Data Organizer

Yangyang Cheng
03/21/2013
Hardware System Overview

- Pixels & SCT
- RODs
- Data Formatter (DF)
- Cluster finding
- Split by layer
- Overlap regions
- HITS
- 50~100 KHz event rate
- S-links
- Raw data ROBs
- Core Crate
- Second stage
- AM brd
- DO
- TF
- HW
- Track data ROB
- ~Offline quality Track parameters
- Track parameters
- ~Offline quality Track parameters

---

2
AUX card & Data Organizer

AUX Responsibilities

• Receive hits from Data Formatter
• Organize hits into Super Strips (SS)
• Send SS to AM Board
• Receive IDs for matched roads from AM board
• Reject bad roads
• Send hits from promising roads to Second Stage board

Data Organizer: A smart database built on the fly so that Hits can be immediately stored/retrieved according to its SSID.
Input/Output

Transceiver

Data Organizer

TrackFitter

- rxchan(10..0)(31..0)
- roadchan(31..0)
- AMMAP (external memory)
- wr_ovflw
- rd_ovflw
- empty
- SSlast
- EE
- layermap(7..0)

- out_pixhit(2..0)(31..0)
- out_scchit(4..0)(15..0)
- out_roadID(31..0)
- out_sectorID(15..0)
Inside the DO Block

Data Organizer

Hit X 11L
(pix 3*2L; sct 5L)

SSID X 11L

RoadID

AMMAPInt
(ext.mem. interface)

SSID +
DCFlag X 11L

DO1 (WR/RD)

error flags

FSM

Hit X 8L
(pix 3L; sct 5L)

RoadID

SectorID

error flags

DO2 (WR/RD)
WRITE/READ mode

• Write Mode:
  – Stores incoming hits according to its SSID

• Read Mode:
  – Sends out pre-stored hits corresponding to incoming RoadID; RoadID is translated into SSID by AMMap; output hits grouped by SSID

• DC (Don’t Care)
  – up to 3 LSB of SSID is ignored in comparison/matching

• Hit information w.r.t. corresponding SSID is stored/retrieved in 3 separate parts
  – **HCM**(HitCountMem): stores HitCnt (the number of hits with specific SSID)
  – **HLP**(HitListPointer): stores the address for the 1\(^{st}\) hit with specific SSID (SSID used as HLP address) in the HLM
  – **HLM**(HitListMem): stores hits sequentially; hits with same SSID stored consecutively (hits with same SSID will be coming consecutively on input stream)
Parallel Structure

DO1(2)

Hit
SSID
(from transceiver)

SSID
(from AMMap: AMMap reads one input stream and sends output on 11 parallel streams)

The above structure duplicated for 11 parallel layers
- 3*2 pixel layers (3 layers in final output; doubled w.r.t. even/odd SSID)
- 5 SCT layers
Write Mode (single layer)

14bit SSID per layer
14bit SSID

32bit pixhit/16bit scthit per layer
32bit pixhit/16bit scthit

fullSSID: 14bit SSID
AMSSID: 11MSB of SSID

HCM(0)
HCM(1)
......
HCM(7)

HCM_wren(7..0)
HCM_wraddr(7..0)(10..0)
HCM_data(7..0)(4..0)/(2..0)

HitCountMem
Content(5bit for pix; 3bit for sct): HitCount for the fullSSID
Addr. (11bit): AMSSID

HLP
HLP_wren
HLP_wraddr(13..0)
HLP_data(9..0)

HitListPointer
Content (10bit): starting addr. for 1st hit in fullSSID
Addr. (14bit): fullSSID

HitListMem
Content (32bit for pix; 16bit for sct): pixhit/scthit
Addr. (10bit): default sequential (starting addr. stored in HLP)

MLADD
MLDATA
HLCTRL_WR
HLM
HLM_wren
HLM_wraddr(9..0)
HLM_data(31..0)/(15..0)

8
**Read Mode (single layer)**

MLRDADD

- 14bit SSID + 2bit DCFlag per layer
- fullSSID: 14bit SSID
- AMSSID: 11MSB of SSID

HCM_rden(7..0)
HCM_rdaddr(7..0)(10..0)

HCM_rden(7..0)
HCM_rdaddr(7..0)(10..0)

HCM(0)
HCM(1)
....
HCM(7)

**HitCountMem**
Content (5bit for pix; 3bit for sct): HitCount for the fullSSID
Addr. (11bit): AMSSID

**HitListPointer**
Content (10bit): starting addr. for 1st hit in fullSSID
Addr. (14bit): fullSSID

HLCTRLE_RD

HCM_q(7..0)(4..0)/(2..0)

**HitListMem**
Content (32bit for pix; 16bit for sct): pixhit/scthit
Addr. (10bit): default sequential (starting addr. stored in HLP)

MLRDDATA

- RoadID
- SectorID
- LayerMap

MLSectorID

MLLayerMap

HLP

HLP_rden
HLP_rdaddr(13..0)

HLP_q(9..0)

HLM

HLM_rden
HLM_rdaddr(9..0)

HLM_q(31..0)/(15..0)

HitCountMem

Content (32bit for pix; 16bit for sct): HitCount for the fullSSID
Addr. (11bit): AMSSID

HitListPointer
Content (10bit): starting addr. for 1st hit in fullSSID
Addr. (14bit): fullSSID

rd_ovflw
empty
SSLast
EE
Finite State Machine

- type FSMSTATES is (  
  - state_write,  
    - writing in new hits  
    - switch to WRready when hitEE=1 i.e. end of event, last hit in event  
  - state_WRready,  
    - wait for the other DO to finishing reading  
    - switch to READ when the other DO is in WRITE mode  
  - state_read,  
    - reading out hits  
    - switch to Rdready when last=1 or empty=1 i.e. all hits in that SSID read out  
  - state_RDready,  
    - wait for the other DO to finish writing  
    - switch to WRITE when the other DO is in READ mode  
  );

- DO1 start in state_write (DO1fsm); DO2 start in state_Rdready (DO2fsm)
AMMapInt (ext. mem. interface)

• using RLDRAMII: MT49H16M36-25
• interface generated using QuartusII v12.1 for Arria V
  – studied through auto-gen. testbench/example
  – using the same pin assignments as example
• AMMap stored on ext. mem.
  – translate input RoadID into SSID+DCFlag per layer
• Multiple compilation warnings/timing issues
  – detailed in separate doc.
VHDL Code Index

• Dotop
  – DO_deflib (defines the signal types used)
  – fsmttop
    • DO1fsm; DO2fsm
  – AMMapInt
  – DO11L
    • MLPIX(SCT)DATA, MLADD
    • HLCWritePIX(SCT)
    • MLRDADD, MLRDDATA, MLSectorID, MLLayerMap
    • HLCReadPIX(SCT)
    • HCM, HLP, HLM (PIX/SCT)
## Resource Usage: Memory

<table>
<thead>
<tr>
<th>RAM name</th>
<th>Width</th>
<th>Depth</th>
<th>Size (M10K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLM_pix</td>
<td>32</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>HLM_sct</td>
<td>16</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>HLP_pix</td>
<td>10</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>HLP_sct</td>
<td>10</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>HCM_pix (*8)</td>
<td>5</td>
<td>11</td>
<td>1*8</td>
</tr>
<tr>
<td>HCM_sct (*8)</td>
<td>3</td>
<td>11</td>
<td>1*8</td>
</tr>
</tbody>
</table>

Total: 5.6Mb/25Mb needed; 7Mb/25Mb implemented
Usage expected/needed/implemented detailed in separate doc.
## Resource Usage: Logic

<table>
<thead>
<tr>
<th></th>
<th>LC Comb.</th>
<th>LC Registers</th>
<th>ALMs needed</th>
<th>ALMs placed</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLCRD_pix</td>
<td>81</td>
<td>370</td>
<td>135.11</td>
<td>205.00</td>
</tr>
<tr>
<td>HLCRD_sct</td>
<td>71</td>
<td>347</td>
<td>116.73</td>
<td>190.17</td>
</tr>
<tr>
<td>HLCWR_pix</td>
<td>2898</td>
<td>2382</td>
<td>1857.50</td>
<td>1937.83</td>
</tr>
<tr>
<td>HLCWR_sct</td>
<td>2898</td>
<td>2332</td>
<td>1840.00</td>
<td>1890.33</td>
</tr>
<tr>
<td>AMMap_int</td>
<td>1027</td>
<td>1534</td>
<td>757.05</td>
<td>963.75</td>
</tr>
<tr>
<td>FSM</td>
<td>141</td>
<td>132</td>
<td>53.92</td>
<td>80.50</td>
</tr>
<tr>
<td>DO1(DO2)</td>
<td>33022</td>
<td>37003</td>
<td>23749.53</td>
<td>26853.27</td>
</tr>
<tr>
<td>DOtop</td>
<td>67214</td>
<td>77033</td>
<td>49034</td>
<td>55366</td>
</tr>
</tbody>
</table>

Total: 49,034/190,240 needed; 55366/190,240 implemented
Usage expected/needed/implemented detailed in separate doc.
HLCtrl_WR

- **SSID(13..0)** -> **AMSSID(10..0)** + **DCBits(2..0)**
  - **DCBits(2..0)** -> int DCIndex range 0 to 7
  - **AMSSID(10..0)** -> int AMSS_index range 0 to 2^11

- **fullSSID**
  - identify new SSID
  - count # of hits in old SSID : HitCnt
  - obtain starting addr. for each SSID by consecutive incrementation

- **DC_Reset**
  - 2^11 bits, each bit is a Reset flag for each HCM address respectively
  - Entire DCReset word set to “0” at beginning of event
  - Corresponding bit set to “1” when the content in the HCM with respective addr. is cleared
  - Only clear the corresponding HCM when its bit inside the DCReset word is “0”
    - So each HCM content could only be reset max once per event.
HLCtrl_WR (cont’d)

• AMSSID
  – incoming AMSSID(10..0) compared with previous AMSSID, simultaneously as fullSSID(13..0) is being processed (as described in previous slide)
    • Same
      – Write current HitCnt to HCM(DC_Index)
    • Different
      – Check DC_Reset(AMSS_Index)
        » DC_Reset(AMSS_Index)=0
          • Reset all 8 HCM at addr=AMSSID
          • Write HitCnt (=1) to HCM(DC_Index)
        » DC_Reset(AMSS_index)=1
          • Write HitCnt (=1) to HCM(DC_Index)
**HLCtrl_RD**

- **DCFlag**, 2 bit word
  - DCCnt= 0, 1, 2, 3 (No. of DCBits used)
  - DCCombNum = 1, 2, 4, 8 (No. of possible SSIDs)
  - DCComb(2..0): possible 3LSB of SSID with DC

- For i range 0 to DCCombNum-1 LOOP
  - If DCFlag = “00”, DCCombNum=1
    - tempDCCComb(2..0)= SSID(2..0)
  - If DCFlag = “01”, DCCombNum=2
    - tempDCCComb(2..0)= SSID(2..1) + i (1 LSB)
  - If DCFlag = “10”, DCCombNum=4
    - tempDCCComb(2..0)=SSID(2) + i (2 LSB)
  - If DCFlag = “11”, DCCombNum=8
    - tempDCCComb(2..0)= i (3 LSB)
  - tempDCIndex = to_integer(tempDCCComb)
  - DCIndex(tempDCIndex)=1, others=0
    - **the possible DCCComb are stored in DCIndex(7..0)**
HLCtrl_RD (cont’d)

• For i range 0 to 7 LOOP
  – if DCIndex(i) = 0
    • not valid DCComb, check next DCComb, i = i+1
  – if DCIndex(i) = 1
    • valid DCComb
      – DCComb(2..0) = std_logic_vector(to_unsigned(i,3))
      – SSIDComb(13..0) = SSID(13..3) + DCComb(2..0)
    • Read HitCntMem
      – Read out HitCnt from HCM(i)
        » hcm_rdaddr(i) = SSID(13..3)
        » hcm_q(i) = HitCnt
  • Read HitListPointer
    – Read out baseaddr from HLP
      » hlp_rdaddr = SSID(13..0)
      » hlp_q = hlm_baseaddr
  • Read HitListMem
    – Read out hits from HLM
      » hlm_rdaddr = hlm_baseaddr + tempCNT
      » for tempCNT range 0 to HitCNT-1 LOOP
        • if tempCNT = HitCNT, HitLast=1
        – if HitLast=1, check next DCComb, i = i+1
      • if i = 7, finished with current AMSSID, SSLast=1
    • If SSLast=1, fetch new SSID