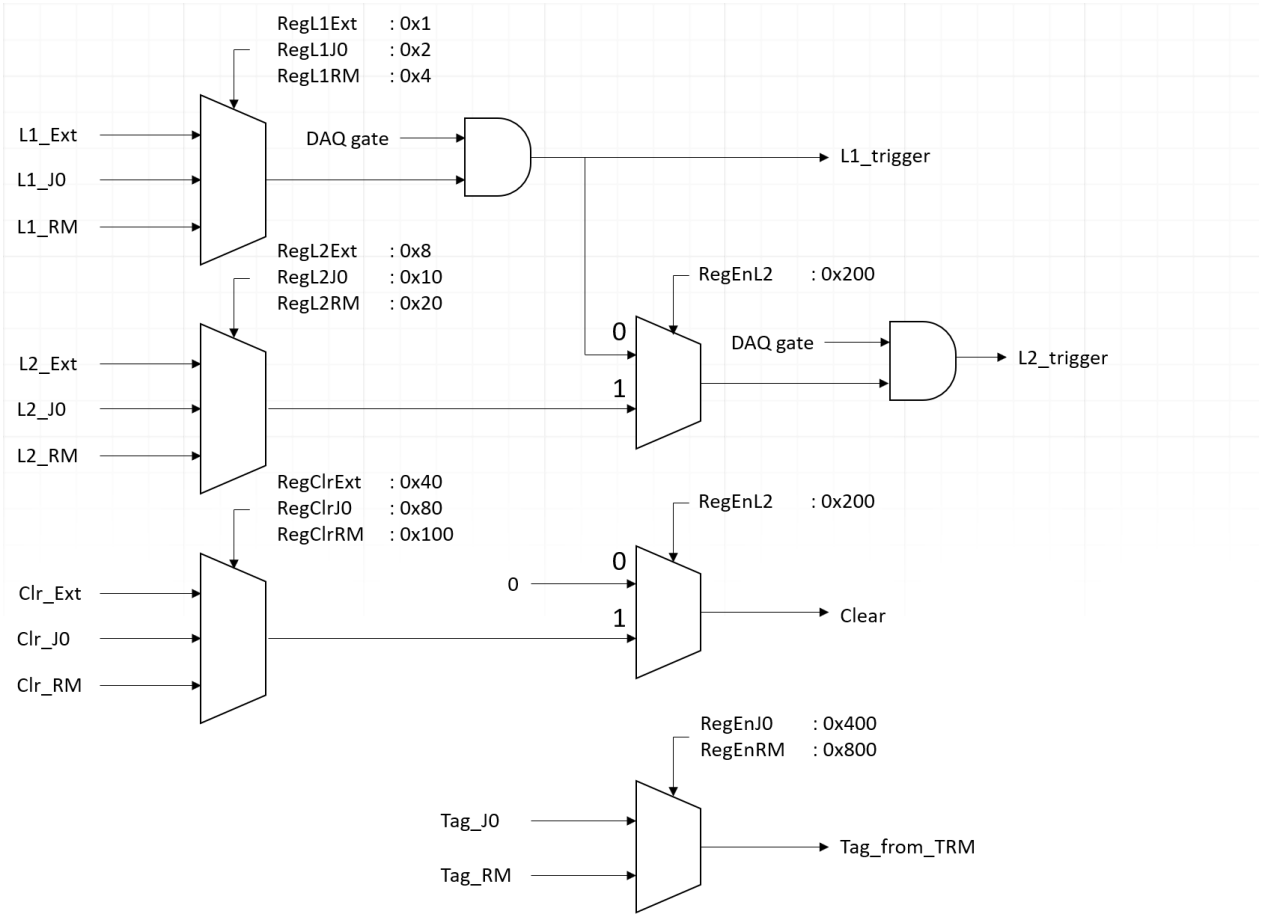

Figure 3: TRM内部のトリガー経路図。

`TRM::SelectTrigger`アドレスに対するレジスタ値のリストです。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12-bit幅のビット列になります。例外的にRegEnJ0のみTRMの外でも使われます。RegEnJ0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。クレートにモジュールを挿入するがJ0 busへは影響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
RegL1Ext	0x1	NIMINからL1 triggerを選択。
RegL1J0	0x2	J0 busからのL1 triggerを選択。
RegL1RM	0x4	Mezzanine HRMからのL1 triggerを選択。
RegL2Ext	0x8	NIMINからのL2 triggerを選択。
RegL2J0	0x10	J0 busからのL2 triggerを選択。
RegL2RM	0x20	Mezzanine HRMからのL2 triggerを選択。
RegClrExt	0x40	NIMINからのClearを選択。
RegClrJ0	0x80	J0 busからのClearを選択。
RegClrRM	0x100	Mezzanine HRMからのClearを選択。
RegEnL2	0x200	0: L2=L1 trigger、1: L2=L2入力
RegEnJ0	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。
RegEnRM	0x800	Tag情報にHRMから受信した値を採用する。

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AddrNnimout1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module busy信号が出力されます。AddrExtL1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、逆の関係になっています。これらのレジスタ値はビット列ではなく整数値として解釈します。

レジスタラベル	レジスタ値	備考
		NIMOUTへ出力可能な内部信号
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	DAQ gateを出力します。
Reg_o_DIP8	0x8	DIP SW2の8-bit目で設定した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
ExtLClear	Reg_i_default	0
ExtLBusy	Reg_i_nimin3	NIMIN3
ExtLRsv2	Reg_i_nimin4	NIMIN4

4.5.2 HUL上のスイッチ・LEDの機能

DIP SW2の機能

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	Not in Use	
7	Not in Use	
8	Level	IOMのDIP8から出力されるレベルです。

Mezzanine HRM (DIP SW2) の機能

DIP SW2の2-bit目が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。

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。

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*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。

*J0 bus*スレーブモードを*OFF*にする
このビットがOFF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。

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)

LED点灯の機能

LED番号	備考
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。

4.5.3 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、イベントバッファが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データリンクスピードが許す限りビルドと転送を続けます。

HUL RMの場合計測ブロックはRVMとTRMのみなります。TRMの情報はデータボディには含まれ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。イベントビルトされる情報はRVMデータのみであり、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のどちらが送信されたかを保存しており、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、clear busyという物も存在しません。L1 riggerによって発生したイベントは必ずビルドされます。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。

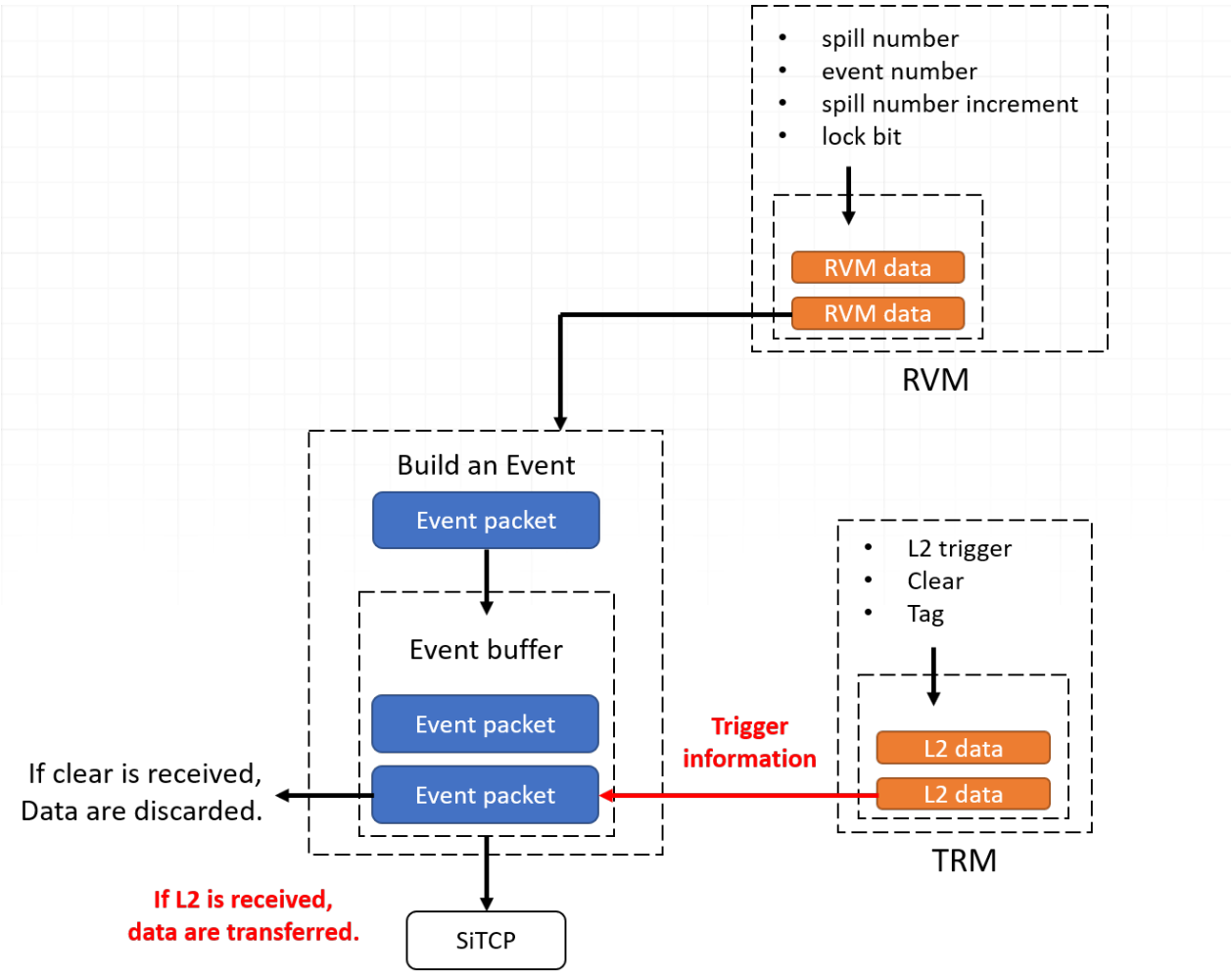

Figure 4: HUL RM firmwareのDAQデータフロー

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SiTCP fullはネットワーク転送が追いつかなくなる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種別	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転送が追いつかない事を意味するので、実質的にSiTCP fullと同等です。
SiTCP full	-	SiTCPのTCPバッファがfullになると出力されます。ネットワークスピードに対して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

HULでは32-bit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。

Header word

```
Header1 (Magic word)
MSB                                                                         LSB
|                                        0xFFFF0415                          |

Header2 (event size)
|       0xFF       |      0x00       |    "00000"    |         Number of word (11-bit)         |
```

"Number of word" はデータボディに何ワード含まれるかを示します。ヘッダの3ワードは含みません。

```
Header3 (event number)
|       0xFF       | HRM exist | "000" | Tag (4-bit) |         Self counter (16-bit)           |
```

*HRM exist*が1の場合HRMメザニンカードが取り付けられていることを示します。すなわち、データボディにRVMのデータワードが存在します。 *Tag*はTRMから受信した4-bitのタグ情報です。下位3ビットがMTMが送信したイベントナンバーの下位3ビットに相当し、4ビット目がスピルナンバーの最下位ビットです。 *Self counter*はイベントが送信されるたびにインクリメントされるローカルイベント番号です。0から始まります。

Data body

```
RVM word
|  0xF9   | "00"  | Lock  | SNI  | Spill Num (8-bit) |           Event Num (12-bit)           |
```

*Lock*はHRMのlock bitです。1でないといけません。*SNI*はspill number incrementであり、1であればスピル番号が変わったことを示しますが、そのように動作しているかどうかはチェックしていません。 *Spill Num*と*Event Num*はそれぞれHRMが受信したスピルナンバーとイベントナンバーです。

4.6 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、28-bit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

固有名	0x4ca1
メジャーバージョン	0x03
マイナーバージョン	0x03

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	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	2018.02.02	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。
v2.x	-	未公開
v3.3	2021.08.01	FMPとSDSを追加。Builder busの導入。Local busの構造変更。

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 slot Dと固定入力ポートに機能が割り当てられています。Mezzanine slotにはDCR v1 (v2)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 slot 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slot 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、28-bit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。

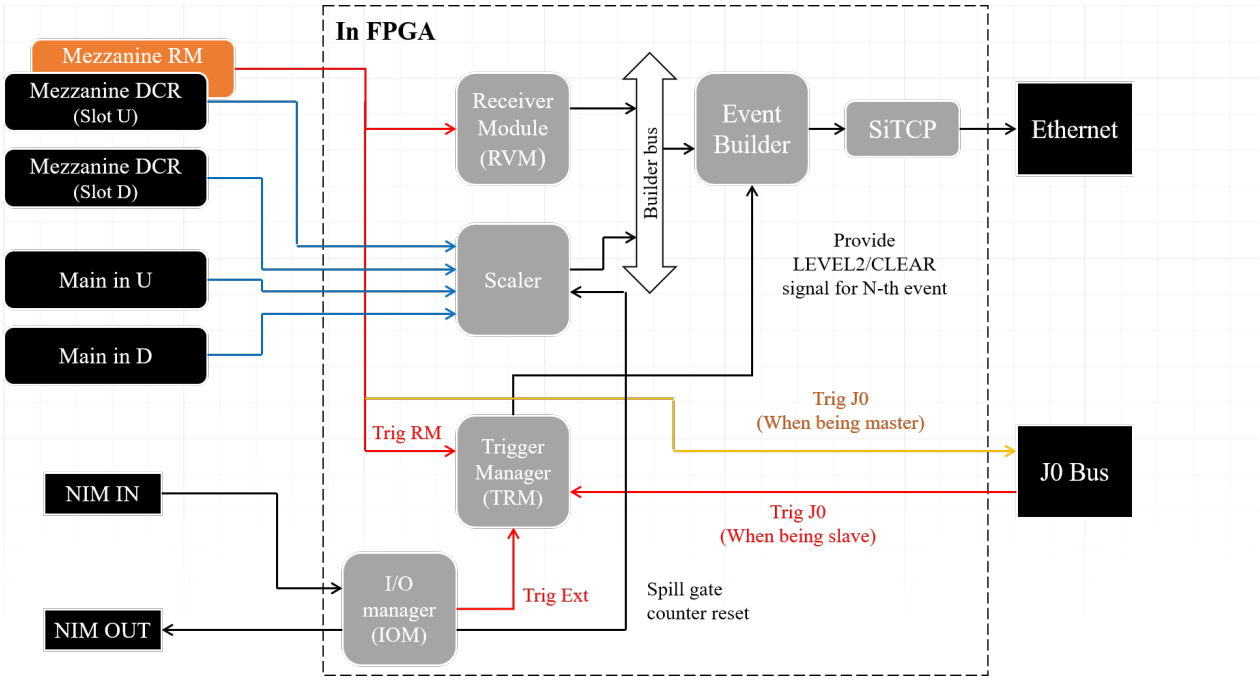

Figure 5: HUL Scaler FWの構造。

4.6.1 レジスタマップ

以下のテーブルはHUL Scaler専用のレジスタです。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。 HUR RMの異なる部分は赤字で示してあります。

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: TRM (module ID = 0x00)				
SelectTrigger	0x0000'0000	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 = 0x01)				
DaqGate	0x1000'0000	R/W	1	DAQ gateのON/OFF。DAQ gateが0だとTRMはtriggerを出力できない。
EvbReset	0x1010'0000	W	-	このアドレスへ書き込み要求することでEVB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。
IO Manager: SCR (module ID = 0x02)				
CounterReset	0x2000'0000	W	-	Software counter resetをアサート。
EnableBlock	0x2010'0000	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：Main in U 2bit目：Main in D 3bit目：Mezzanine U 4bit目：Mezzanine D
EnableHdrst	0x2020'0000	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：Main in U 2bit目：Main in D 3bit目：Mezzanine U 3bit目：Mezzanine U
IO Manager: IOM (module ID = 0x03)				
NimOut1	0x3000'0000	R/W	4	NIMOUT1へ何を出力するかを設定する。
NimOut2	0x3010'0000	R/W	4	NIMOUT2へ何を出力するかを設定する。
NimOut3	0x3020'0000	R/W	4	NIMOUT3へ何を出力するかを設定する。
NimOut4	0x3030'0000	R/W	4	NIMOUT4へ何を出力するかを設定する。
ExtL1	0x3040'0000	R/W	3	extL1にどのNIMINを接続するか設定。
ExtL2	0x3050'0000	R/W	3	extL2にどのNIMINを接続するか設定。
ExtClr	0x3060'0000	R/W	3	Ext clearにどのNIMINを接続するか設定。
ExtSpillGate	0x3070'0000	R/W	3	内部信号線のext spill gateにどのNIMINを接続するか設定。
ExtCCRst	0x3080'0000	R/W	3	内部信号線のext counter resetにどのNIMINを接続するのか設定。
ExtBusy	0x3090'0000	R/W	3	Ext busy入力にどのNIMINを接続するか設定。
ExtRsv2	0x30A0'0000	R/W	3	Ext rsv2入力にどのNIMINを接続するか設定。

4.6.2 各ブロックの機能

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28-bit長で一周すると0に戻ります。スケーラは32chごとに4つのブロックに分けられており、EnableBlock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ExtCounterReset(NIMIN、IOMで制御)、もしくはソフトリセットのCounterReset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EnableHdrst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ExtSpillGate（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ます。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。

レジスタラベル	レジスタ値	備考
		NIMOUTへ出力可能な内部信号
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	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。

以下はIOMレジスタの初期値のテーブルです。

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
ExtLClear	Reg_i_default	0
ExtSpillGate	Reg_i_default	1
ExtCounterReset	Reg_i_nimin2	NIMIN2
ExtLBusy	Reg_i_nimin3	NIMIN3
ExtLRsv2	Reg_i_nimin4	NIMIN4

4.6.3 HUL上のスイッチ・LEDの機能

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。

4.6.4 DAQの動作

データフローを図に示します。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イベントビルドします。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。

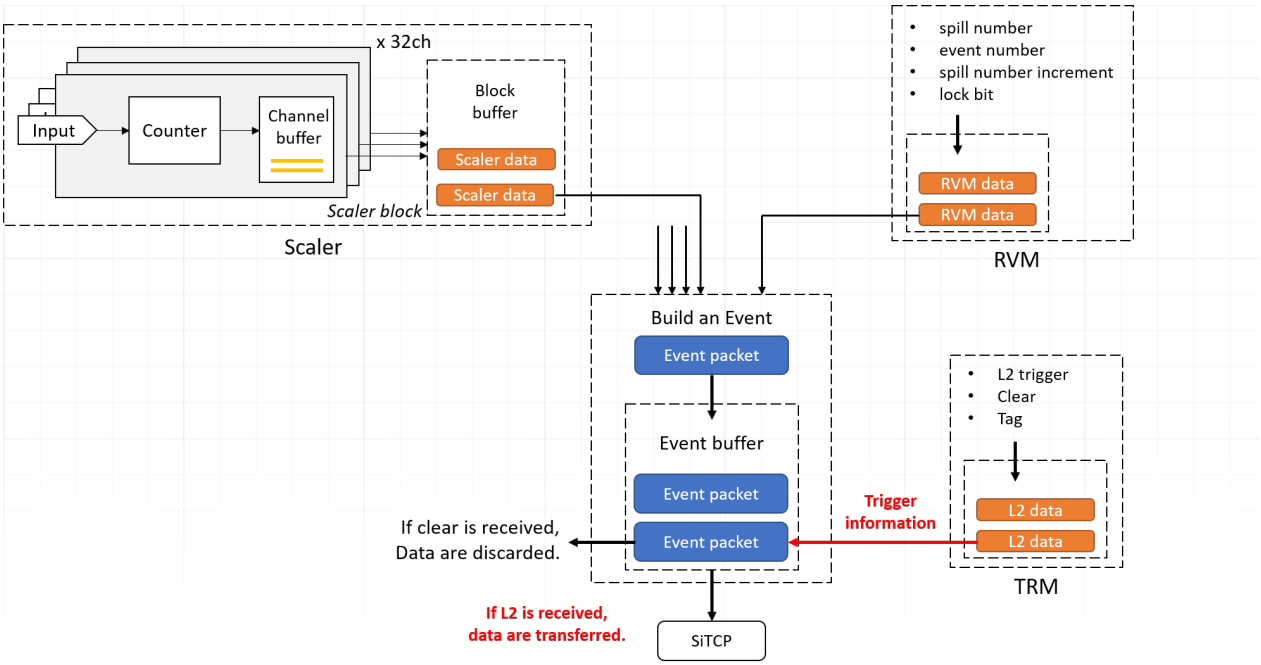

Figure 6: HUL Scaler FWのDAQデータパス。

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SiTCP fullはネットワーク転送が追いつかなくなる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種別	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	-	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転送が追いつかない事を意味するので、実質的にSiTCP fullと同等です。
SiTCP 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

Header word

```
Header1 (Magic word)
MSB                                                                   LSB
|                                    0xFFFF4CA1                         |

Header2 (event size)
|       0xFF       |      0x00       |    "00000"    |        Number of word (11-bit)        |
```

*Number of word*はデータボディに何ワード含まれるかを示します。ヘッダの3ワードは含みません。

```
Header3 (event number)
|       0xFF       | HRM exist | "000" | Tag (4-bit) |        Self counter (16-bit)          |
```

*HRM exist*が1の場合HRMメザニンカードが取り付けられていることを示します。すなわち、データボディにRVMのデータワードが存在します。 *Tag*はTRMから受信した4-bitのタグ情報です。下位3ビットがMTMが送信したイベントナンバーの下位3ビットに相当し、4ビット目がスピルナンバーの最下位ビットです。 *Self counter*はイベントが送信されるたびにインクリメントされるローカルイベント番号です。0から始まります。

Data body

```
RVM word
|  0xF9   | "00"  | Lock  | SNI  | Spill Num (8-bit) |           Event Num (12-bit)          |
```

*Lock*はHRMのlock bitです。1でないといけません。*SNI*はspill number incrementであり、1であればスピル番号が変わったことを示しますが、そのように動作しているかどうかはチェックしていません。 *Spill Num*と*Event Num*はそれぞれHRMが受信したスピルナンバーとイベントナンバーです。

```
SCR word
|  SCR block (4-bit)   |                          Counter (28-bit)                          |
```

*SCR Block*そのデータがどのSCRブロックに属しているのかを示します。SCRデータにはチャンネル番号を示すフィールドが存在しません。データはチャンネルの番号の低いほうから高いほうへ順番に並んでいます。

SCR block bits	入力ポート
0x8	Main in U
0x9	Main in D
0xA	Mezzanine U
0xB	Mezzanine D

4.7 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、異なる点のみ記述します。

Firmware固有名と現在の最新版

固有名	0x30cc
メジャーバージョン	0x03
マイナーバージョン	0x04

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にした。EventBufferの深さを2048から4096にしてpgfullを4058にした。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。
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。
v1.8	2017.12.19	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のワード数ビット幅を11-bitから12-bitへ変更。
v1.9	2018.02.02	>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も抱えているため、該当部分を改善した。
v2.x	-	未公開
v3.4	2021.08.01	FMPとSDSを追加。Builder busの導入。Local busの構造変更。

モジュール動作概要

HUL MH-TDCではRMで動作未定義となっていた残りのmezzanine slot Dとメイン入力ポートに機能が割り当てられています。Mezzanine slotにはDCR v1 (v2)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 slot 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slot Uに割り当てられている32ch分のデータは削除されます。

MH-TDCは300 MHz 4相の多相クロック式のTDCを実装しており、1-bit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、トリガーからさかのぼる事の出来る時間の長さは13.7 us、時間分解能は300 ps (σ)です。

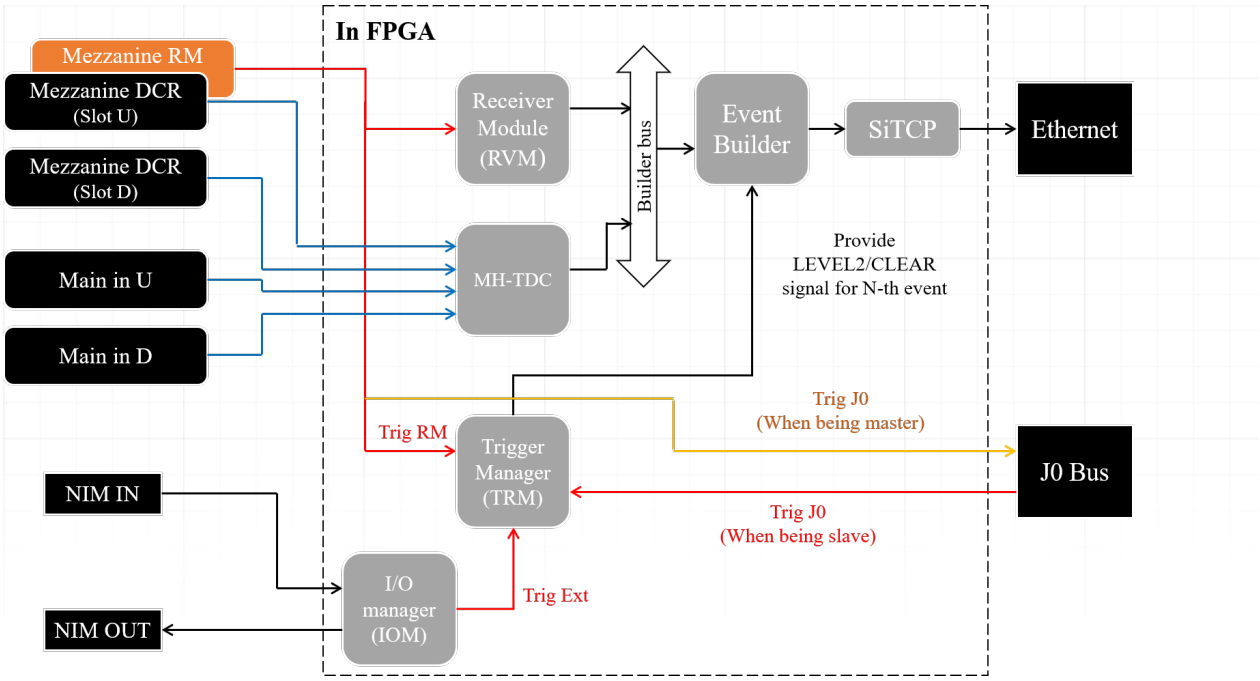

Figure 7: HUL MH-TDC FWの構造。

4.7.1 レジスタマップ

以下のテーブルはHUL MH-TDC専用のレジスタです。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。 HUR RMの異なる部分は赤字で示してあります。

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: TRM (module ID = 0x00)				
SelectTrigger	0x0000'0000	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 = 0x01)				
DaqGate	0x1000'0000	R/W	1	DAQ gateのON/OFF。DAQ gateが0だとTRMはtriggerを出力できない。
EvbReset	0x1010'0000	W	-	このアドレスへ書き込み要求することでEVB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。
IO Manager: TDC (module ID = 0x02)				
EnableBlock	0x2000'0000	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：Main in U 2bit目：Main in D 3bit目：Mezzanine U 4bit目：Mezzanine D
PtrOfs	0x2010'0000	R/W	11	内部制御変数。ユーザーは触らない。
WindowMax	0x2020'0000	R/W	11	Ring bufferからヒットを探す時間窓の上限値。1-bitが6.666... nsに相当。詳細は後述。
WindowMin	0x2030'0000	R/W	11	Ring bufferからヒットを探す時間窓の下限値。1-bitが6.666... nsに相当。詳細は後述。
IO Manager: IOM (module ID = 0x03)				
NimOut1	0x3000'0000	R/W	4	NIMOUT1へ何を出力するかを設定する。
NimOut2	0x3010'0000	R/W	4	NIMOUT2へ何を出力するかを設定する。
NimOut3	0x3020'0000	R/W	4	NIMOUT3へ何を出力するかを設定する。
NimOut4	0x3030'0000	R/W	4	NIMOUT4へ何を出力するかを設定する。
ExtL1	0x3040'0000	R/W	3	extL1にどのNIMINを接続するか設定。
ExtL2	0x3050'0000	R/W	3	extL2にどのNIMINを接続するか設定。
ExtClr	0x3060'0000	R/W	3	Ext clearにどのNIMINを接続するか設定。
ExtBusy	0x3070'0000	R/W	3	Ext busy入力にどのNIMINを接続するか設定。
ExtRsv2	0x3080'0000	R/W	3	Ext rsv2入力にどのNIMINを接続するか設定。

4.7.2 各ブロックの機能

Trigger Manager (TRM)
機能およびレジスタはHUL RMと同様です。

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`WindowMax`レジスタおよび`WindowMin`レジスタによって設定可能です。これらのレジスタは1-bitがコースカウント精度の11-bit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busyが出力されます。

Channel bufferからblock bufferへデータをまとめる際に、1 ch/eventに許される最大ヒット数が設定されます。1 ch/eventに許されるヒットはTDC chの大きいほうから16ヒットまでです。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	0.833... 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)

レジスタラベル	レジスタ値	備考
		NIMOUTへ出力可能な内部信号
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L2 trigger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。

以下はIOMレジスタの初期値のテーブルです。

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
ExtLClear	Reg_i_default	0
ExtLBusy	Reg_i_nimin3	NIMIN3
ExtLRsv2	Reg_i_nimin4	NIMIN4

4.7.3 HUL上のスイッチ・LEDの機能

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。

4.7.4 DAQの動作

データフローを図に示します。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イベントビルトします。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。

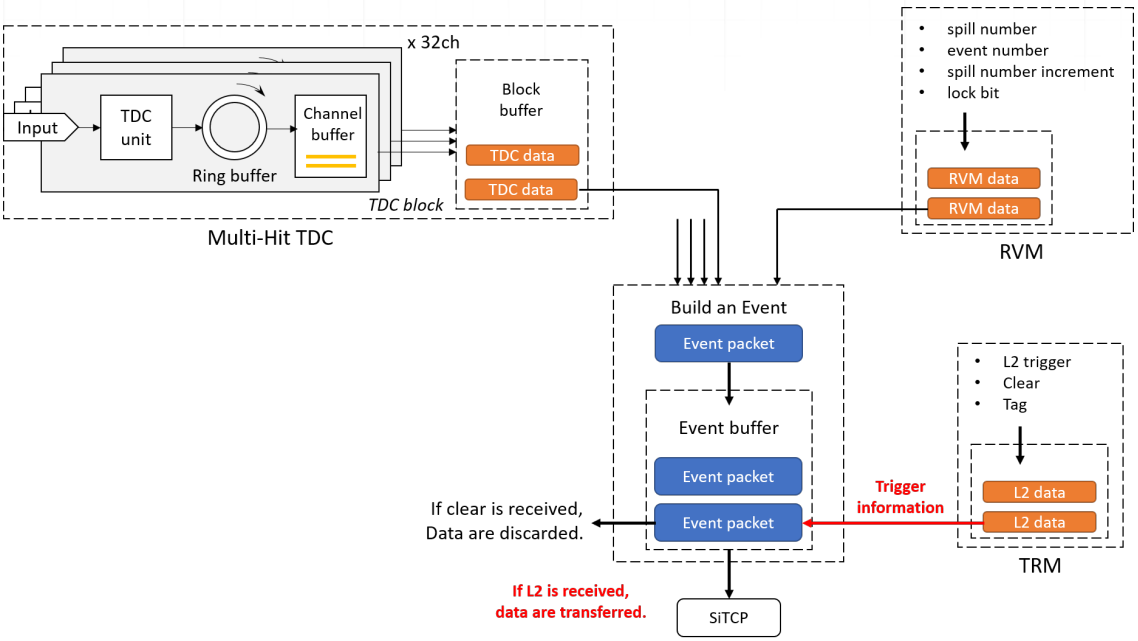

Figure 8: HUL MH-TDC FWのDAQデータフロー。

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種別	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	サーチ窓幅に依存	Ring bufferからヒット情報を探している間、すなわち`(WindowMax-WindowMin) x 6.66.. ns`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。つまりTCP転送が追いつかない事を意味するので、実質的にSiTCP fullと同等です。
SiTCP 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

Header word

```
Header1 (Magic word)
MSB                                                                         LSB
|                                        0xFFFF30CC                          |

Header2 (event size)
|       0xFF       |      0x00       |    "0000"    |         Number of word (12-bit)          |
```

*Number of word*はデータボディに何ワード含まれるかを示します。ヘッダの3ワードは含みません。

```
Header3 (event number)
|       0xFF       | HRM exist | "000" | Tag (4-bit) |         Self counter (16-bit)           |
```

*HRM exist*が1の場合HRMメザニンカードが取り付けられていることを示します。すなわち、データボディにRVMのデータワードが存在します。 *Tag*はTRMから受信した4-bitのタグ情報です。下位3ビットがMTMが送信したイベントナンバーの下位3ビットに相当し、4ビット目がスピルナンバーの最下位ビットです。 *Self counter*はイベントが送信されるたびにインクリメントされるローカルイベント番号です。0から始まります。

Data body

```
RVM word
|  0xF9   | "00"  | Lock  | SNI  | Spill Num (8-bit) |           Event Num (12-bit)           |
```

*Lock*はHRMのlock bitです。1でないといけません。*SNI*はspill number incrementであり、1であればスピル番号が変わったことを示しますが、そのように動作しているかどうかはチェックしていません。 *Spill Num*と*Event Num*はそれぞれHRMが受信したスピルナンバーとイベントナンバーです。

```
TDC word
|  Magic word (8-bit)  |  "0" + Ch (7-bit)  | "00" |           TDC (14-bit)                  |
```

*Magic word*の定義は以下のようになります。

- 0xCC Leading
- 0xCD Trailing

*Ch*には0から127までのチャンネル番号が入ります。チャンネルアサインについては2章を参照してください。
*TDC*はTDCデータです。

4.8 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つあるため若干複雑です。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

以前はリーディングエッジのみ測定できるファームウェアと両エッジ測定できるファームウェアは区別されていましたが、新しいファームウェアでは1つに統合されました。測定するエッジの選択はレジスタで行います。

固有名	0x80cc
メジャーバージョン	0x05
マイナーバージョン	0x00

更新歴

バージョン	リリース日	変更点
v2.5	2017.12.19	リーディング測定FWの初期版
v2.6	2018.02.02	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	リーディング・トレーリング測定FWの初期版
v3.3	2018.02.02	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。
v4.5	2021.08.01	SEMとXADCの導入。Builder busの導入。測定エッジのレジスタによる選択を可能に変更。 XDC上にあった軽微な間違いを修正（v4.3とは機能的に変わらない）。 このバージョンまでTDCサンプリングクロックとシステムクロックの周波数は520 MHzと130 MHz。
v5.0	2023.01.17	メザニンカードからのトリガー信号出力を実装した。 Local bus bridgeを修正した。Mezzanine FW v5.0はv3.7以前のHUL HRTDC BASE ファームウェアと互換性がない。 TDCサンプリングクロックとシステムクロックを500 MHzと125 MHzへ変更。

モジュール動作概要

Mezzanine HR-TDCとHUL HR-TDC BASEを接続した状態のブロック図を図に示します。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BusBridgeを通じて2段階アクセスでレジスタを制御します。BsuBridge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、ソフトウェアのセクションでもう一度説明します。

時間測定のみを行うMezzanine HR-TDCと、イベントビルトやトリガー制御、および各IOの管理を行うBASE側に分かれます。トリガー情報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が出力したlevel 1 triggerのみが送信されます。この信号がmezzanine HR-TDCにとってはcommon stop信号です。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リングバッファ長は15.7 us、時間分解能は25 ps (σ) (common stopに対して)、20 ps (σ) (チャンネル間の差分)です。

DAQデータをmezzanineからBASEへ高速で転送するために、5線の信号線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8 ns (4 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、event builderに渡す準備をします。

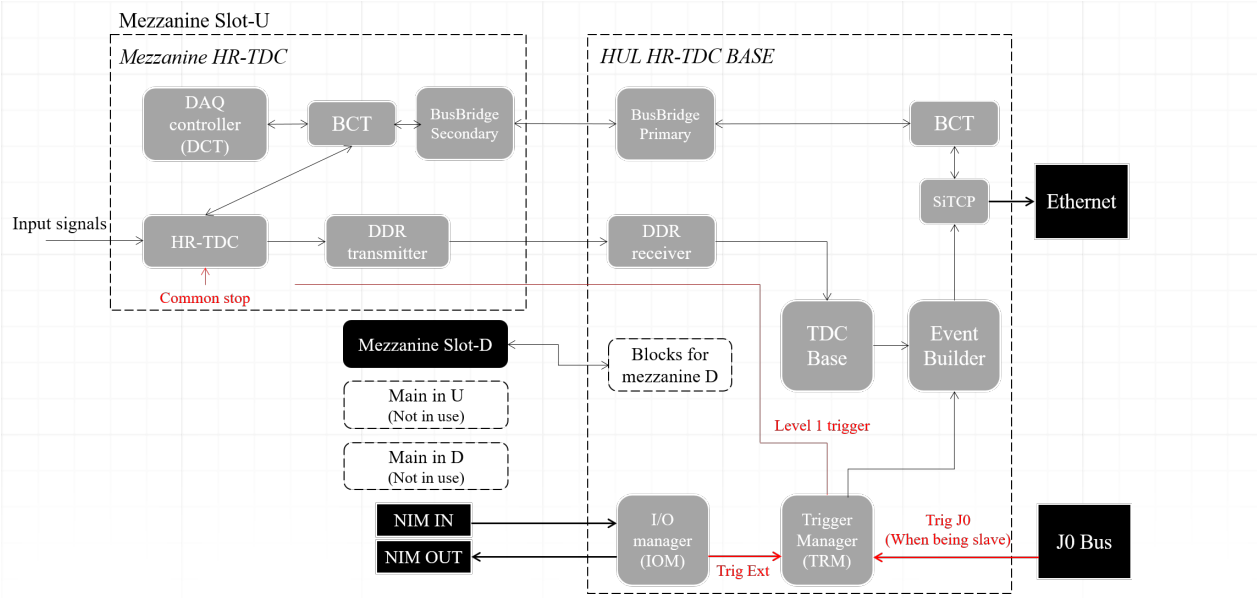

Figure 9: HUL HR-TDC FWのブロック構成。

4.8.1 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500 MHz (2.0 ns)のため、2.0 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。簡単には「2000 ps/最大tap番号」がtapあたりのdTであり、TDC 1-bit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estimatorと呼びます。

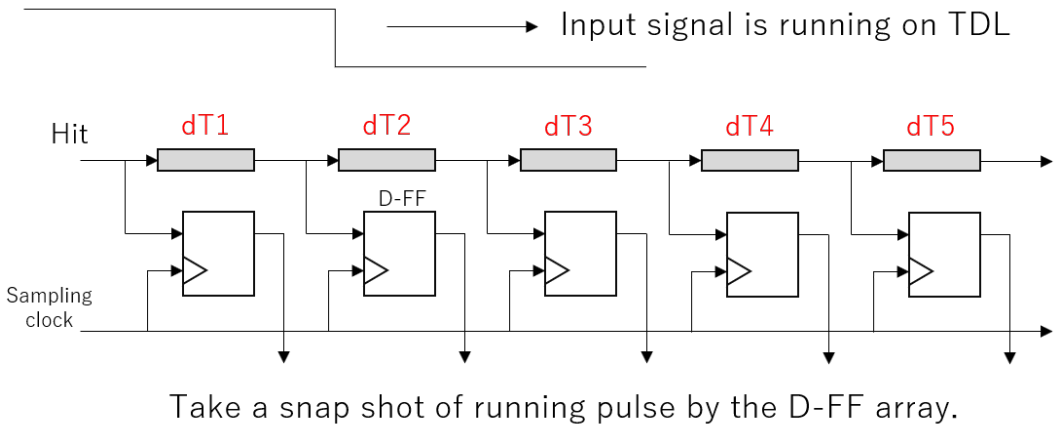

Figure 10: Tapped-delay-lineの概念図。

時間校正

Estimatorを生成するための手順を図に示します。横軸をtap番号として、時間相関のない入力対するヒストグラムを生成します。各ビンの重さがdTの値に比例します。ここでヒストグラムの総エントリー数は固定値であり、ある値 (0x7ffff)に達するまでヒストグラムへのデータフィルを続けます。ヒストグラム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wiとすると、N番目のestimatorは

$$E_n = w_n/2 + \sum_{0}^{n-1}(w_i)$$

として計算されます。そのため、FPGA内部にはfine count ヒストグラムを生成するための機構とfine countからestimatorへ変換して出力する回路が必要になります。最も分かりやすい方法は検出器の信号を使ってfine countヒストグラム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。）ヒストグラム生成はDAQとは無関係であるため、入力信号は全て自動的にヒストグラム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、クロックを使ってヒストグラム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TDL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でヒストグラム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の最初に校正することを想定しています。

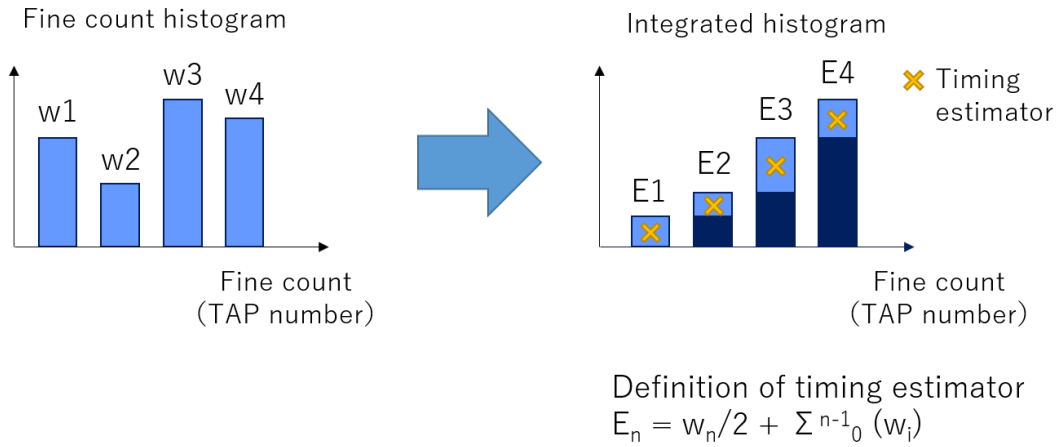

Figure 11: Fine count histogramからEsitomatorを生成する手順。

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19-bit幅でそのままでは過剰なので、下位8-bitを捨てて11-bit幅にしてから出力します。対してもう一方のRAMではヒストグラム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、`Controll::AutoSw`と`ReqSwitch`です。`Controll::AutoSw`が1であれば自動切り替え、0であれば`ReqSwitch`へ書き込み動作をするとRAMが手動で切り替わります。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、`Controll::Through`を1にすることでfine countが直接現れます。

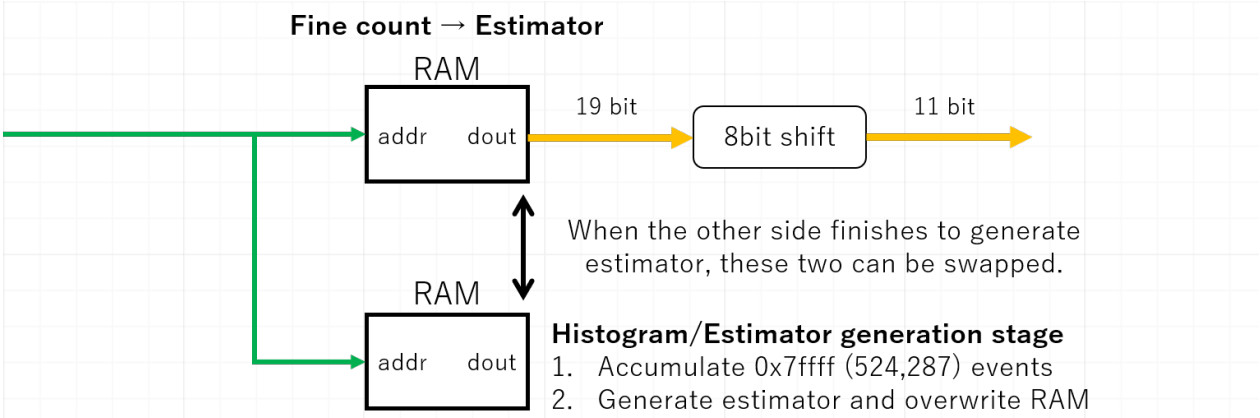

Figure 12: Estimator LUTの切り替えパターン。

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、2.0 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125 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の中で一度部分的にイベントビルドされ、HUL側へ転送されます。簡単に時間に直すにはestimatorの最大値2048と500 MHzのクロックの値を使って、 時間 = TDC value/2048/ClkFreq (ClkFreq = 0.50 GHz (FW ver 5.0 以降) もしくは = 0.52 GHz (FW ver 4.5以前)) 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い場合は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 `WindowMax` レジスタおよび `WindowMin` レジスタによって設定可能です。これらのレジスタは1-bitがコースカウント精度の11-bit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精度が異なります。

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。

トリガー出力

Version 5.0よりHR-TDCからトリガー信号を出力できるようになりました。 前述のhit bitを全てのチャンネルに対して論理和取り出力します。 トリガー出力を利用しない場合、TrigMaskレジスタによりチャンネル毎にマスクする事が出来ます。

High-resolution TDC 仕様	
TDC精度	~30 ps
コースカウント精度	8.0 ns
Ring buffer長	16.3 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

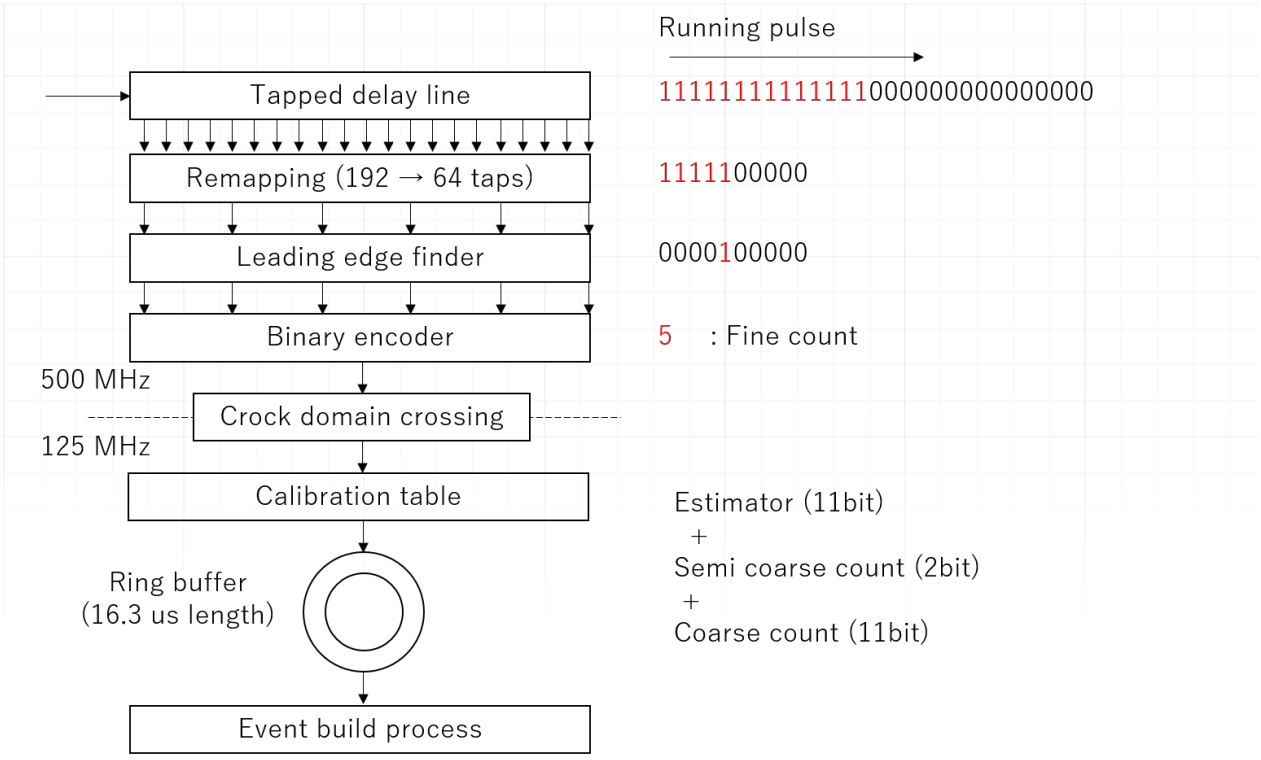

Figure 13: HR-TDCのブロック構造。

4.8.2 レジスタマップ

Mezzanine HR-TDCの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`RegisterMap.hh`内で`namespace HRTDC_MZN`に属するものです。 Mezzanine HR-TDCへはBus Bridge Primary (BBP)を通じてバスブリッジしてアクセスするため、RBCPのアドレスで直接指定することは出来ません。 **Version 5.0よりlocal bus bridgeを修正したことでアドレス範囲が12-bit幅から16-bit幅へ拡張されました。**

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: DCT (module ID = 0x0)				
TestMode	0x0000	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++ソフトウエアのddr_initialize内部で使用する。
ExtraPath	0x0010	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	0x0020	R/W	1	DAQ gate。1だとcommon stopがHR-TDCへ入力される。
EnBlocks	0x0030	R/W	2	Leading/trailing測定ブロックを有効化する。1ビット目がleading block、2ビット目がtrailing block。デフォルトで0のため必ず設定する必要がある。
DAQ Controller: TDC (module ID = 0x01)				
Control	0x1010	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) StopDout (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。StopDout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。
ReqSwitch	0x1020	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	0x1030	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	0x1040	R/W	11	内部制御変数。ユーザーは触らない。
WindowMax	0x1050	R/W	11	Ring bufferからヒットを探す時間窓の上限値。1bitが8.0 nsに相当。詳細はMH-TDCを参照。
WindowMin	0x1060	R/W	11	Ring bufferからヒットを探す時間窓の下限値。1bitが8.0 nsに相当。詳細はMH-TDCを参照。
TrigMask	0x1070	R/W	32	各チャンネルのトリガー出力をマスクする。各ビットがチャンネルに対応しており、0にセットするとマスクされる。例：チャンネル0をマスクする場合、0xFFFF'FFFEをセットする。
DAQ Controller: SDS (module ID = 0xC)				
SdsStatus	0xC000	R	8	SDSモジュールのステータスを取得します。

レジスタ名	アドレス	読み書き	ビット幅	備考
XadcDrpMode	0xC010	R/W	1	XADCのDRPモード選択をします。 • 0x0: Read mode • 0x1: Write mode
XadcDrpAddr	0xC020	R/W	7	XADCのDRP addressを与えます。
XadcDrpDin	0xC030	R/W	16	XADCのDRPのデータ入力を与えます。
XadcDrpDout	0xC040	R	16	XADCのDRPのデータ出力を取得します。
XadcExecute	0xC050	W	-	XADCのDRPアクセスを実行します。
SemCorCount	0xC0A0	R	16	訂正可能なSEUをSEMが訂正した回数を取得します。
SemRstCorCount	0xC0B0	W	-	SemCorCountをリセットします。
SemErroAddr	0xC0C0	W	40	SEMのinject_addressポートに入力するアドレスを与えます。
SemErroStrobe	0xC0D0	W	-	SEMのinject_strobeポートにパルスを入力します。
DAQ Controller: BCT (module ID = 0xE)				
Reset	0xE000	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	0xE010	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	0xE020	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。

4.8.3 Mezzanine上のスイッチ・LEDの機能

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	-	未使用
2	-	未使用
3	-	未使用
4	-	未使用

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種別	BUSY長	備考
Sequence busy	サーチ窓幅に依存	Ring bufferからヒット情報を探している間、すなわち`(WindowMax-WindowMin) x 8.0 ns`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。つまりTCP転送が追いつかない事を意味するので、実質的にSiTCP fullと同等です。

4.8.4 HUL HR-TDC BASEについての詳細説明

HUL HR-TDC BASEはDDR receiverとBctBusBridge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。BctBusBridgeはmezzanine slot upとdownにそれぞれ独立で用意されています。BusBridgeはHUL側のBCTとメザニン側のBCTの橋渡しをします。HUL側のBCTからはBusBridgeはローカルモジュールの一種、メザニン側のBCTからは外部リンクのように見えます。メザニンへアクセスするためには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HUL側のBCTはBusBridgePrimary (BBP)に対して、読み書きのコマンド値、メザニン側の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BBP内部へ格納します。2回目のアクションで`BBP::Exec`を呼び出すと、バスブリッジを実行し通信を行います。メザニン側のBCTを書き込みで駆動するか、読み出しで駆動するかは、1回目のアクションで指定したコマンド値で決まります。BBPは通信プロセス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`BBP::Exec`を呼ぶとHUL側のBCTがデッドロックします。この場合`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。BusBridge制御のためのC++関数は`BctBusBridgeFunc.cc`にまとめられています。詳細は6章で述べます。

HR-TDC BASE固有名と現在の最新版

固有名	0x80eb
メジャーバージョン	0x04
マイナーバージョン	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	2018.02.02	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。
v3.7	2021.08.01	SDS, FMPの追加。Builder busの導入。BCTの構造変更。
v4.0	2023.01.17	Mezzanine HR-TDC v5.0.対応版。Mezzanine HR-TDC v4.5以前のバージョンはサポートしていない。
v4.1	2023.02.24	v4.0が正しく動作しないことがあるバグを発見。v4.1はその対策版。

4.8.5 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存在しません。

加えて本ファームウェア用の `RegisterMap.hh` には先頭に `kEnSlotup` と `kEnSlotDown` 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が刺さっていないスロットはfalseにしてください。各実験の本番用ソフトに移植する際には無くても良い変数です。

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: TRM (module ID = 0x00)				
SelectTrigger	0x0000'0000	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 = 0x01)				
DaqGate	0x1000'0000	R/W	1	DAQ gateのON/OFF。DAQ gateが0だとTRMはtriggerを出力できない。
EvbReset	0x1010'0000	W	-	このアドレスへ書き込み要求することでEVB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。
InitDDR	0x1020'0000	W	-	DDR receiverへ向けて初期化要求を行う。
CtrlReg	0x1030'0000	R/W	4	DDR receiverを制御するためのレジスタ。詳細は後述。
Status	0x1040'0000	R	4	DDR receiverのステータスレジスタ。詳細は後述。
IO Manager: IOM (module ID = 0x02)				
NimOut1	0x2000'0000	R/W	4	NIMOUT1へ何を出力するかを設定する。
NimOut2	0x2010'0000	R/W	4	NIMOUT2へ何を出力するかを設定する。
NimOut3	0x2020'0000	R/W	4	NIMOUT3へ何を出力するかを設定する。
NimOut4	0x2030'0000	R/W	4	NIMOUT4へ何を出力するかを設定する。
ExtL1	0x2040'0000	R/W	3	extL1にどのNIMINを接続するか設定。
ExtL2	0x2050'0000	R/W	3	extL2にどのNIMINを接続するか設定。
ExtClr	0x2060'0000	R/W	3	Ext clearにどのNIMINを接続するか設定。
ExtBusy	0x2070'0000	R/W	3	Ext busy入力にどのNIMINを接続するか設定。
cntRst	0x2090'0000	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。
Bus Bridge Primary: BBP (module ID = 0x3, 0x4)				
Txd	0x3000'0000	W	32	Slot-UのセカンダリFPGA (mezzanine card上のFPGA)に対してlocal bus bridgeを介してへ書き込むデータ。
Rxd	0x3010'0000	R	32	Slot-UのセカンダリFPGAからlocal bus bridgeを介して読み出したデータ。
Exec	0x3100'0000	W	-	Bus bridge primaryを駆動しslot-UのセカンダリFPGAと通信を行うためのスタート信号をアサートする。
Txd	0x4000'0000	W	32	Slot-DのセカンダリFPGA (mezzanine card上のFPGA)に対してlocal bus bridgeを介してへ書き込むデータ。
Rxd	0x4010'0000	R	32	Slot-DのセカンダリFPGAからlocal bus bridgeを介して読み出したデータ。
Exec	0x4100'0000	W	-	Bus bridge primaryを駆動しslot-DのセカンダリFPGAと通信を行うためのスタート信号をアサートする。

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。

レジスタラベル	レジスタ値	備考
RegL1Ext	0x1	NIMINからL1 triggerを選択。
RegL1J0	0x2	J0 busからのL1 triggerを選択。
RegL2Ext	0x8	NIMINからのL2 triggerを選択。
RegL2J0	0x10	J0 busからのL2 triggerを選択。
RegClrExt	0x40	NIMINからのClearを選択。
RegClrJ0	0x80	J0 busからのClearを選択。
RegEnL2	0x200	0: L2=L1 trigger、1: L2=L2入力
RegEnJ0	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。

DAQ controller (DCT)

DCT内部の `CtrlReg` と `Status` が示すビットの詳細を述べます。

レジスタラベル	Bit番号	備考
		CTRLレジスタの内訳
RegTestModeU	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。
RegTestModeD	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。
EnableU	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。
EnableD	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。
FRstU	5th bit (0x10)	このビットを立てるとSlot-UのFPGAに倒してフォースリセット信号をアサートする。v3.7以前のファームウェアではMIFブロックに実装されていた機能。
FRstD	6th bit (0x20)	このビットを立てるとSlot-DのFPGAに倒してフォースリセット信号をアサートする。v3.7以前のファームウェアではMIFブロックに実装されていた機能。
		Statusレジスタの内訳
BitAlignedU	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。
BitAlignedD	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。
BitErrorU	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。
BitErrorD	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。

レジスタラベル	レジスタ値	備考
		NIMOUTへ出力可能な内部信号
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_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。
Reg_o_TrigOutU	0xD	Slot-Uのmezzanine HR-TDCから出力されるトリガー出力をアサインします。
Reg_o_TrigOutD	0xE	Slot-Dのmezzanine HR-TDCから出力されるトリガー出力をアサインします。
Reg_o_TrigOutUD	0xF	Slot-U/D両方のmezzanine HR-TDCから出力されるトリガー出力の論理和をアサインします。
		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の初期レジスタです。

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
ExtLClear	Reg_i_default	0
ExtLBusy	Reg_i_nimin3	NIMIN3
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	Not in use	未使用
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	Not in Use	
7	Not in Use	
8	Level	IOMのDIP8から出力されるレベルです。

LED番号	備考
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	点灯中はDIP SW2のLEDがONです。

4.8.6 DAQの動作

データフローを図に示します。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部分的なイベントビルドが行われデータが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イベントビルドします。

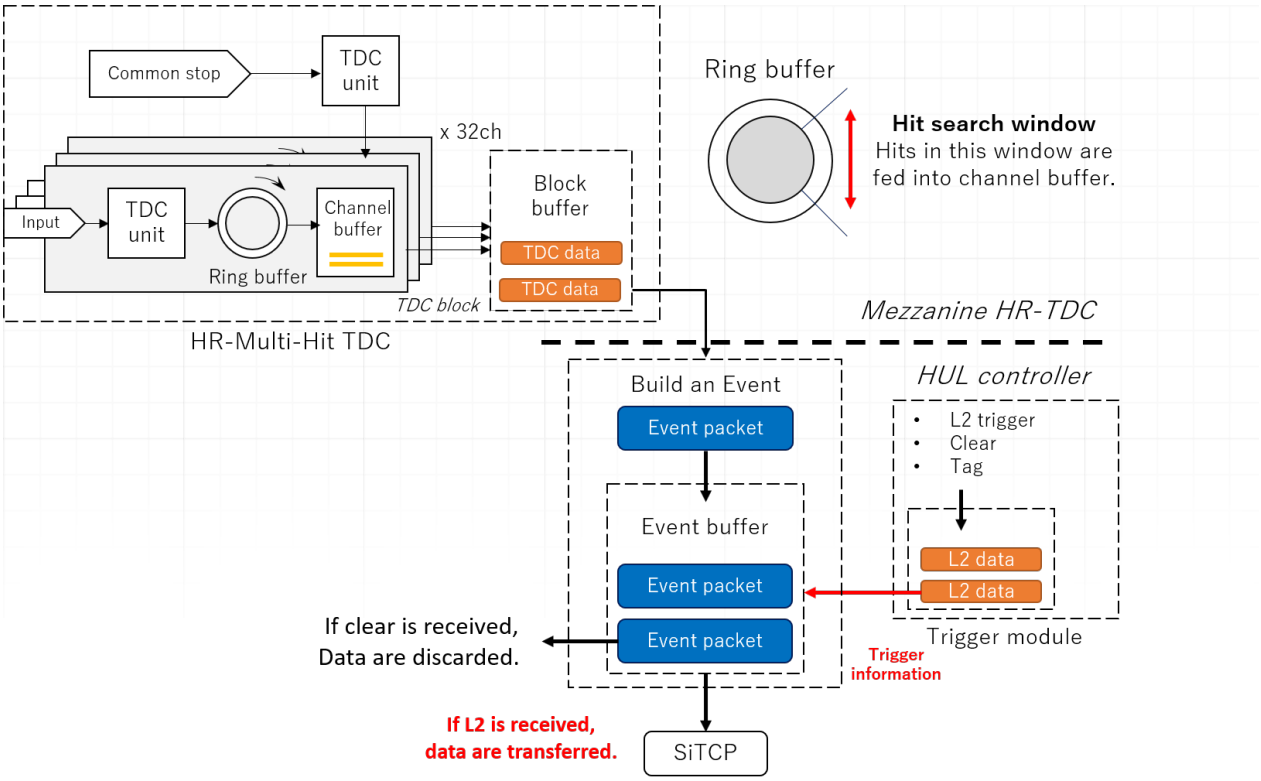

Figure 14: Mezzanine HR-TDCとHUL HR-TDC BASEを合わせたDAQのデータフローブロック図

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種別	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 busy	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。つまりTCP転送が追いつかない事を意味するので、実質的にSiTCP fullと同等です。
SiTCP 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

```
Header1 (Magic word)
MSB                                                                   LSB
|                                    0xFFFF80EB                         |

Header2 (event size)
|   0xFF00   |   OvrFlow   |   "000"   |          Number of word (12-bit)          |
```

*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。*OverFlow*はHUL全体で1chでもover flowチャンネルがあると立ちます。

```
Header3 (event number)
|   0xFF   | "0000" |  Tag (4-bit)  |                Self counter (16-bit)                  |
```

*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。

```
Sub-headers
|   0xFA00  | "0" |  OverFlow  |  StopDuout  |  Through  |          # of word (12-bit)         |
|   0xFB00  | "0" |  OverFlow  |  StopDuout  |  Through  |          # of word (12-bit)         |
```

それぞれmezzanine HR-TDCのヘッダです。*0xFA00*と*0xFB00*がそれぞれslot-UとDに対応します。*OverFlow*はそれぞれのメザニン中での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`StopDout`と`Through`の状態を示します。*Number of word*は各メザニンのワード数を示します。

Data body

```
|  Magic word (3-bit)  |  Ch (5-bit)  |                    TDC (24-bit)                     |
```

*Magic word*は以下のように定められています。

- 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。

4.9 Three-dimensional matrix trigger

このファームウェアのバージョン表記は他のFWと異なっており、メジャーバージョンが1ですがFMPやSDSは搭載されています。

固有名	0xe033
メジャーバージョン	0x01
マイナーバージョン	0x01

更新歴

バージョン	リリース日	変更点
v1.1	2020.12.02	実験で利用した最終版

動作概要

このファームウェアはJ-PARC E03実験とE42実験の際に用いられたマトリックスコインシデンストリガーです。似たような機能を持つマトリックストリガー回路を作成したい場合の例題として利用してください。本ファームウェアにはデータ収集機能はありません。このFWでは3種類のホドスコープの三次元相関からトリガーを生成します。図にブロック図を示します。この実験ではBH2、TOF、SCHという3種類のタイミングホドスコープ間のマトリックス相関を用います。丸カッコ内の数字はチャンネル番号を表しており、合計14336パターン (8x28x64) の組み合わせが発生します。入力はニ重FFで同期されます。クロック速度は350 MHzです。後述のDWGやマトリックスパターンのブロックも同様のクロックで駆動されています。BH2がNIM-INへも繋がれているのはテストを簡便に行うためです。このファームウェアはメザニンスロットがSCHの入力を受けるためDCR v1/v2が必須です。

同期された入力信号はDelay Width Generator (DWG) で幅と遅延時間の調整がされます。各DWGはRBCPを通じて調整が可能です。350 MHz (2.857... ns) のクロック精度で32段階の調整が可能です。詳しくはソフトウェアの項で述べます。DWGではパルス出力中にもう一度パルス入力があった場合2つのパルスを繋げます。麻痺型モデルで表されるデッドタイムの振る舞いと同様です。

MTX3DとMTX2Dはそれぞれ三次元マトリックスコインシデンスと二次元マトリックスコインシデンスのブロックです。このFWでは1つ三次元トリガーと2つの二次元トリガーが実装されており、それぞれマトリックスパターンを設定可能です。RBCPを用いて全てのマトリックスエレメント1つ1つのOn/Offの切り替えが可能です。どのように実現しているかは後述します。

各マトリックストリガーはIOMを通じてNIMOUTから出力可能です。このFWでは実験の要求から二次元マトリックスの出力を三次元マトリックスの結果でVETOする経路が用意されています。この実験では三次元トリガーがビームVETOの役割を果たしていたためタイミングが良く分かっており、二次元トリガーに対しては固定長ディレイでVETO位置を調整しています。このあたりは実験条件に合わせて変更してください。

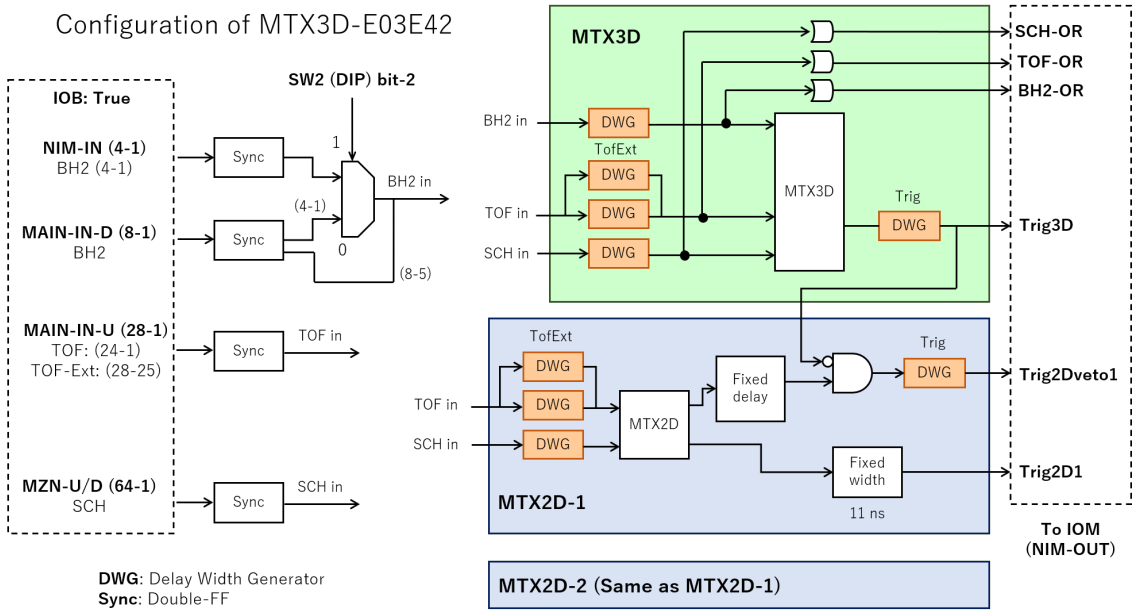

Figure 15: Matrix3D triggerのブロック図。左側入力、右側が出力。

4.9.1 マトリックスパターンの設定方法

14336パターンのスイッチをアドレスで解決しようと思うととてつもなく大きなエンコーダが必要になり、一般的にFPGAで実現するのは非現実的です。ここで3次元のレジスタを(28x64)x8と分解すると、8-bit幅・1792長のレジスタと捉えることが出来ます。すなわち、8-bit幅で長さが1792のシフトレジスタを用意することで、全てのレジ

スタビットの設定が可能となります。二次元の場合1-bit幅で長さが1792のシフトレジスタを用意します。RBCP (BCT) は同一のアドレスに1792回書き込むだけで設定が実現でき、極めてリソース効率が良いです。レジスタ設定はシステムクロックに対して低速なクロックで行っても良いため、このFWでは10 MHzのクロックでシフトレジスタを駆動しています。

4.9.2 DWGの構造

DWGは遅延と幅の生成をシフトレジスタへのビットパターンのプリセットによって実現しています。例えば先頭から `00011111000...` というビットパターンを入力があったタイミングでシフトレジスタへセットしたとします。そうするとこのパターンは遅延量が3でパルス幅が5の波形を与えます。遅延量と幅の情報からビットパターンへの変換はソフトで行う事にしています。最大幅と遅延量がそれぞれ32のため、DWGの取るレジスタ幅は64-bitです。

4.9.3 レジスタマップ

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: MTX2D-1 (module ID = 0x00)				
TofDWG	0x0000'0000	R/W	64	Tof (1-24) 入力のDWGレジスタを設定します。
TofExtDWG	0x0010'0000	R/W	64	Tof (25-28) 入力のDWGレジスタを設定します。
SchDWG	0x0020'0000	R/W	64	SCH入力のDWGレジスタを設定します。
TrigDWG	0x0030'0000	R/W	64	二次元トリガー出力のDWGを設定します。
EnableMtx	0x0040'0000	W	1	二次元マトリックスパターンを設定するアドレスです。シフトレジスタの最下位にレジスタを書きます。
ExecSR	0x0100'0000	W	-	二次元マトリックスパターンを設定するシフトレジスタを1つシフトさせます。
Trigger Manager: MTX2D-2 (module ID = 0x01)				
各レジスタはMTX2D-1と同様です				
Trigger Manager: MTX3D (module ID = 0x02)				
TofDWG	0x2000'0000	R/W	64	Tof (1-24) 入力のDWGレジスタを設定します。
TofExtDWG	0x2010'0000	R/W	64	Tof (25-28) 入力のDWGレジスタを設定します。
SchDWG	0x2020'0000	R/W	64	SCH入力のDWGレジスタを設定します。
Bh2DWG	0x2030'0000	R/W	64	SCH入力のDWGレジスタを設定します。
TrigDWG	0x2040'0000	R/W	64	三次元トリガー出力のDWGを設定します。
EnableMtx	0x2050'0000	W	8	三次元マトリックスパターンを設定するアドレスです。シフトレジスタの最下位にレジスタを書きます。
ExecSR	0x2100'0000	W	-	三次元マトリックスパターンを設定するシフトレジスタを1つシフトさせます。
Trigger Manager: IOM (module ID = 0x03)				
Nimout1	0x3000'0000	R/W	4	Nimout1ポートへの出力を設定します。
Nimout2	0x3010'0000	R/W	4	Nimout2ポートへの出力を設定します。
Nimout3	0x3020'0000	R/W	4	Nimout3ポートへの出力を設定します。
Nimout4	0x3030'0000	R/W	4	Nimout4ポートへの出力を設定します。

I/O Manager (IOM)

レジスタラベル	レジスタ値	備考
	NIMOUTへ出力可能な内部信号	
Reg_o_Trig3D	0x0	MTX3Dトリガーを出力します。
Reg_o_Trig2DVeto1	0x1	MTX3DでVETOされた後のMTX2D-1トリガーを出力します。
Reg_o_Trig2DVeto2	0x2	MTX3DでVETOされた後のMTX2D-2トリガーを出力します。
Reg_o_Trig2D1	0x3	MTX2D-1トリガーを出力します。
Reg_o_Trig2D2	0x4	MTX2D-2トリガーを出力します。
Reg_o_TofOR	0x5	TofORを出力します。
Reg_o_SchOR	0x6	SchORを出力します。
Reg_o_Bh2OR	0x7	Bh2ORを出力します。

4.10 Mass trigger (TOF based trigger)

このファームウェアは2023年現在更新がストップしています。Mezzanine HR-TDC v5.0や最新のソフトウェアとは互換性がありません。以前のバージョンの物を引き続き利用してください。

固有名	0x20d1
メジャーバージョン	0x03
マイナーバージョン	0x00

更新歴

バージョン	リリース日	変更点
v3.0	2020.12.02	実験で利用した最終版

Mass trigger (MsT) はJ-PARC K1.8での通称であり、機能的にはTOFベースのトリガー生成回路です。ダイポール磁気スペクトロメータを通過した粒子の軌道は大雑把に粒子の運動量と相関があります。2つのホドスコープの二次元マトリックス相関から運動量範囲を制限し、各マトリックスエレメントに対してTOFの分布を取ると粒子の質量ごとにTOFピークが分離します（低い運動量であれば）。Mass triggerはmezzanine HR-TDCとHULのメイン入力を用いて、二次元マトリックスとTOF情報によるトリガー生成を行うためのファームウェアです。HR-TDCの情報を得るためにはcommon stop入力が必要であるため、mass triggerはlevel2 decisionを行い、level2 triggerかもしくはfast clear信号を生成することが仕事です。

ブロック図を図に示します。MsTはHRMとHR-TDCのメザニンカードを必要としています。それぞれ、slot-Uとslot-Dへマウントしてください。Main INへ入力された信号は低速なTDCでデジタイズされヒットレジスタ情報を与えます。そのため、このファームウェアにおけるTOFは厳密には検出器間の時間差でなく、common stopとの時間差です。Mass triggerを利用するためにはlevel1 triggerが非同期回路で生成されている必要があります。この仕様はK1.8ビームラインの都合に合わせてあるため、真のTOFを計算させたい場合ファームウェアの改修が必要になります。

SCH (64ch)とTOF (32ch) 間の2048パターンに対してTOF windowのチェックを行います。この時、HR-TDCから返ってきたマルチヒットデータ全てに対して判定を行うため、同一チャンネルに対して複数回の判定が行われることがあります。1つでもアクセプト範囲にTOF値があればトリガーが発行されます。1つのアクセプト範囲にデータがない場合クリア信号が出力されます。アクセプト窓の設定はSiTCP経由で行います。MsTでトリガー判定を行いたいのはlevel1 triggerが物理トリガーの場合だけです。キャリブレーショントリガーの場合は判定無しでlevel2 triggerを発生させなければいけません。判定すべきlevel1 triggerかどうかを知らせるために、このファームウェアではpiK flagという信号をlevel1 triggerに続いて入力することになっています。Flag入力が無ければ判定を行わず、必ずlevel2 triggerが発行されます。

本ファームウェアにはデータ収集機能が存在します。HR-TDCとLR-TDCによって得られた時間情報と、MsTの判定結果がデータとして返ってきます。本ファームウェアはトリガー生成回路ですが、データ収集のためには外部からトリガー入力が必要です。

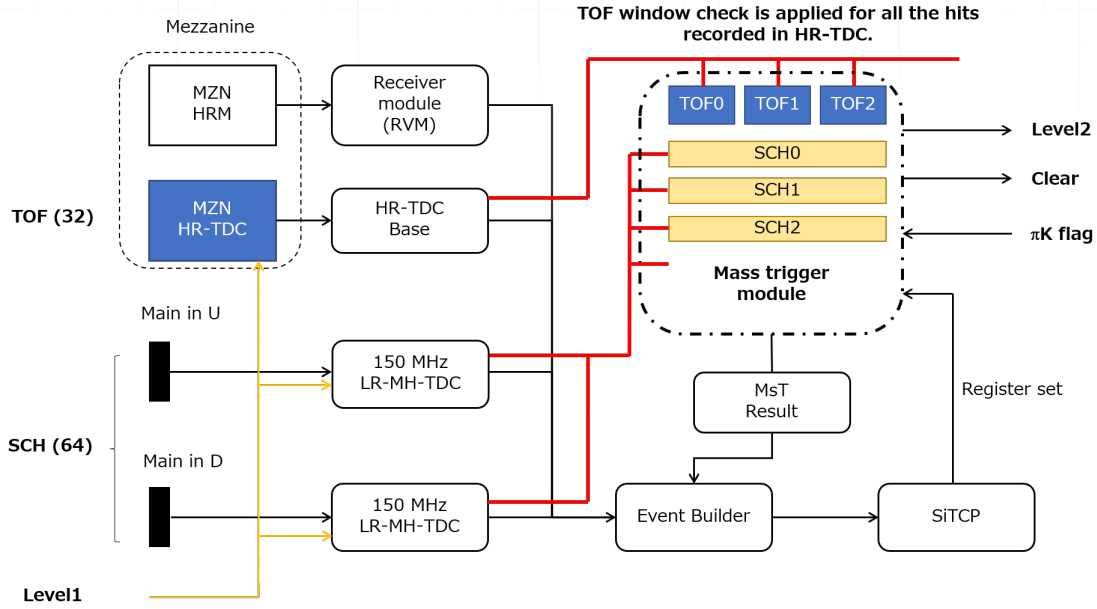

Figure 16: Mass triggerのブロック図。

タイミングチャート

K1.8ビームラインでmass triggerを導入する目的は、主にVMEモジュール用にfast clearを生成し不必要なバスアクセス時間を減らす事です。K1.8ビームラインではchained-block-transfer (CBLT) でVMEバスアクセスを行っていますが、おおよそ100 usのバスアクセス時間がかかります。VMEモジュール内のmulti-event bufferを活用して、バスアクセス時間をlevel1 triggerに対するbusyに含めない工夫はしていますが、潜在的なbusyであるためlevel1 triggerのレートが7 kHzあたりから急激にDAQ効率が悪くなります。そのような実験ではmass triggerを導入してlevel2判定を加えてバスアクセスの回数を減らします。

図に想定タイミングチャートを示します。Mass triggerはHR-TDCとLR-TDCにcommon stop入力が入った時点から動作を開始します。HR-TDCからデータの転送が終わると判定を開始します。Mass triggerはmulit-hitを全て処理するため、ここまでの時間は可変です。Level1から固定時間後に判定結果を出力するために`MST::TimerPreset`で出力までの時間を設定します。今のファームウェアバージョンでは500程度の大きな値を設定することを推奨しています。`MST::TimerPreset`は判定プロセスよりも優先度が高いため、指定時間後に判定が終わっていなかった場合後述のno decision flagを立てたうえで強制的にlevel2 triggerを出力します。No decision flagが頻繁に立つ場合`MST::TimerPreset`の値が小さすぎます。

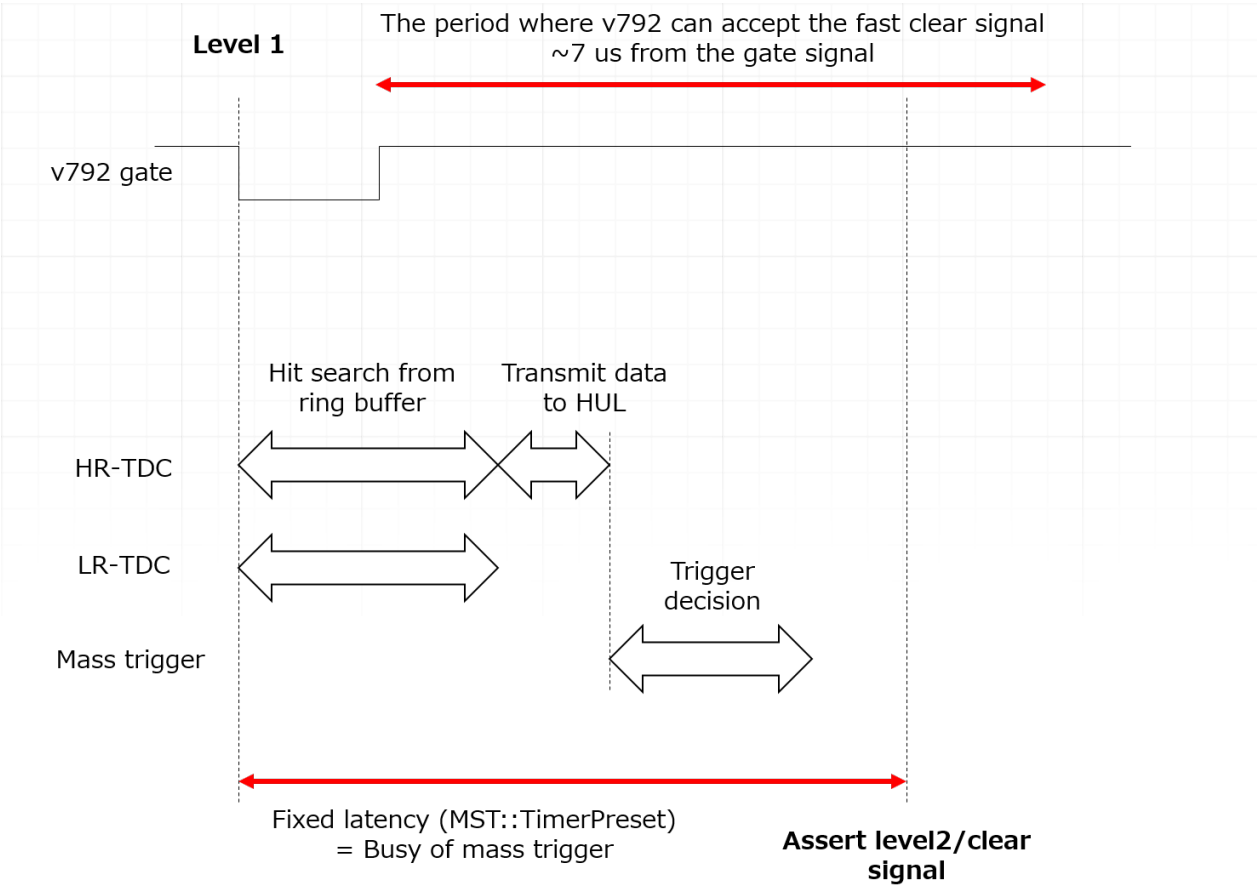

Figure 17: 想定しているlevel1 triggerからlevel2判定までのタイムチャート。

判定フロー

Level2 trigger判定のフロー図を図に示します。図の下側に書かれているリストは各判定状態におけるフラグ（および内部信号）の状況を示しています。判定動作はlevel1 trigger受信で開始します。TDCブロックから情報を集め判定回路が各チャンネル独立に動作します。`MST::TimerPreset`に指定した時間がたった後、piK flagを受信しているか、および判定プロセスがすべて終了しているかのチェックを行います。どちらかを満たしていない場合、no decision flagを立ててlevel2 triggerを出力します。次にHR-TDCから取得したTDC値がアクセプト範囲にあるかどうかのチェックを行います。1つでも存在すればmass trigger acceptとなり、level2 triggerが出力されます。1つもない場合clear判定となり、敗者復活判定へ続きます。Clear判定を下されたイベントはDAQで取得されないため、クリアしているイベントをサンプル検査するために`MST::ClearPreset`に指定した値に1度、敗者復活アクセプトを出します。敗者復活判定が下された場合consolation acceptのフラグが立ち、level2 triggerが出力されます。敗者復活の条件を満たさない場合fast clearが出力されます。

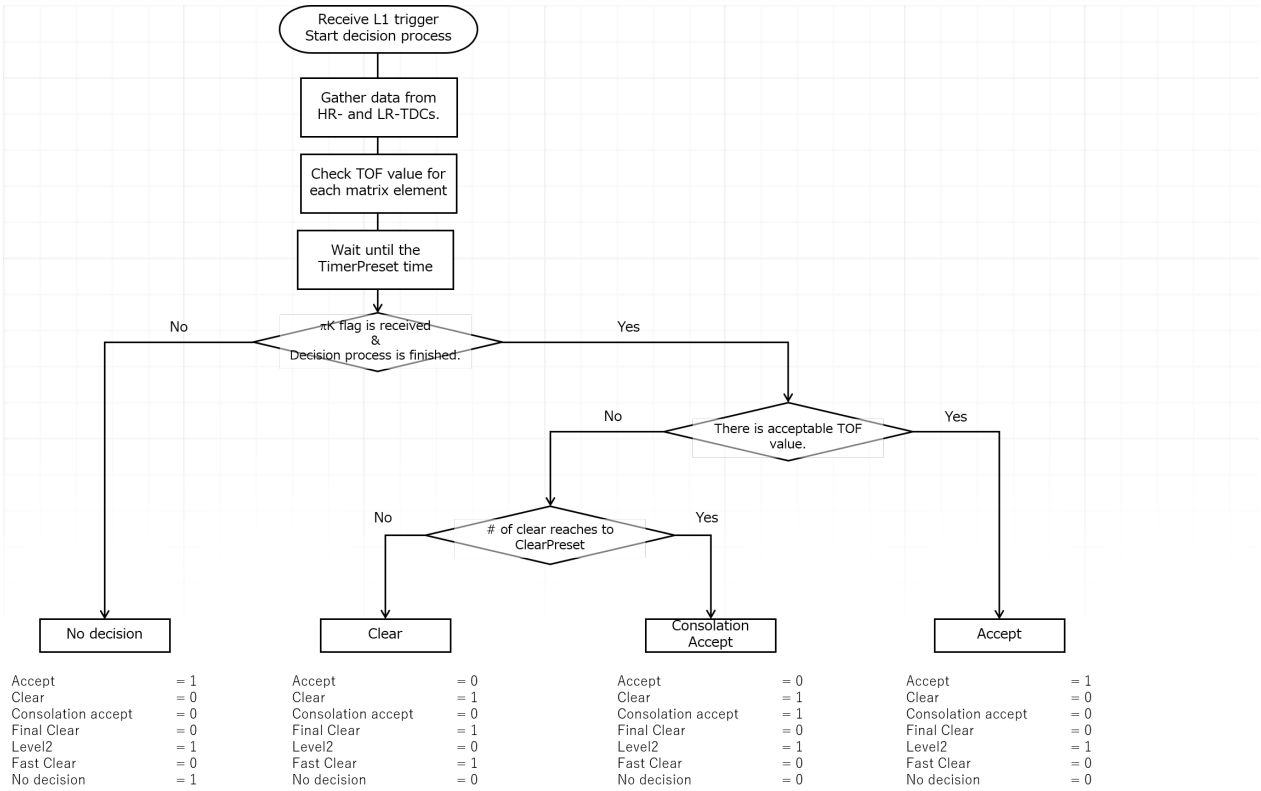

Figure 18: Mass triggerの判定フロー。

TOFのアクセプト範囲の設定

Mass triggerには2種類のTDC範囲設定を行うレジスタが存在します。`LRTDC::WinMax`と`LRTDC::WinMin`、およびmezzanine HR-TDCのウィンドウレジスタはcommon stop入力に対してヒットサーチを行う範囲を設定します。これは通常のTDCファームウェアと同様です。`MST::WixMax`と`MST::WinMin`はlevel2判定でアクセプトとなる範囲を与えます。Mass triggerでは32 chのHR-TDC入力と64 chのLR-TDC入力間の二次元マトリックスに対して、行列要素毎にこのレンジを設定することが出来ます。マトリックスコインシデンストリガーの時と同様に24-bit幅で長さが2048のシフトレジスタを採用しています。`MST::WixMax`と`MST::WinMin`の両方を0に設定すると、その行列要素に対しては判定を行いません。

4.10.1 レジスタマップ

レジスタ名	アドレス	読み書き	ビット幅	備考
Trigger Manager: TRM (module ID = 0x00)				
SelectTrigger	0x0000'0000	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 = 0x01)				
DaqGate	0x1000'0000	W	-	DAQ gateのON/OFF。DAQ gateが0だとTRMはtriggerを出力できない。
EvbReset	0x1010'0000	R/W	1	このアドレスへ書き込み要求することでEVB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。
InitDDR	0x1020'0000	W	-	DDR receiverへ向けて初期化要求を行う。
CtrlReg	0x1030'0000	R/W	4	DDR receiverを制御するためのレジスタ。
Status	0x1040'0000	R	4	DDR receiverのステータスレジスタ。
IO Manager: IOM (module ID = 0x02)				
NimOut1	0x2000'0000	R/W	5	NIMOUT1へ何を出力するかを設定する。
NimOut2	0x2010'0000	R/W	5	NIMOUT2へ何を出力するかを設定する。
NimOut3	0x2020'0000	R/W	5	NIMOUT3へ何を出力するかを設定する。
NimOut4	0x2030'0000	R/W	5	NIMOUT4へ何を出力するかを設定する。
ExtL1	0x2040'0000	R/W	3	extL1にどのNIMINを接続するか設定。
ExtL2	0x2050'0000	R/W	3	extL2にどのNIMINを接続するか設定。
ExtClr	0x2060'0000	R/W	3	Ext clearにどのNIMINを接続するか設定。
ExtBusy	0x2070'0000	R/W	3	Ext busy入力にどのNIMINを接続するか設定。
cntRst	0x2090'0000	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。
PiKTrig	0x20A0'0000	R/W	3	Mass triggerに対する物理トリガーフラグ入力。Level1 triggerに続いて入力があるとlevel2判定を行う。
IO Manager: MIF-Down				
Connect	0x3000'0000	R/W	-	MIF-Downからmezzanine HR-TDCのBCTへ向けて通信プロセスを開始する。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、メザニンのBCTへのアクセス方法が切り替わる仕様となっている。
Reg	0x3010'0000	R/W	20	MIF-Downに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。 • [19:8]: Local address • [7:0]: Register valus
ForceReset	0x3100'0000	W	-	MIF-Downの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。
Low-resolution TDC: LRTDC				
PtrOfs	0x4010'0000	R/W	11	内部制御変数。ユーザーは触らない。

レジスタ名	アドレス	読み書き	ビット幅	備考
WindowMax	0x4020'0000	R/W	11	Ring bufferからヒットを探す時間窓の上限値。1-bitが6.666... nsに相当。
WindowMin	0x4030'0000	R/W	11	Ring bufferからヒットを探す時間窓の下限値。1-bitが6.666... nsに相当。
Mass trigger: MST				
ClearPreset	0x5000'0000	R/W	7	このレジスタに書かれた値に1度、敗者復活アクセプトが出力される。0に設定すると敗者復活判定を行わない。
TimerPreset	0x5010'0000	R/W	9	Level1 triggerを受信してからlevel2 trigger/clearを出力するまでの遅延時間。
WinMax	0x5020'0000	W	24	アクセプトするTOF範囲の上限値を与えます。シフトレジスタの最下位にレジスタを書きます。
WinMin	0x5030'0000	W	24	アクセプトするTOF範囲の下限値を与えます。シフトレジスタの最下位にレジスタを書きます。
Exec	0x5040'0000	W	-	シフトレジスタを1つシフトさせます。
Bypass	0x5050'0000	R/W	1	このレジスタが1になると判定回路をバイパスしてlevel2 triggerを出力するようになります。実験中に一時的にmass triggerをバイパスしたい場合に利用します。

I/O Manager (IOM)

レジスタラベル	レジスタ値	備考
		NIMOUTへ出力可能な内部信号
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L2 trigger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。
Reg_o_clk1kHz	0xC	1 kHzのクロックを出力します。
Reg_o_Accept	0xD	Accept flagを出力します。
Reg_o_Clear	0xE	Clear flagを出力します。
Reg_o_ConsolationAccept	0xF	Consolation accept flagを出力します。
Reg_o_FinalClear	0x10	Final clear flagを出力します。
Reg_o_Level2	0x11	Level2 flagを出力します。
Reg_o_FastClear	0x12	Fast clear flagを出力します。
Reg_o_NoDecision	0x13	No decision flag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。

以下はIOMレジスタの初期値のテーブルです。

NIM出力ポート	初期レジスタ
NIMOUT1	Reg_o_ModuleBusy
NIMOUT2	reg_o_RML1
NIMOUT3	reg_o_RML2
NIMOUT4	reg_o_RMClr

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
ExtLClear	Reg_i_default	0
ExtLBusy	Reg_i_nimin3	NIMIN3
ExtLRsv2	Reg_i_nimin4	NIMIN4
cntRst	Reg_i_default	0
PiKFlag	Reg_i_nimin2	NIMIN2

4.10.2 HUL上のスイッチ・LEDの機能

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。

4.10.3 DAQの動作

Mass triggerからはLR-TDCとHR-TDCの測定結果、およびlevel2判定の結果（フラグ）が返ってきます。Level2判定はイベント毎に行うため、level2判定を行っている間BUSYになります。

Module Busyとなるタイミング

BUSYの定義は以下に列挙するBUSY信号のORになります。

BUSY種別	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	サーチ窓幅に依存	Ring bufferからヒット情報を探している間、すなわち`WindowMax`-`WindowMin`分のBUSYが出力されます。Mass triggerにはLR-TDCとHR-TDCが存在するため、長いほうが採用されます。
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。つまりTCP転送が追いつかない事を意味するので、実質的にSiTCP fullと同等です。
SiTCP 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。
Decision busy	`MST::TimerPreset`に依存	Level1 trigger受信からlevel2/clear出力までの間BUSYになります。`MST::TimerPreset`に依存します。

データ構造

ヘッダワード

```
Header1 (Magic word)
MSB                                                                   LSB
|                                    0xFFFF20D1                         |

Header2 (event size)
|       0xFF00       | OerFlow  |    "000"    |         Number of word (12-bit)          |
```

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。

```
Header3 (event number)
|   0xFF   | HRM bit | "000" |  Tag (4-bit)  |            Self counter (16-bit)          |
```

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。

```
Data body
RVM data
|  0xF9  | "00" |  Lock  |  SNI  |  Spill Num (8-bit))  |          Event Num (12-bit)         |

Mass trigger data
|                0x2000                   |   "0000'0000'0"   |  Mass trigger flag (7-bit)  |

HR-TDC block
Sub-hearder
|  0x8000  | "00" |  OerFlow  |  StopDuout  |  Through  |          # of word (11-bit)         |
HR-TDC data
|  Magic word (3-bit)  |  Ch (5-bit)  |                    TDC (24-bit)                       |

LR-TDC block
Sub-hearders
|  0xFC10  | "00" |  OerFlow  |  StopDuout  |  Through  |          # of word (11-bit)         |
|  0xFC20  | "00" |  OerFlow  |  StopDuout  |  Through  |          # of word (11-bit)         |
LR-TDC data
|   0xCC   |  "0"  |   Ch (7-bit)   |   "00000"   |                TDC (11-bit)               |
```

データボディ領域ではsub-headerに続いてその領域のデータが返ってきます。

Mass trigger flagの内訳

ビット番号	フラグ
0	No decision
1	Level2
2	Fast clear
3	Consolation accept
4	Final clear
5	Accept
6	Clear

4.11 Streaming TDC

固有名	0x11dc
メジャーバージョン	0x03
マイナーバージョン	0x05

更新歴

バージョン	リリース日	変更点
v3.5	2021.4.	

動作概要

このファームウェアはtrigger-less DAQ用に開発された外部トリガー無しに連続的に時間測定を行うTDCです。J-PARC MRの遅い取り出しの2秒間の間、1 nsの精度で時間測定を行うことを想定しています。Trigger-less DAQではフロントエンド回路上でトリガーによるイベント選別を行わないため、入力信号全てをデジタイズしPCへデータ転送を続けます。

HULはtrigger-less DAQにおいてdata streamingを行うにはデータリンクスピードが不十分であり、また十分なメモリ搭載していません。このファームウェアは連続時間測定の技術実証のために開発した側面が強いです。今後HULではこのファームウェアの開発は継続せず、**AMANEQ**というtrigger-less DAQ用に開発された回路に引き継ぐ予定です。Trigger-less DAQや連続読み出しに興味のある方は本多へ連絡をお願いいたします。

このような背景があるので、このファームウェアには試験的に実装した機能も存在します。ここでは汎用的に使えそうな部分のみ説明することにします。

ブロック構造

入力はmain-inの上側を利用します。そのほか、テスト入力、VETO入力、スピルゲート入力をNIMINポートから行います。本TDCのチャンネル数は32 chです。テスト入力が有効の場合、main in Uの入力の代わりにNIMIN2からの信号が全チャンネルに配られます。高繰り返しのテスト信号を入れるとデータレートがすぐにリンクスピードを上回るので、利用時にはレートに気を付けてください。VETO入力はHIGHの間入力信号をマスクします。スピルゲートよりもさらに細かく入力信号の選択をしたい場合に利用してください。スピルゲートはJ-PARCの遅い取り出しのスピルゲートを想定しています。スピルゲートがHIGHの状態の時だけ、本TDCはデータを送信します。本TDCは端的にはスピルスタートからの時間を連続的に測るTDCです。そのため、スピルゲートは2秒間ONで2秒間OFFのように周期的にやってくることが前提になっています。

時間計測Online Data Processing (ODP) blockで行われます。ここで入力信号の立ち上がりと立下りの両エッジの時間を測定し、エッジペアを見つけます。この段階でTime-Over-Thresholdが計算され、立下りの時刻情報は破棄されます。以後、データワード内には立ち上がり時刻とTOT値が含まれます。

ODPブロックまでは各チャンネル毎に独立にデータ処理されます。Vital blockではデータパスを1つにまとめ上げ、データ列へハートビートデータという特殊データの挿入を行います。本TDCのコースカウントは125 MHzのクロックで駆動されている16-bitカウンタ (heartbeat generator)によって付与され、500 μs程度で1周します。それ以上の長さで時刻再構成を行うために、heartbeat methodという方式を導入しています。Heartbeat generatorが1周するごとにデータ列にheartbeatデータを挿入します。解析ではheartbeatデータの数を数えることで、スピルスタートからの時間を再構成することが出来ます。Heartbeatデータはデータ列のちょうど区切りの位置に挿入されなければならないため、vital blockにはtime frameの切れ目を見つける工夫が施されています。

データレートがSiTCPのスピードを超えた場合の対処など、さらに詳しい情報は参考文献を参照してください。(R.Honda et al., PTEP, 2021)

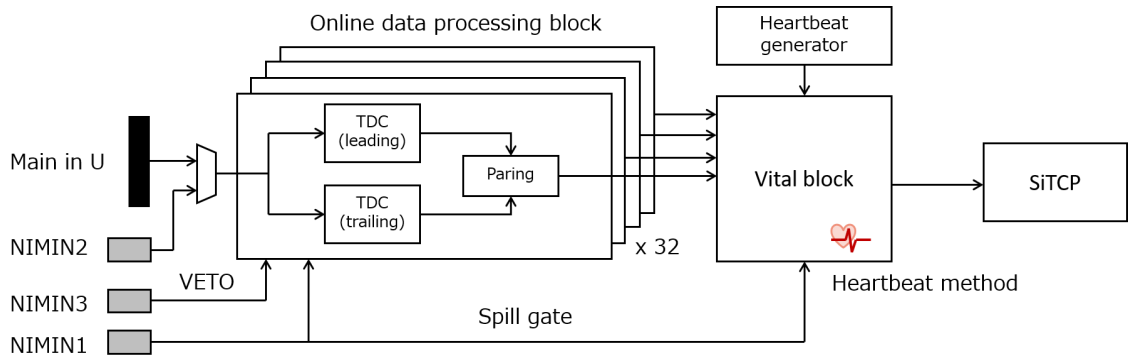

Figure 19: Streaming TDCのブロック図。

Streaming TDC仕様	
TDC精度	0.97... ns
最長スピルゲート長	33秒
最小パルス幅	~4 ns
最大パルス幅	150 ns
ダブルヒット分解能	~7 ns

4.11.1 レジスタマップ

TOT filerとtime-walk correctorはこのUGの中では説明していません。もし利用する場合は参考文献を読んでください。

レジスタ名	アドレス	読み書き	ビット幅	備考
DAQ Controller: DCT (module ID = 0x0)				
SelectTrigger	0x0000'0000	R/W	12	DAQデータ取得を開始するためのゲート。
Online Data Processing block: ODP (module ID = 0x1)				
EnFilter	0x1000'0000	R/W	1	TOT filterの機能を有効にします。
MinTh	0x1010'0000	R/W	8	TOT filterの下限閾値を設定します。この値よりTOTが小さいとフィルターされます。
MaxTh	0x1020'0000	R/W	8	TOT filterの上限閾値を設定します。この値よりTOTが大きいとフィルターされます。
EnZeroThrough	0x1030'0000	R/W	1	TOT値が0だったヒット（立下りエッジが見つからなかったヒット）をフィルターしないようにします。
TwCorr0	0x1040'0000	R/W	8	Time-walk correctorの領域0の補正値を設定します。
TwCorr1	0x1050'0000	R/W	8	Time-walk correctorの領域1の補正値を設定します。
TwCorr2	0x1060'0000	R/W	8	Time-walk correctorの領域2の補正値を設定します。
TwCorr3	0x1070'0000	R/W	8	Time-walk correctorの領域3の補正値を設定します。
TwCorr4	0x1080'0000	R/W	8	Time-walk correctorの領域4の補正値を設定します。

4.11.2 HUL上のスイッチ・LEDの機能

DIP SW2の機能

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	Enable test in	ONにするとNIMIN2が全チャンネルの入力に接続されます。
3	MCD mode bit-1	MCDメザニンを搭載した際の動作を決めるビットです。通常はOFF (0)に設定します。 • 0b11: Master • 0b10: Repeater • 0b01: Slave • 0b00: Stand alone
4	MCD mode bit-2	MCDメザニンを搭載した際の動作を決めるビットです。通常はOFF (0)に設定します。
5	Enable signal copy	このビットがONだと、下側のメザニンコネクタへ入力信号のコピーが出力されます。別の回路でも信号を使いたい時に利用します。DTLメザニンカードを下側のメザニンスロットへ取り付けてください。
6	Not in Use	
7	Not in Use	
8	Not in Use	

LED点灯の機能

LED番号	備考
LED1	点灯中はTCP接続が張られています。
LED2	機能していません。
LED3	点灯中はスピルゲート入力がHIGHであることを示します。
LED4	機能していません。

NIM-IOへの信号アサイン

Streaming TDCにはIOMが存在しないため、入出力のアサインを変える事はできません。

NIMIO	備考
	NIM-IN
NIN-IN1	スピルゲート入力。
NIM-IN2	テスト入力。
NIM-IN3	VETO入力。
NIM-IN4	未使用
	NIM-OUT
NIM-OUT1	未使用
NIM-OUT2	未使用
NIM-OUT3	NIM-IN1に入力されたスピルゲートのコピー出力。クロック同期を取った後の信号のため立ち上がりタイミングは量子化されている。
NIM-OUT4	未使用

4.11.3 DAQの動作

Streaming TDCはtirgger-less DAQで使用することを想定しているため、データ取得のトリガーという概念は存在しません。 データ転送を行う条件が揃うとFPGA即座にネットワークにデータを送信しようと試みるため、PC側は先にデータ準備が出来ていないといけません。 FPGAがデータを出力する条件は次の3つが全て成立している事です。 1. TCP connectionが成立している。 2. DAQ gateがONになっている。 3. Spill gateがON (NIMIN1の信号がHIGHである)。 本FWを使用する際には、上記順番の通りに処理することをお勧めします。 DAQ gateがONでspill gateがONになるとODPブロックはデータ計測を開始します。 この時点でTCP接続が確立していないと内部バッファにデータが溜まり続けていくため、バッファがあふれてしまう可能性があります。 TCP接続を確立してからRBCPでDAQ gateをONにしてください。

データ構造

Streaming TDCの1ワードは40-bitです。受け取ったデータをそのまま `int32_t` や `int64_t` にキャストできないので、ソフトウェアの記述は工夫してください。 本FWにはspill start, TDC data, heartbeat, busy, spill endの5種類のデータが存在します。 それぞれ先頭の4ビットで識別が可能です。 Heartbeatデータはheartbeat (time)

frameの境目に挿入され、spill startから何回目のheartbeatであるかを示すカウンターが下位16-bitに埋め込まれています。 もしvital blockの状態がbusyモードの場合にはheartbeatデータの代わりにbusyデータが返ってきます。

```
Spill start word
MSB                                                                         LSB
|      0x1      |           RSV (20-bit)              |                0xFFFF               |

TDC data
|      0xD      | '0' | TOT(8-bit) | Type(2-bit) | Ch(6-bit) |           TDC(19-bit)          |

Heartbeat data
|      0xF      |           RSV (20-bit)              |         heartbeat number (16-bit)      |
Busy data
|      0xE      |           RSV (20-bit)              |         heartbeat number (16-bit)      |

Spill end data
|      0x4      |           RSV (20-bit)              |                0xFFFF               |
```

TDC dataに含まれるtypeは現状きちんと機能していません。 通常のTDC dataであれば `00` です。そのほかに `01` と `10` のパターンがありますが、これらであった場合そのデータは無視してください。

Busyモード

FPGAが送信しようとするデータ量がデータリンクのスピードを超えると転送が追い付かなくなり、FPGA内部のバッファが溢れます。 このような状況にFWが陥ると、vital blockは通常の動作モードからbusyモードへ状態を遷移させ復帰を試みます。 Streaming TDCではtriggerを止めるためのBUSY信号は存在しませんが、busyモードではbusyデータがheatbeatデータの代わりに現れるようになります。 Busyデータが返ってきたheartbeatフレームではデータ欠落が起きていますが、どのデータをどれだけ落としたかは分かりません。 一部または全てのデータが欠落しているということだけが分かります。 一旦busyモードに移行すると最低3フレーム復帰するために必要とします。 更に詳しい動作については参考文献を参照してください。

データの並び

Streaming TDCから出力されるデータの並びを時間軸上に表した物を図に示します。 Spill gateの立ち上がりを検出するとまずspill start dataが出力されます。 次に最初のheatbeat frame内のTDC dataが続き、frameの境目にheartbeat (busy) dataが挿入されます。 Busy dataが返ってきた場合該当frameではデータ欠落が起きています。

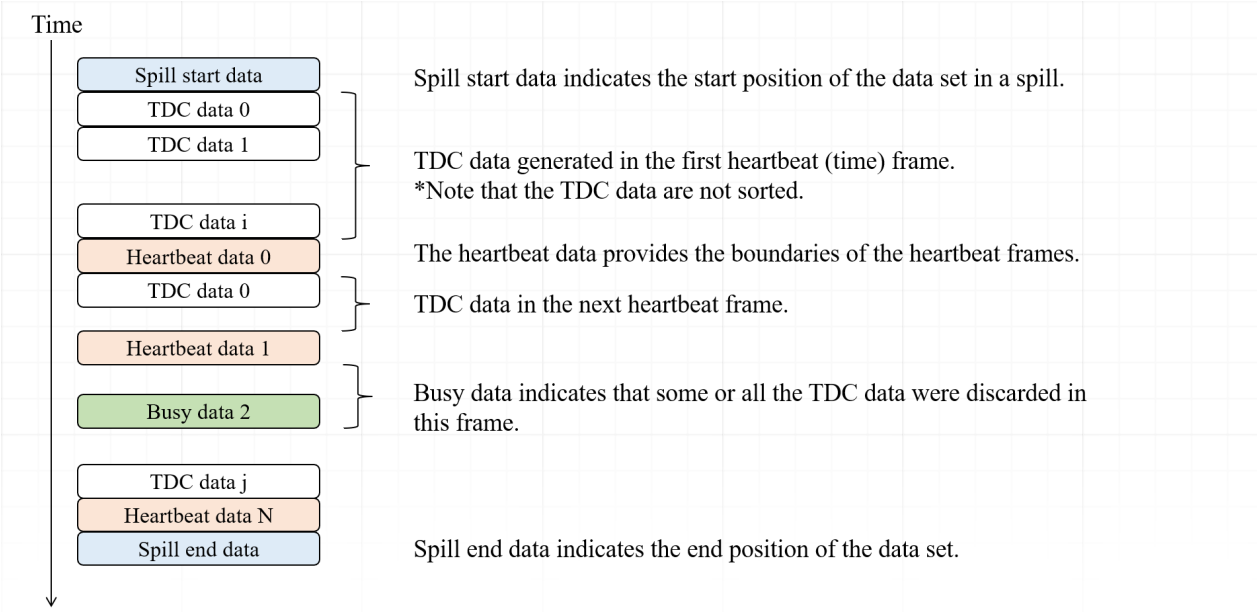

Figure 20: Streaming TDCから出力されるデータの並び順

5. For developer

この章ではFPGA firmware（以下FW）開発者向けの情報を説明します。

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。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。

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 MH-TDC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、安いラップトップで済ませようというのはお勧めしません。

5.1 Vivado projectに関して

要素	コメント
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
合成のストラテジー	Vivado synthesize Default

5.1.1 命名規則

Gitlab上のhul-officialにFWを上げる可能性がある場合，著者の命名規則に揃えるようお願いいたします。

5.1.2 SiTCPやIP catalogを使って生成したIPについて

SiTCPは株式会社Bee Beans Technologiesから提供されているライブラリです。詳しい情報はここから入手してください。

SiTCPは企業から提供されているライブラリであるためgitlab上のFWには同梱されていません。Cloneした後以下のようにSiTCPというディレクトリを生成し、その中にライブラリファイルを追加してください。HULのプロジェクトはSiTCPというディレクトリを見に行くように設定されているので、VivadoでXPRファイルを開く前にこの作業をすますとエラーが出ずに済みます。SiTCPディレクトリは.gitignoreによって除外されています。何かの手違いでSiTCPライブラリがGit管理下に入ってしまったFWはhul-officialへはmerge出来ませんので注意してください。

```
HUL_Skelton.git
├── HUL_Skelton.cache
├── HUL_Skelton.hw
├── HUL_Skelton.ip_user_files
├── HUL_Skelton.runs
├── HUL_Skelton.sim
├── HUL_Skelton.srcs
└── SiTCP
```

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が、IP生成物のアーカイブであるためファイルサイズが大きいです。このままだとGitリポジトリを圧迫するので、HULのFWではcore containerは利用しないでください。

5.2 HUL firmwareの内部構造について

5.2.1 Top level entity ports

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

信号名	信号規格	コメント
CLKOSC	LVCMOS33	基板上の発振器からの50 MHzクロック入力です。
PROB_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
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
MAIN_IN_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。
MAIN_IN_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。
		メザニンスロット
MZN_SIG_Up/n	VCCO=1.8Vまでの差動・単一端信号	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ります。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、双方向ポートを使うのか、浮き配線の処理などはファームウェア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・単一端信号	同上です。
		J0バス信号
J0RS	LVCMOS33	J0バスのS1-7番を読み出しポートです。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MTM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。

信号名	信号規格	コメント
J0DC	LVCMOS33	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。

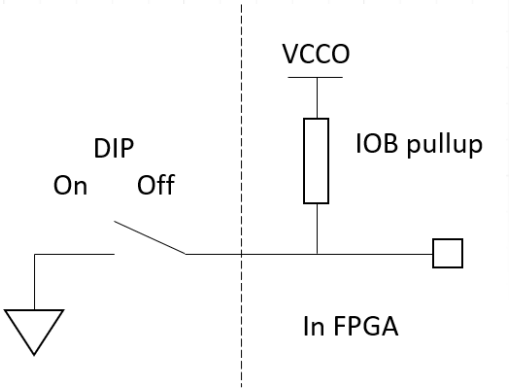

Figure 1: Circuit diagram of DIP SW

5.2.2 SiTCP

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。

5.2.3 Local bus controller

Local bus controller (BCT) はRCNPの味村さんが書いたソースコードを著者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とソフトウェアのFPGAModuleクラスは対の関係になっており、著者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お勧めします。

BCTのバス通信シーケンスを下図に示します。BCTがバス通信サイクルを始める起点はexternal bus (SiTCP RBCP) 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local busのモジュールIDおよびローカルアドレスの関係は下図の通りです。ソフト側からはRBCPのアドレスが再配置されている事は分からないようになっています。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PROB_B_ONの駆動やリセットの発行を行ったり、FW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

ジュールは自分の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。

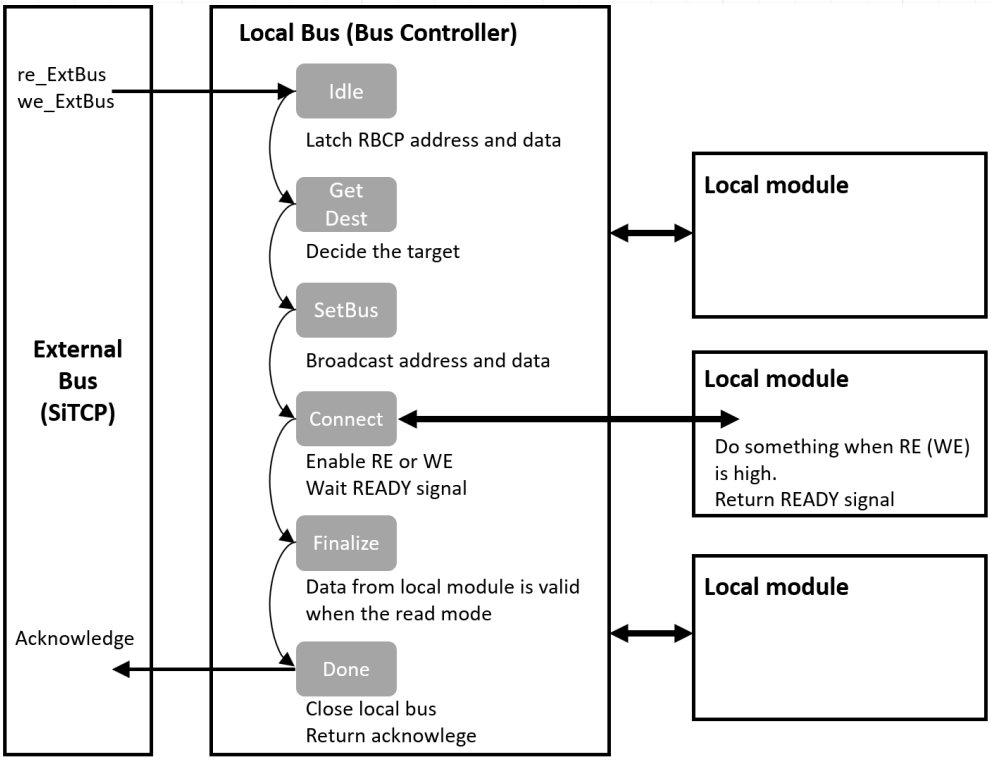

Figure 2: BCTの動作フローチャート

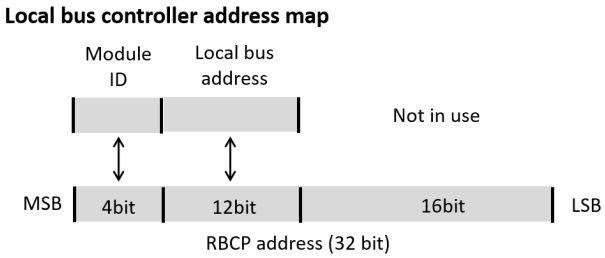

Figure 3: BCTのアドレスマップ

Local bus addressと多バイト読み出し（書き込み）

Local busを介した読み出し場合8-bit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。HULでは12ビットのローカルアドレスのうち上位8ビットをレジスタをIDするために利用し、下位4ビット何バイト目かを指すために利用します。例えば、モジュールインデックスが0x8のローカルモジュール内に、16ビット幅のレジスタAとBが存在する場合BCTとRBCPのアドレスは以下の様になります。

BCT address	RBCP address	コメント
0x000	0x8000'0000	レジスタAの1バイト目（下位8ビット）
0x001	0x8001'0000	レジスタAの2バイト目（上位8ビット）
0x010	0x8010'0000	レジスタBの1バイト目（下位8ビット）
0x011	0x8011'0000	レジスタBの2バイト目（上位8ビット）

多バイト読み出しをさせる場合のローカルモジュール側のHDLの例を以下に示します。

```
when kReadLength(kNonMultiByte'range) =>
  if( addrLocalBus(kMultiByte'range) = k1stbyte) then
    reg_length_read(7 downto 0) <= dataLocalBusIn;
  elsif( addrLocalBus(kMultiByte'range) = k2ndbyte) then
    reg_length_read(kWidthRead-1 downto 8) <= dataLocalBusIn(kWidthRead-1-8 downto 0);
  else
end if;
```

`hul_software` に同梱されている `FPGAModule` クラスでは多バイト読み書きをサポートしており、BCTアドレスの下位4ビットは自動でインクリメントされます。

5.2.4 Local Busへモジュールを追加する

ローカルモジュールを追加する場合、次の点を変更してください。

defBus.vhd内部の変数

項目	コメント
kNumModules	BCTを除いたローカルモジュール数を指定
Module ID	kMidから始まるIDを追加。 RBCP addressで指定する番地となる。連番である必要はない。
Leaf	Local busのインデックスを決めるためのIDを追加。 連番である必要があります。

BusController.vhd内部

ステートマシンでGetDest状態のelse構文以下で、追加したモジュールのwhen構文を追加。

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新しいLeaf IDを指定。

6. Software

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

- CentOS7
- gcc version 4.8.5

HR-TDC用のソースコードのみROOTファイルを作るためにCERN ROOTに依存します。 ROOTがインストールできない環境ではHR-TDCをmake対象から外してください。

ディレクトリ構造

CoreLibはこのソフトウェアり利用される共通のライブラリを提供します。CommonはSDSやFMPの制御など、各ファームウェアで共通の要素を制御するためのプログラムを提供します。そのほか各ファームウェア専用のソースコードはFW名がついたディレクトリ内に存在します。

```
hul_software.git
├── Makefile
├── CoreLib
├── Common
└── FW directories
```

6.1 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、デバッグ用で移植の必要がない物が存在します。重要な物だけ説明します。
全ての実行体は引数無しで起動するとusageが表示されるようになっています。

CoreLib

rbcp.hh
RBCPパケットの構造体です。そのまま移植してください。

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。かなり古い実装方法で今はc++の標準機能で同様の事が出来るためそのうちなくします。

BitDump
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。

Utility
コンソール上でプログレスバーを表示したり文字を点滅させたりします。移植の必要はないのですがFlashMemoryProgrammerの中で利用しています。

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を必要とし、最後の引数は表示モードです。`UDPRBCP.hh`内に`RbcpDebugMode`という列挙体で定義されていま

す。

- NoDisp: 通信時に何も表示しません。
- Interactive: 何をしている程度の情報を出力します。
- Debug: 全ての情報を標準出力します。

通常使う際はNoDisp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ます。最低限これがあればHULの制御が可能です。

FPGAModule

UDPRBCPを隠蔽してBusControllerレベルの通信機能を提供するクラスです。UDPRBCPを包含するためコンストラクタはUDPRBCPを引数に取ります。BCTを通じて多バイトの読み書きをサポートすることが主な機能です。

```
int32_t WriteModule(const uint32_t local_address,
                    const uint32_t write_data,
                    const int32_t  n_cycle = 1);
```

レジスタを書き込むメソッドです。第1引数はRBCP addressです。4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2引数が書き込むレジスタです。第3引数は書き込みバイト数です。戻り値は `UDCPRBCP::DoRBCP` の結果です。この関数は32-bitと64-bitレジスタ用にオーバーロードされています。`n_cycle` が取れる範囲はそれぞれ1-4と1-8です。

```
uint32_t ReadModule(const uint32_t local_address,
                    const int32_t  n_cycle = 1);
```

レジスタを読み出すためのメソッドです。第1引数はRBCP address。第2引数は多バイト読み出しの回数を設定します。読み出した結果は関数の戻り値に現れます。`n_cycle` の値に限らず `uint32_t` の型で与えられます。この関数も32-bitと64-bit用にオーバーロードされています。

```
int32_t WriteModule_nByte(const uint32_t  local_address,
                          const uint32_t* write_data,
                          const int32_t   n_byte);
```

同一のアドレスに複数バイト書き込むための関数です。この関数では書き込みのたびにアドレスの最下位をインクリメントしません。想定する書き込み先はFIFOです。

```
int32_t ReadModule_nByte(const uint32_t  local_address,
                         const int32_t   n_byte);
```

同一のアドレスから複数バイト読み出すための関数です。この関数では読み出しのたびにアドレスの最下位をインクリメントしません。想定する読み出し元はFIFOです。読み出したデータは `rd_data_` というクラス変数に格納されます。`GetDataIterator` でイテレータを取得してアクセスしてください。

SiTCPController

SiTCPの予約領域に直接アクセスするための関数です。EraseEEPROM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 ResetSiTCP(const std::string ip)
```

SiTCPをソフトリセットします。

```
void WriteSiTCP(const std::string ip,
                const uint32_t addr_ofs,
                const uint32_t reg)
```

SiTCP予約領域アドレスに1-byte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 ReadSiTCP(const std::string ip,
                   const uint32_t addr_ofs)
```

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 EraseEEPROM(const std::string ip)
```

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。

Common

FlashMemoryProgrammerとSelf Diagnosis Systemの制御用のプログラムがまとめられています。これらはHULのFWの共通部分のため、レジスタは`RegisterMapCommon.hh`としてまとめられています。HUL制御の例題集のような側面があるので、各main関数の動きを説明します。

assert_bctreset

`BCT::Reset`にアクセスしFW内でリセット信号をアサートさせます。

erase_eeprom

SiTCPControllerのEraseEEPROMを呼び出しす実行体です。

reconfig_fpga

能動的にFPGAをSPIフラッシュメモリからリコンフィグする実行体です。FPGAは電源投入時にSPIフラッシュメモリからコンフィグされますが、それを任意のタイミングで行うためのプログラムです。FPGA放射線ダメージを受けた、新しいMCSをFMPを使ってSPIフラッシュメモリへダウンロードした、などのタイミングで使います。

read_xadc

SDS内のXADCインターフェースにアクセスして、FPGAの温度、VCCINT電圧、VCCAUX電圧を取得します。

read_sem

SDS内のSEMインターフェースにアクセスして、訂正可能なSEUを修正した回数、SEMの動作状態 (watchdog)、修正不可能な放射線損傷を検知したかどうか (uncorrectable)を読み出します。SEUを訂正した回数はBCTリセット以上のリセット信号か`SemRstCorCount`にアクセスで0に戻ります。Uncorrectableに1が立っている場合FPGAの再コンフィグが必要です。

reset_sem

SEMのソフトリセットをアサートします。

inject_sem_error

SEMを使って人工的にSEUを発生させます。デバッグ用です。DRPを使ってSEMにアクセスしているので、理解すれば他の動作もさせる事が出来ます。詳しくはXilinxのPG036を参照してください。

flash_memory_programmer

MCSファイルをSiTCPを通じてSPIフラッシュメモリへ書き込むためのプログラムです。MCSファイルをバイナリに変換（MCSは実はテキストファイルです）、SPIメモリの型番のチェック、メモリイレース、プログラム、リードバックとベリファイを順番に実行します。

mcs_converter

MCSファイルをあらかじめバイナリに変換しておくためのプログラムです。

check_spi_device

HUL上にのっているSPIフラッシュメモリの型番を調べます。何が乗っているか分からないときに使ってください。

verify_mcs

SPIフラッシュメモリからリードバックして指定のMCSファイルとの整合性をチェックします。

6.2 各FW用のプログラム

殆どのFWでは`debug`と`daq`の実行体だけが実装されています。`daq`はデータ取得するための実行体です。DAQコードのサンプルとして使ってください。`RegisterMap.hh`にはFW特有のレジスタのアドレスが書かれています。ここではFW特有のプログラムについて説明します。

DaqFuncs.cc

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SetTdcWindow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、EventCycle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EventCycle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 != EvenCycle)`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。

daq

HULからデータ読み出しを実行します。 次の様にIP address、RUN番号、および取得するイベント数を引数に与えてください。`./bin/daq [ip address] [RUN no] [# of events]`. データファイルは`data/run1.dat`のようにコマンドが実行されたディレクトリのサブディレクトリ`data`内に生成されます。そのため、この実行体は各ファームウェアディレクトリの下で実行されることが想定されています。`data`ディレクトリが存在しない場合make実行時に生成されます。

データ収集中、読み出したデータが何イベントかに1回コンソール上に表示されます。

6.2.1 HR-TDC

initialize

DDR receiverの初期化、および校正クロックを使ってestimator用のLUTの校正を行います。電源投入後に1度実行するする必要があります。

decoder

デバッグ用にデコーダが用意されています。次のようにRUN番号を与えて実行してください`./bin/decoder [RUN no]`. このプログラムは`data`ディレクトリ内のデータファイルを読み、`rootfile`ディレクトリにデコード結果のroot fileを生成します。daqと同様に各ファームウェアディレクトリの下で実行されることが想定されています。

TTreeの構造

Branch name	Description
through	各メザニンHR-TDCのTDC::Throughの設定を示します。 インデックス0と1はスロットUとスロットDに対応します。
stop_out	各メザニンHR-TDCのTDC::StopDoutの設定を示します。 インデックス0と1はスロットUとスロットDに対応します。
over_flow	Sub-headersのOverFlow bitを示します。インデックス0と1はスロットUとスロットDに対応します。
Decoded common stop TDC data.	
stop_fine_count	Common stop dataのFine count (TDC dataの下位13 bit)です。TDC::StopDoutが1である場合利用可能なデータです。
stop_estimator	Common stop dataのestimator値です。TDC::StopDoutが1である場合利用可能なデータです。
stop_coarse_count	Common stop dataのcouarse count (TDCデータの上位11 bit)。TDC::StopDoutが1である場合利用可能なデータです。
common_stop	Common stopのTDC値。TDC::StopDoutが1である場合利用可能なデータです。
Decoded data for leading edge measurement.	
num_hit	リーディングエッジデータのヒット数です。
num_hit_ch	各チャンネルにおけるリーディングエッジデータのヒット数です。
channel	チャンネル番号です。この配列のインデックス番号はデコードされた順番を示します。
fine_count	リーディングエッジデータのfine count値です。TDC::Throughが1の時有効なデータです。この配列のインデックス番号はデコードされた順番を示します。
estimator	リーディングエッジデータのestimator値です。TDC::Throughが0の時有効なデータです。この配列のインデックス番号はデコードされた順番を示します。
coarse_count	リーディングエッジデータのcoarse count値です。この配列のインデックス番号はデコードされた順番を示します。
tdc_leading	リーディングエッジデータのTDC値です。この配列のインデックス番号はデコードされた順番を示します。
Decoded data for trailing edge measurement.	
tdc_trailing	トレーリングエッジデータのTDC値です。この配列のインデックス番号はデコードされた順番を示します。
ブランチ名がtから始まる物はトレーリングエッジデータのデコード結果を示します。	

7. Practical Usage

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。

7.1 Vivadoを使ったMCSの生成とダウンロード

MCSの生成

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。Vivadoでプロジェクトを開き、tool → Generate Memory Configuration Fileを選択します。出てきた画面を図様に埋めてOKを押すとMCSが生成されます。この画面はVivado 2020.1のものです。

HULには2021年現在3種類のSPIフラッシュメモリが使われています。それぞれVivadoでは以下のパーツを選択してください。

基板上の部品	メーカー	Vivadoで選択するパーツ
N25Q128A	Micron	mt25ql128-spi-x1_x2_x4
MT25QL512	Micron	mt25ql512-spi-x1_x2_x4
S25FL256S	Spansion	s25fl256sxxxxxx0-spi-x1_x2_x4

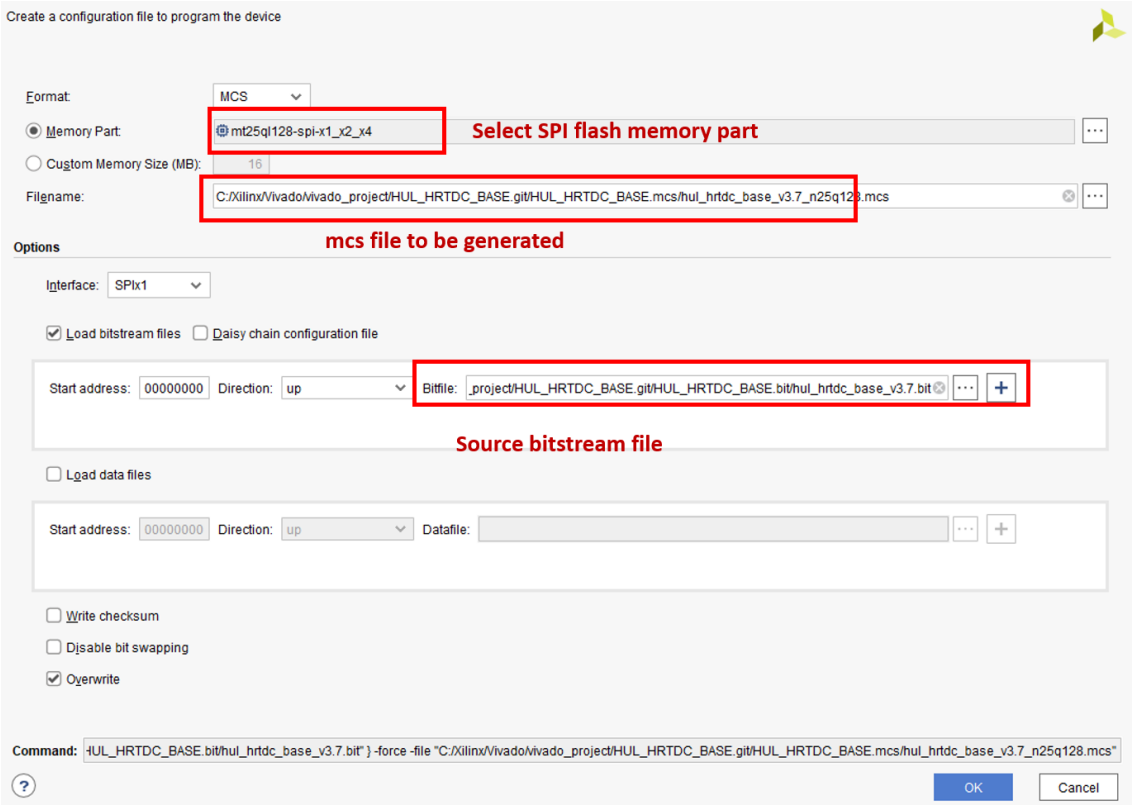

Figure 1: MCS生成画面

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OS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 (XC7K160T-1)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。MCSの場合SPIフラッシュメモリにダウンロードするのですが、この時点では画面上に出てきていません。これはSPIフラッシュメモリが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メモリを選択します。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。

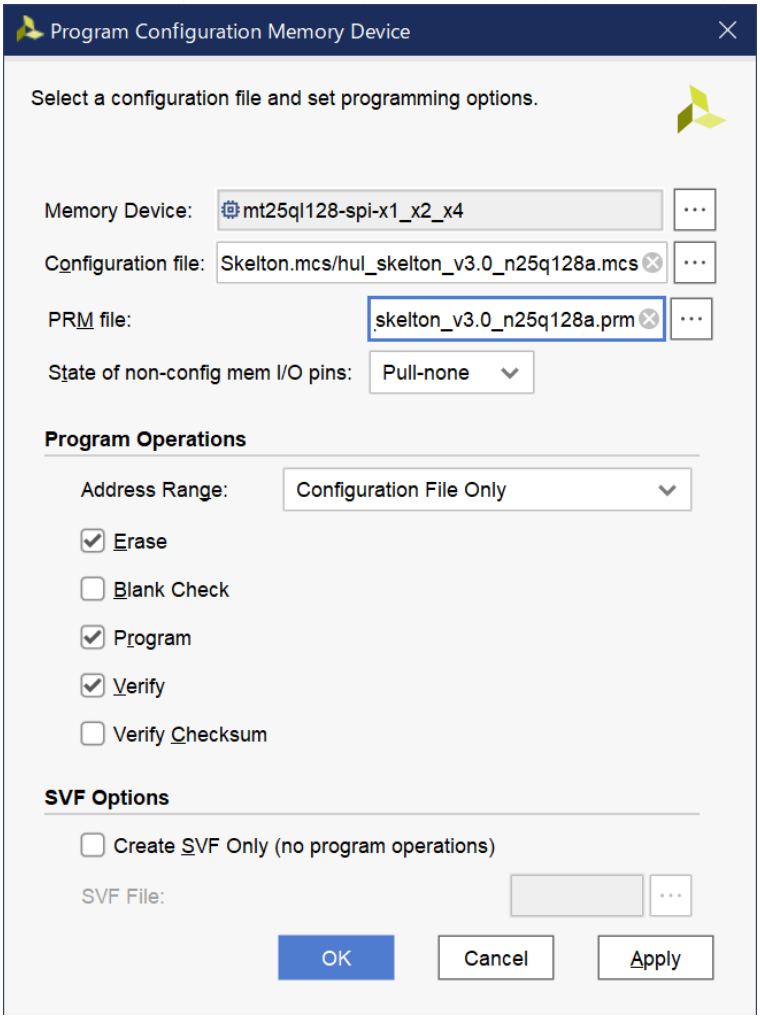

Figure 2: Configuration Memory Device追加画面

N25Q128Aの取り扱いについて

MicronのN25q128aはHULを設計した時に採用した石ですが、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年後期の共同購入時、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S25FL256SAGNFI0が搭載されています。ソースコードから合成してMCSファイルを作る場合、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s25fl256sxxxxxxx1-spi-x1_x2_x4へ変更してください。

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
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このトラブルに遭遇します。Hardware managerが立ち上がっていてもサーバをCloseしていれば回避できます。

7.1.1 Mezzanine HR-TDCへのMCSダウンロード

Hardwareの章で記述した通りmezzanine HR-TDCでは2021年現在2種類のflash memoryのパターンがあります。ダウンロードの際には以下のパーツを選択してください。

基板上の部品	メーカー	Vivadoで選択するパーツ
N25Q128A	Micron	mt25qu128-spi-x1_x2_x4
MT25QU256A	Micron	mt25qu256-spi-x1_x2_x4

7.2 FMP用いたSiTCP経由のMCSダウンロード

Flash Memory Programmerを使う事でネットワーク（RBCP）経由でSPIフラッシュメモリへMCSをダウンロード出来ます。JTAGを利用する場合USBダウンロードケーブルを基板に挿す必要があるため、作業者が1つ1つ基板の近くで作業する必要があります。実験中にFWを入れ替えようと思った場合大がかりな作業となります。FMPを利用するとモジュールに触らずして計測室からMCSのダウンロード、およびFPGAの再コンフィグが可能となり、リモートでメンテナンスが出来ます。

MCSファイルはJTAGで書き込む場合と同じものを利用してください。SPIフラッシュメモリに書き込むための実行体は`hul_software.git/Common/bin`にあります。以下のように実行してください。もし、`mcs_converter`で変換済みのバイナリファイルがある場合は`-b`オプションを付けて実行してください。

```
./bin/flash_memory_progorammer [ip-address] [mcs file path]
```

ダウンロードが成功し、ベリファイの結果が元のMCSと一致すると以下のような表示が出ます。もし、途中で通信が途絶してしまったり、mismatchと表示された場合は成功するまでリトライしてください。この段階ではSPIフラッシュに書いただけでFPGAには反映されていません。正常に書き込めた後`reconfig_fpga`を使ってFPGAを再コンフィグしてください。

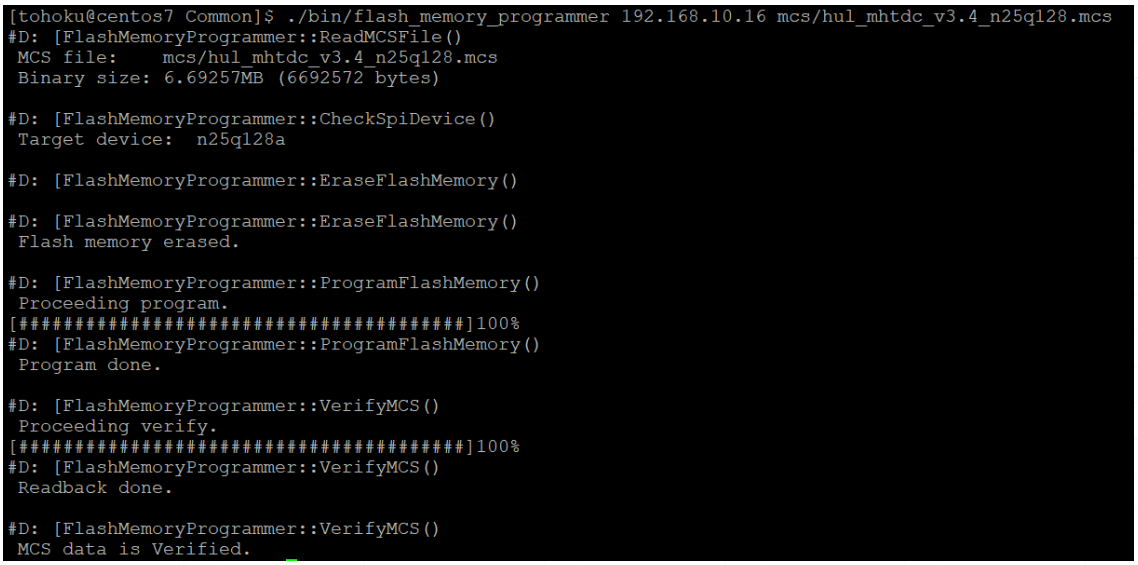

Figure 3: FMPを通じてMCSをダウンロードした際のコンソール表示。図は書き込みが成功した例。

既知の問題点

フラッシュメモリのページ書き込みにはそれなりの時間がかかり、PCのマシンスペックや通信環境次第ではUDP RBCPが1サイクル回るまでの間にページ書き込みが終わりません。現在のFMPはページ書き込み中に次の書き込み動作をブロックするように設計されておらず、次のページ書き込みのリクエストが早すぎると通信が途絶したりUDP RBCPのバスエラーが発生したりします。現状 `usleep` で強制的に次の書き込みを遅らせていますが、良い解決方法ではないし書き込みかかる時間も長くなります。将来的にはFPGA側で対応できるように改良したいですが、まだめどはたっていません。

7.3 使用時のハウツーなど

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。 `daq_func.cc` で `TRM::SelectTrig` に設定するレジスタを `TRM::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。

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。（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::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::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
モジュールリセット (BCTリセット等) をかけた後DAQが回らないという現象が既知の問題として知られていましたが、version 3系で解決したと思われます。同様の問題が発生したら報告してください。

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mt25qu128-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 `EnSlotUp/EnSlotDown` をfalseにしてください。
4. `./bin/debug` を実行します。HUL本体のバージョン、メザニンのバージョンが表示されます。ここでバスエラーが出る場合何かがおかしいです。まずソフトを確認し、間違いがなければメザニンとの接続が疑われるので一度付けなおしてください。
5. `./bin/initialize` 実行しDDRレシーバの初期化、校正クロックによる校正とestimatorの生成を行ってください。両方が成功していることを確認してください。
6. 配布したソースコードはNIM-IN1にコメンストップを入力することを想定しています。 NIM-IN1へトリガーを接続して、 `./bin/daq [run no] [# of events]` を実行してください。。 データファイルは `./data/run[run no].dat.` の様に生成されます。
7. `./bin/decode [run no]` 実行しデコードの実効とROOT fileの生成を行います。ROOT fileは `./rootfile/run[run no].root` のように生成されます。

7.4 その他

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購入するか伝えてください。在庫があれば即納してもらえます。HUL controllerではればSiTCPライセンスは価格に含まれています。 在庫がない場合どのように製造するかはその都度相談になります。