Comparison of data formats

Jan C. Bernauer for the organizers

Streaming Readout VII, 2020

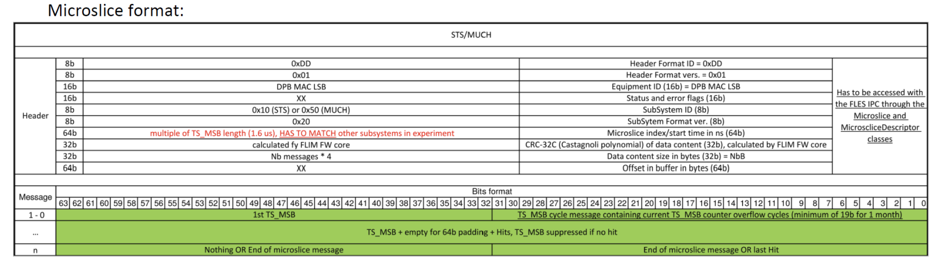
RBRC
RIKEN BNL Research Center

Stony Brook University

Thank you all for sending me the slides!

- Full slides are attached at the end. I'll show here some selections of things I noticed.

CBM



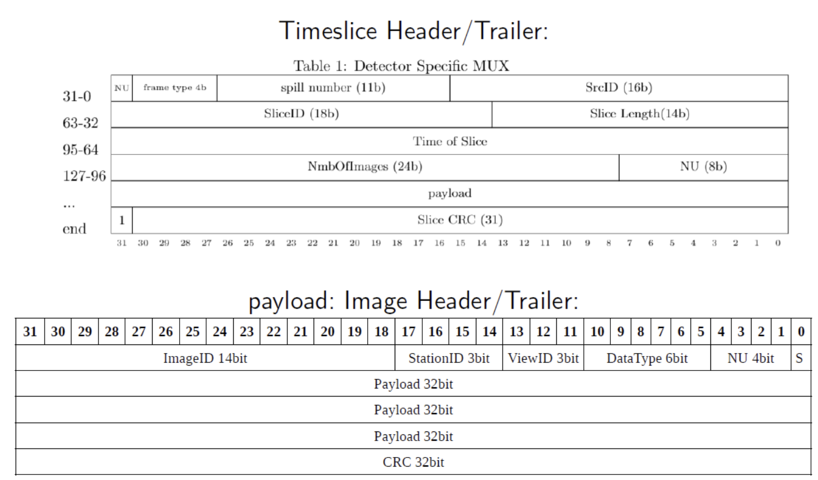
Microslice format:

STS/MUCH				
Header	8b	0xDD	Header Format ID = 0xDD	Has to be accessed with the FLES IPC through the Microslice and MicroscliceDescriptor classes
	8b	0x01	Header Format vers. = 0x01	
	16b	DPB MAC LSB	Equipment ID (16b) = DPB MAC LSB	
	16b	XX	Status and error flags (16b)	
	8b	0x10 (STS) or 0x50 (MUCH)	SubSystem ID (8b)	
	8b	0x20	SubSytem Format ver. (8b)	
	64b	multiple of TS_MSB length (1.6 us), HAS TO MATCH other subsystems in experiment	Microslice index/start time in ns (64b)	
	32b	calculated fy FLIM FW core	CRC-32C (Castagnoli polynomial) of data content (32b), calculated by FLIM FW core	
	32b	Nb messages * 4	Data content size in bytes (32b) = NbB	
	64b	XX	Offset in buffer in bytes (64b)	

Message	Bits format																																																															
	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1 - 0	1st TS_MSB																																TS_MSB cycle message containing current TS_MSB counter overflow cycles (minimum of 19b for 1 month)																															
...	TS_MSB + empty for 64b padding + Hits, TS_MSB suppressed if no hit																																																															
n	Nothing OR End of microslice message																																End of microslice message OR last Hit																															

- Byte aligned, padded to 8 bytes
- A microslice is part of a buffer? (Offset?)
- CRC

COMPASS++/AMBER



Timeslice Header/Trailer:

Table 1: Detector Specific MUX

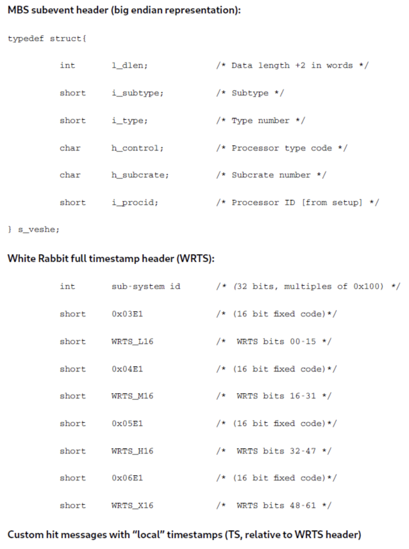
<table>
<tr><td>31-0</td><td colspan="1">NU</td><td colspan="4">frame type 4b</td><td colspan="11">spill number (11b)</td><td colspan="16">SrcID (16b)</td></tr>
<tr><td>63-32</td><td colspan="18">SliceID (18b)</td><td colspan="14">Slice Length(14b)</td></tr>
<tr><td>95-64</td><td colspan="32">Time of Slice</td></tr>
<tr><td>127-96</td><td colspan="24">NmbOfImages (24b)</td><td colspan="8">NU (8b)</td></tr>
<tr><td>...</td><td colspan="32">payload</td></tr>
<tr><td>end</td><td colspan="1">1</td><td colspan="31">Slice CRC (31)</td></tr>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
</table>

payload: Image Header/Trailer:

<table>
<tr><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td colspan="14">ImageID 14bit</td><td colspan="4">StationID 3bit</td><td colspan="3">ViewID 3bit</td><td colspan="6">DataType 6bit</td><td colspan="4">NU 4bit</td><td colspan="1">S</td></tr>
<tr><td colspan="32">Payload 32bit</td></tr>
<tr><td colspan="32">Payload 32bit</td></tr>
<tr><td colspan="32">Payload 32bit</td></tr>
<tr><td colspan="32">CRC 32bit</td></tr>
</table>

- bit packed
- 2xCRC

MBS



MBS subevent header (big endian representation):

```
typedef struct{

          int       l_dlen;             /* Data length +2 in words */

          short     i_subtype;          /* Subtype */

          short     i_type;             /* Type number */

          char      h_control;          /* Processor type code */

          char      h_subcrate;         /* Subcrate number */

          short     i_procid;           /* Processor ID [from setup] */

} s_veshe;
```

White Rabbit full timestamp header (WRTS):

```
          int       sub-system id       /* (32 bits, multiples of 0x100) */

          short     0x03E1              /* (16 bit fixed code)*/

          short     WRTS_L16            /*  WRTS bits 00-15 */

          short     0x04E1              /* (16 bit fixed code)*/

          short     WRTS_M16            /*  WRTS bits 16-31 */

          short     0x05E1              /* (16 bit fixed code)*/

          short     WRTS_H16            /*  WRTS bits 32-47 */

          short     0x06E1              /* (16 bit fixed code)*/

          short     WRTS_X16            /*  WRTS bits 48-61 */
```

Custom hit messages with "local" timestamps (TS, relative to WRTS header)

RCDAQ

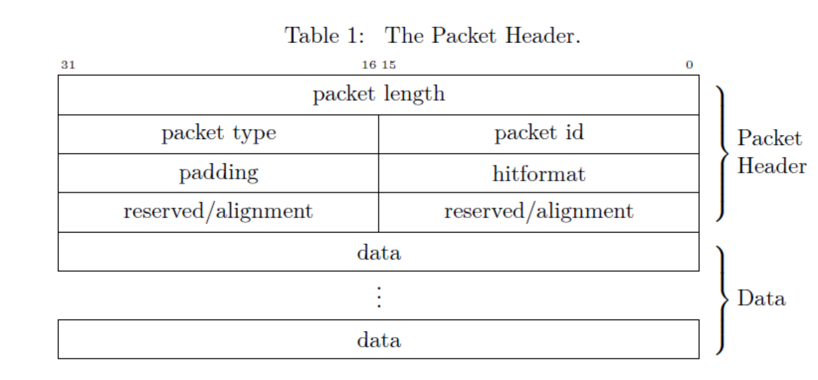


Table 1: The Packet Header.

31 16	15 0	
packet length		Packet Header
packet type	packet id	
padding	hitformat	
reserved/alignment	reserved/alignment	
data		Data
⋮		
data		

- short aligned, 32 bit padded
- no crc?

Take away / Questions

- Rarely footer (which might be nice for CRC in a streaming setup?)
- Mostly byte aligned
- Typical fields: Length (first?), event nr/timing, channel
- Length of all, or payload?
- Since we won't have "events", how do we specify the time?
    - Absolute wall clock time?
    - Bunch Nr (+offset for finer timing?)

Streaming readout for multi-purpose DAQ system MBS

data packets from free running systems:

- free running data acquisition systems sends formatted **MBS sub-events** of hits
- each sub-event is **headed by White Rabbit Time Stamp (WRTS) - 1 ns units, starting from 1970**
- each sub-event contains **data from “many” hits** (MBS container)
- **each hit has TS** of variable size, but significantly smaller than WRTS
- each hit TS must have a **sufficient correlation to full WRTS** header
- **hit data format** inside MBS container has no dependency for time sorting and **can be chosen freely by each detector/sub-system**

MBS subevent header (big endian representation):

```
typedef struct{
          int       l_dlen;             /* Data length +2 in words */
          short     i_subtype;          /* Subtype */
          short     i_type;             /* Type number */
          char      h_control;          /* Processor type code */
          char      h_subcrate;         /* Subcrate number */
          short     i_procid;           /* Processor ID [from setup] */
} s_veshe;
```

White Rabbit full timestamp header (WRTS):

```
          int       sub-system id       /* (32 bits, multiples of 0x100) */
          short     0x03E1              /* (16 bit fixed code)*/
          short     WRTS_L16            /*  WRTS bits 00-15 */
          short     0x04E1              /* (16 bit fixed code)*/
          short     WRTS_M16            /*  WRTS bits 16-31 */
          short     0x05E1              /* (16 bit fixed code)*/
          short     WRTS_H16            /*  WRTS bits 32-47 */
          short     0x06E1              /* (16 bit fixed code)*/
          short     WRTS_X16            /*  WRTS bits 48-61 */
```

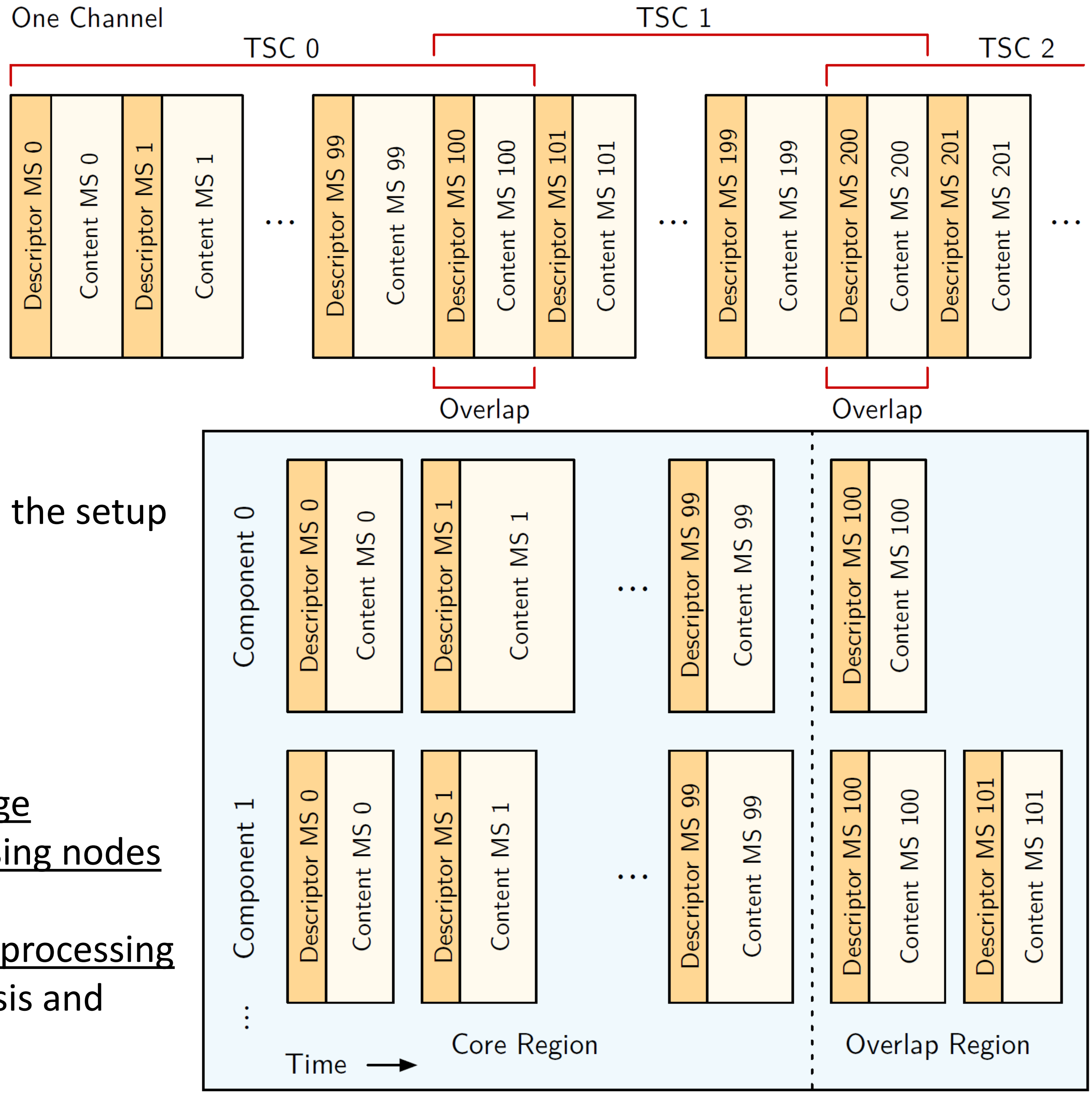
Custom hit messages with “local” timestamps (TS, relative to WRTS header)

…..

CBM: Data containers for transport and analysis

- MicroSlice(µS) = self-contained data container from a single CRI and for a fixed period of time
  - Output of the CRI in CBM => generated in FW
  - Constant length in experiment time
  - Typical period of time: 10’s of µs to ms
  - Length adapted to data format, data rate (beam condition), container efficiency and network performance

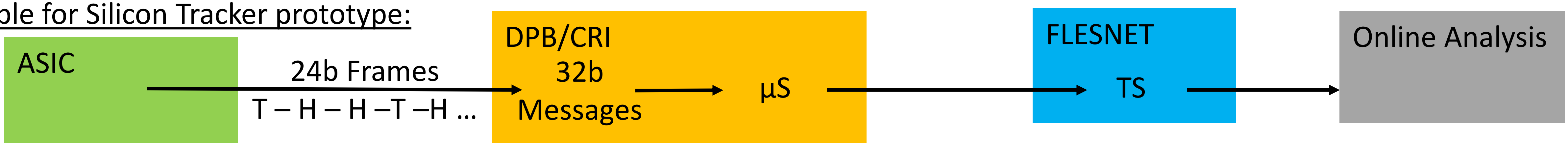
- TimeSlice(TS) = container collecting the µS of all CRI cards in the setup and for a given number of µS
  - Assembled by FLESNET (CBM DAQ prototype)
    ⇒ generated in SW
  - Typical number of µS per source: 10-1000
    ⇒ time range: ms to s
  - Includes overlap µS to avoid analysis losses close to edge
  - Length adapted to match memory ressource in processing nodes
  - One TS for the full setup for each time interval
  - Input unit for Online analysis: TS are distributed to the processing nodes for reconstruction, event building, physics analysis and selection



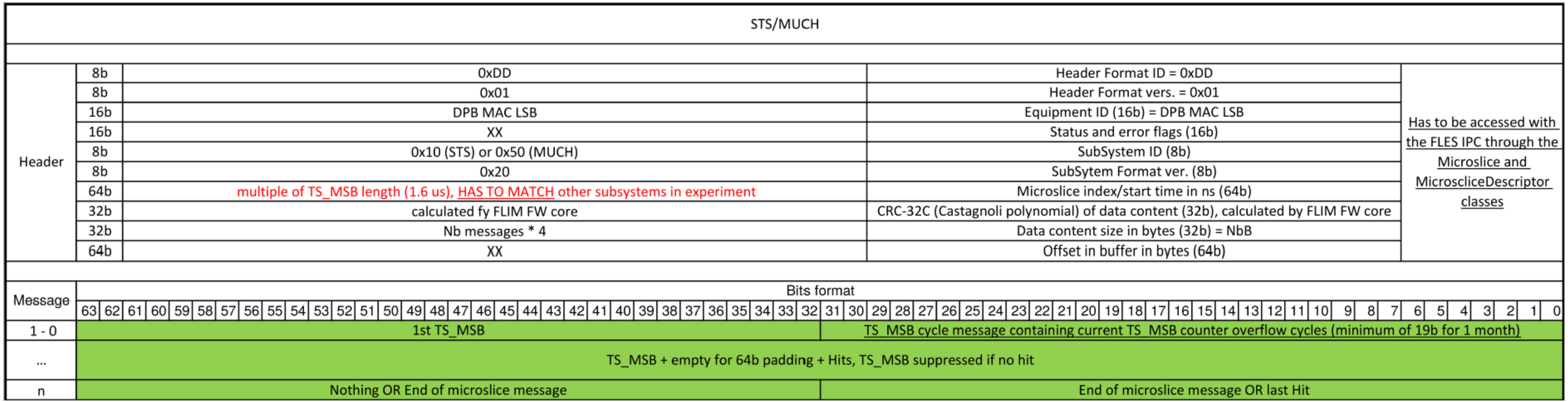
CBM: Data sources, Data Format and constraints on containers

- Data sources = ASICs and/or FPGAs, self-triggered and free-streaming
- Messages = representation of the signals generated in data sources, not context free as optimized for best compromise between resolution and bandwidth usage
- Typical stream organization: Periodic context messages with MSB of timestamp + Hit messages with LSB & ADC, TOT, chan …
- MicroSlice(µS) = self-contained data container from a single CRI and for a fixed period of time, granularity of length choice depends on context messages interval

Example for Silicon Tracker prototype:



Microslice format:

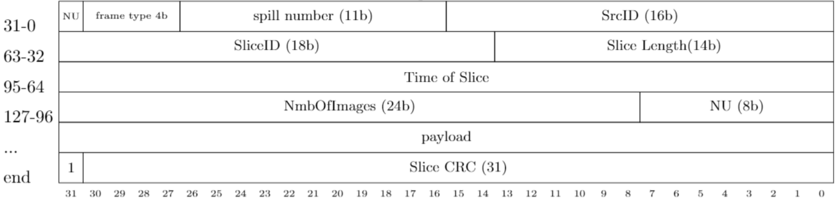


		STS/MUCH		
Header	8b	0xDD	Header Format ID = 0xDD	Has to be accessed with the FLES IPC through the Microslice and MicroscliceDescriptor classes
	8b	0x01	Header Format vers. = 0x01	
	16b	DPB MAC LSB	Equipment ID (16b) = DPB MAC LSB	
	16b	XX	Status and error flags (16b)	
	8b	0x10 (STS) or 0x50 (MUCH)	SubSystem ID (8b)	
	8b	0x20	SubSytem Format ver. (8b)	
	64b	multiple of TS_MSB length (1.6 us), HAS TO MATCH other subsystems in experiment	Microslice index/start time in ns (64b)	
	32b	calculated fy FLIM FW core	CRC-32C (Castagnoli polynomial) of data content (32b), calculated by FLIM FW core	
	32b	Nb messages * 4	Data content size in bytes (32b) = NbB	
	64b	XX	Offset in buffer in bytes (64b)	
Message		Bits format		
		63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
1 - 0		1st TS_MSB	TS_MSB cycle message containing current TS_MSB counter overflow cycles (minimum of 19b for 1 month)	
…		TS_MSB + empty for 64b padding + Hits, TS_MSB suppressed if no hit		
n		Nothing OR End of microslice message	End of microslice message OR last Hit	

COMPASS++/AMBER FriDAQ Protocol

Timeslice Header/Trailer:

Table 1: Detector Specific MUX



31-0	NU	frame type 4b	spill number (11b)	SrcID (16b)
63-32	SliceID (18b)	Slice Length(14b)		
95-64	Time of Slice			
127-96	NmbOfImages (24b)	NU (8b)		
...	payload			
end	1	Slice CRC (31)		

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

payload: Image Header/Trailer:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ImageID 14bit														StationID 3bit				ViewID 3bit			DataType 6bit						NU 4bit				S
Payload 32bit																															
Payload 32bit																															
Payload 32bit																															
CRC 32bit																															

1 Data Formats

1.1 The Packet Header

The composition of the full 16-byte header is shown in table 1.

Table 1: The Packet Header.

31 16	15 0	
packet length		Packet Header
packet type	packet id	
padding	hitformat	
reserved/alignment	reserved/alignment	
data		Data
⋮		
data		

The length is measured in 32-bit units (DWords), which allows us to have packet lengths of up to 16 GBytes, although most packets are substantially smaller. Each packet is a multiple of 32bit units (so each data structure starts at least on a 32 bit boundary). We usually choose higher data alignment boundaries (64bit or even 128bit). Aligning the data blocks to the prevailing CPU data bus widths (64bit at present) speeds up the processing of data.

The fields in the header are

packet length the overall length of the packet structure in Dwords

packet id a unique identifier for the packet that says which unit generated the packet

packet type indicates the fundamental storage size in the packet, expressed in bytes (1 (character data), 2 (16bit), 4 (32 bit),...). This field is also known as the "swap unit" in case the data payload has to be byte-swapped for a different CPU architecture. It also gives the unit for the padding value.

hitformat This value identifies an algorithm to decode the data payload so the decoded data can be accessed by a set of standard APIs.

padding The amount of additional data added to bring the packet size to the desired alignment boundary.

2 reserved/alignment fields Those fields can hold 2 16bit values as needed to verify the proper alignment of various data blocks. They are set to 0 if unused.

31 16	15 0	
packet length = 6		Packet Header
type = 2	id = 1001	
padding = 1	hitformat = 3002	
0x3A	0x79CE	
40	20	Data
0	55	

Table 2: An example of a (fictitious) 64-bit aligned packet that holds the three 16bit values 20, 40,and 55, and a combined alignment value of 0x3A79CE.

31 16	15 8	7 0	
Event length			Event Header
reserved		event type	
Event Sequence			
Run Number			
Time/Alignment 0			
Time/Alignment 1			
Luminosity Block			
reserved			
data			Data
⋮			
data			

Table 3: The structure of an Event Header.

Table 2 shows the composition of a (fictitious) packet with id 1001 that holds three 16-bit values (so it has the packet type 2) 20, 40, and 55. In order to maintain the 64bit alignment of the data, an additional 16bit word is added, which gives a padding value of 1.

1.2 The Event Header

While in transit between components, for example between a SEB and a Buffer Box, a number of packets is preceded by a *Event Header*, sometimes also called the *Frame Header*.

Table 3 shows the structure of the event header. The event length is again given in

	31 16 15 8	7 0	
0x0000890c	event length = 0x890c = 35084		Event Header
0x00000002	0x000000	type = 2	
0x00000002	Event Sequence = 2		
0x00001051	Run Number= 0x1051 = 4177		
0x00000000	Time field 1 = 0		
0x5be1e129	Time field 2 = 0x5be1e129 = 1541529897		
0x00000000	Luminosity Block = 0		
0x00000000	reserved = 0		
	packet data		Data
	⋮		
	packet data		

Table 4: A hex-dump of an actual Event Header and its structure. The event type 2 denotes streaming data. Because the first time field is 0, the 2nd word is interpreted as a Unix time (1541529897). This corresponds to a date of Nov 6, 2018, 13:44:57, when the data were taken.

DWords (32bit length).

The Event header structure has two general-purpose time and alignment fields. If the first alignment field is 0, the second field is interpreted as a Unix time (32bits). Else the two fields are interpreted as system-specific alignment data.