celepixel

芯仑科技
CeleX-5 相机套件
SDK 使用手册

芯仑科技（上海）有限公司

Content

Version Control

Version	Date	Section	Description	Author
1.0	2019.04.12	All	New	Xiaoqin Hu, Hua Ren
1.1	2019.05.21	All	Update	Xiaoqin Hu

1. 概述

1.1. CeleX-5 Sensor 芯片相机套件的基本介绍

1. 1. 1 CeleX-5 相机套件工作原理

CeleX-5 芯片套件有两种工作方式：并口输出数据方式和 MIPI 串口输出数据方式。通过并口输出数据的这种方式跟 CeleX-4 芯片套件的工作方式一致，即通过一块 Opal Kelly 的 FPGA 处理板把 Sensor 的数据传输出，在此就不再赘述。

本 SDK 使用的是 MIPI 串口输出方式，图 1-1 给出了它的基本工作原理。它需要一个把 MIPI 数据转成 USB3.0 数据的驱动，然后 PC 从 USB3.0 Driver 中获取 sensor 的数据，同样 PC 端也可以通过 Driver 对 Sensor 进行一些配置。

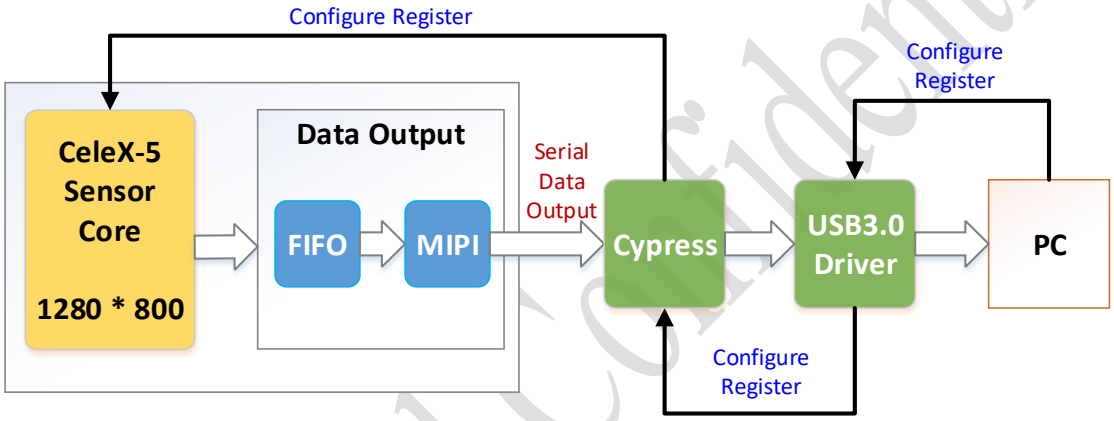


Fig. 1-1 Basic working principle of the CeleX-5 Chipset

1.1.2 术语

下表中列出了在本文档中出现的一些术语以及它们的解释。

No.	Terminology	Description
1	Event Mode	*Event Address-only Mode*, *Event Optical-flow Mode* 和 *Event Intensity Mode* 的总称，在这些模式下 CeleX-5 sensor 只输出动态的数据。
2	Full-frame Mode	*Full-frame Picture Mode*, *Single Full-frame Optical-flow Mode* 和 *Multiple Full-frame Optical-flow Mode* 的总称，在这些模式下 CeleX-5 sensor 输出的是一个完整的 full-frame 数据。
3	Fixed Mode	固定模式，CeleX-5 sensor 只工作一种固定的模式下。

4	Loop Mode	循环模式，CeleX-5 sensor 可以 3 种模式之间自动的切换。
5	Event Address-only Mode	CeleX-5 sensor 的一种工作模式，在该模式下，Sensor 检测视野中的动态信息并仅输出检测到的 Event 的行/列地址。
6	Event Optical-flow Mode	CeleX-5 sensor 的一种工作模式，在该模式下，Sensor 检测视野中的动态信息并仅输出检测到的 Event 的行/列地址，同时，也会输出 optical-flow 信息，但在模式下没有亮度信息
7	Event Intensity Mode	CeleX-5 sensor 的一种工作模式，在该模式下，Sensor 检测视野中的动态信息并仅输出检测到的 Event 的行/列地址，同时，也会输出亮度信息，但在模式下没有 optical-flow 信息。
8	Full-frame Picture Mode	CeleX-5 sensor 的一种工作模式，在该模式下，Sensor 可以生成带有像素点亮度信息的 full-frame 图像帧，且所有像素的亮度信息都是在某个时刻被同时采样的。
9	Single Full-frame Optical-flow Mode	CeleX-5 sensor 的一种工作模式，在该模式中，Sensor 可以生成带有 optical-flow 信息的 full-frame 图像帧。
10	Multiple Full-frame Optical-flow Mode	CeleX-5 sensor 的一种工作模式，在该模式中，Sensor 可以在一段持续时间内生成若干带有 optical-flow 信息的 full-frame 图像帧。
11	Event Binary Pic	本 SDK 中输出的 Event 的二值图像帧
12	Event Gray Pic	本 SDK 中输出的 Event 的灰度图像帧
13	Event Accumulated Gray Pic	本 SDK 中输出的 Event 的累加灰度图像帧
14	Event Count Pic	本 SDK 中输出的 Event 的触发次数图像帧

1.2. CeleX-5 Sensor 的工作原理

CeleX™ 是针对机器视觉而设计的智能图像传感器系列。传感器中的每一像素点能够独立自主地监测相对光强的变化，并在到达阈值时被激发发出被读出信号。行和列仲裁电路实时处理像素激发信号，并确保即使同时接收到多个请求时能够按有序的方式逐一处理。传感器依据被激发的事件，输出连续的异步数据流，而不是图像帧。CeleX™ 传感器监测的运动物体**速度不再受传统的曝光时间和帧速率限制**。它可以侦测高达万帧/秒昂贵高速相机才能获取到的高速物体运动信息，而且还能大幅降低后端处理量，结合相关配套后端处理软件，

已可实现视频回放帧率的高速运动物体信息捕捉，回放帧率可以在 25~10000 帧/秒范围内进行选择。

CeleX™ 能并行输出三种模式数据：传统图像帧，动态数据流和全幅光流数据。这款传感器可以在各种领域中进行应用，如：辅助/自动驾驶，UAV，机器人，监控等。

CeleX-5 是一款多功能智能图像传感器，具有一百万像素和片上集成的一些附加功能。传感器支持几种不同的输出格式：纯二进制地址事件，具有像素强度信息或定时信息的地址事件。此外，传感器的读出方案可以是异步数据流或同步全帧。输出格式和读出方案的不同组合使该传感器具有很大的灵活性，总共支持 6 种独立的操作模式。

为了进一步满足不同应用的要求，传感器还可以配置为 Loop 模式，可以在三种不同模式之间自动切换。

1.2.1. CeleX-5 Sensor 的 Fixed 模式

CeleX-5 Sensor 共有 6 种固定工作模式：*Event Address Only 模式, Event Optical-flow 模式, Event Intensity 模式, Full-frame Picture 模式, Single Full-frame Optical-flow 模式, Multiple Full-frame Optical-flow 模式*。

1.2.1.1. Event 模式（仅有地址信息）

在该模式下，Sensor 检测视野中的动态信息并仅输出检测到的 Event 的行/列地址。需要注意的是，在模式下既没有亮度信息也没有 optical-flow 信息。该模式的优点在于 Sensor 可以以更高速地检测 Event 并输出信息，其工作原理如图 1-2 所示。

1.2.1.2. Event 模式（带 Optical-Flow 信息）

在该模式下，Sensor 检测视野中的动态信息并仅输出检测到的 Event 的行/列地址，同时，也会输出 optical-flow 信息，需要注意的是，在模式下没有亮度信息，其工作原理如图 1-2 所示。

1.2.1.3. Event 模式（带亮度信息）

在该模式下，Sensor 检测视野中的动态信息并仅输出检测到的 Event 的行/列地址，同时，也会输出亮度信息，需要注意的是，在模式下没有 optical-flow 信息，其工作原理如图 1-2 所示。

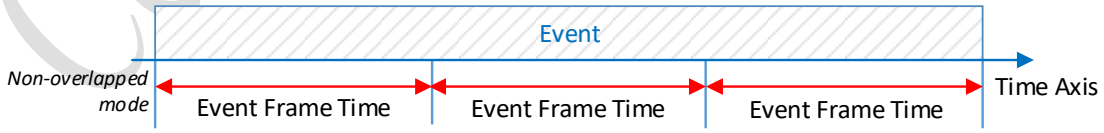


Fig. 1-2 Event data mode

在 Event 模式下，通过 API，可以同时创建多种 Event 图像帧：累加的图像帧（用最新变化的像素点的灰度值累加出来的灰度图） 和只有变化的像素点的二值图像帧（发生变化的像素点的灰度值标记为 255，没有发生变化的像素点的灰度值被标记为 0）等等。

在 Event 数据模式下，建帧时间（Frame Time）是可以调整的（范围：1~1000ms），用户可以通过调用 API 接口 *setEventFrameTime* 来修改这个时间。创建 Event 图像帧的方法见 1.2.4.2 章节。

1.2.1.4. Full-Frame Picture 模式

在该模式下，Sensor 可以生成带有像素点亮度信息的 full-frame 图像帧，且所有像素的亮度信息都是在某个时刻被同时采样的，其工作原理如图 1-3 所示。因为 Sensor 可以生成连续的 full-frame 图像帧，所以它可以用作基于帧的 APS 传感器。

Sensor 的工作频率与传输数据的帧率为线性关系，即频率为 x MHz，则 fps 则为每秒 x 帧。如果 Sensor 的工作频率为 100MHz，则表示 Sensor 每秒可生成大约 100 个 full-frame 图像帧，也即是 Transmission Time 为 10ms。

在此模式下，您可以通过调用 API 接口 *setClockRate* 来修改 Sensor 的工作频率，从而修改传输时间（或 Frame Time），还可以通过调用 API 接口 *setBrightness* 和 *setContrast* 来提高图像质量。

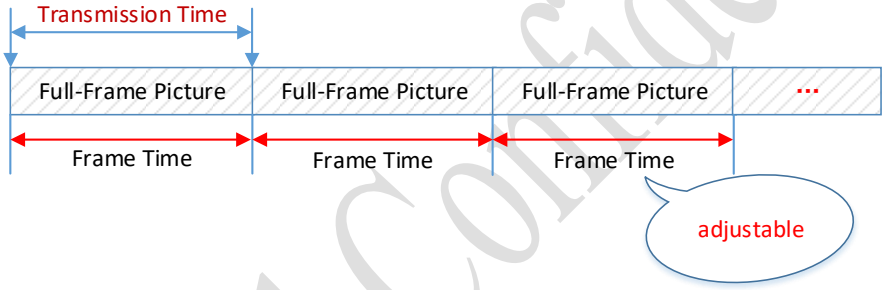


Fig 1-3. Full-Frame Picture Mode

1.2.1.5. Single Full-Frame Optical-Flow 模式

在该模式中，Sensor 可以生成带有 optical-flow 信息的 full-frame 图像帧。Sensor 首先在指定的持续时间（*Accumulated Time*）内累积 Event，之后，Sensor 将在 *Transmission Time* 内输出一个 full-frame 图像帧，其工作原理如图 1-4 所示。

用户可以设置的参数是 a）累计时间（*Accumulated Time*，硬件参数），b）传输时间（*Transmission Time*，硬件参数，与主时钟频率相关）。

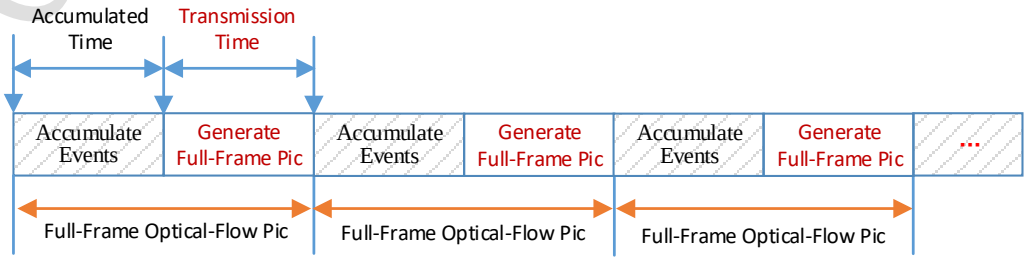


Fig 1-4. Full-Frame Optical-Flow (Single) Mode

1.2.1.6. Multiple Full-Frame Optical-Flow 模式

在该模式中，Sensor 可以在一段持续时间内生成若干带有 optical-flow 信息的 full-frame 图像帧。与 *Event Optical-flow Mode* 不同的是，在此模式下 Sensor 输出的数据为 full-frame 图像而不是异步事件流。与 *Single Full-frame Optical-flow Mode* 不同的是，Sensor 可以在一段时间内产生多个 full-frame 图像，其工作原理如图 1-5 所示。

该模式最好的是 Loop 模式中使用，固定模式下不建议使用。在 Loop 模式中，可以通过调用 API 接口 *setPictureNumber* 来设置每次中要输出几个 full-frame 的光流数据。

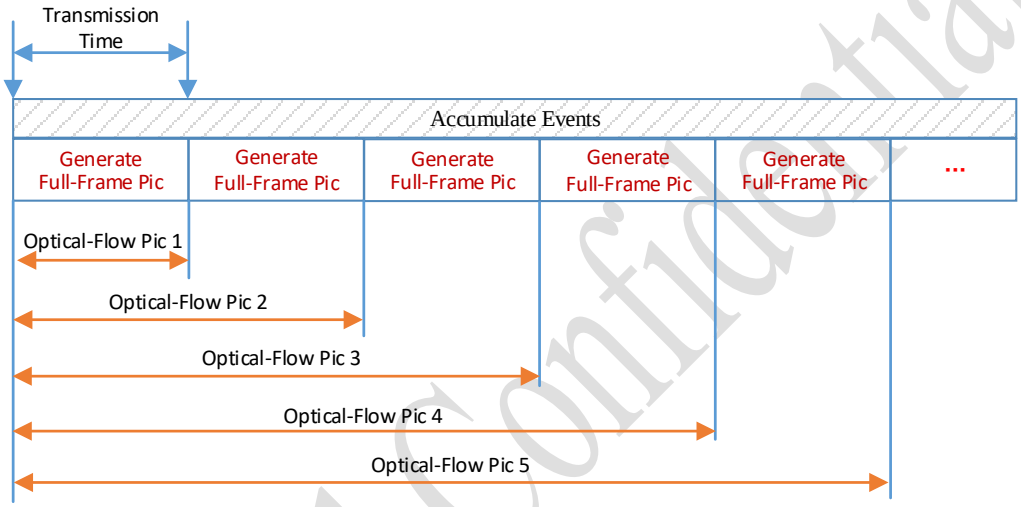


Fig 1-5. Full-Frame Optical-Flow (Multiple) Mode

1.2.2. CeleX-5 Sensor 的 Loop 模式

对于 Fixed 模式，Sensor 始终只能工作在上面介绍的 6 种模式中的一种模式下。对于 Loop 模式，Sensor 可以工作在 3 个模式下，并在这 3 个模式之间启动切换，如下图 1-6 所示。

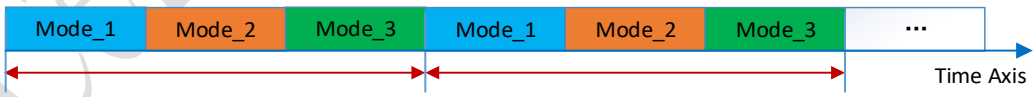


Fig 1-6. Loop Mode

要想让 Sensor 工作在 Loop 模式，首先需要调用 API 接口 *setSensorModeEnable* 开启该功能，然后调用 API 接口 *setSenorLoopMode* 为每个 Loop 设置一个模式，即分别为 Mode_1，Mode_2 和 Mode_3 设置一个模式，这样，Sensor 即可同时输出 3 模式的数据。

下面给出如何选择 Loop 模式中的 3 个模式的 2 点建议：

a) 3 个模式最好的搭配为：一个 *Full-frame Picture Mode*，一个 *Event Mode* 和一个 *Optical-flow Mode*。

b) 只选择一种 Optical-flow 模式。

默认情况下，*Loop Mode* 中的第 1 个模式（Mode_1）为 *Full-frame Picture Mode*，它的持续时间为 **10ms**，这个时间可以通过调用 API 接口 *setPictureNumber* 来修改。

Loop Mode 中的第 2 个模式（Mode_2）为 *Event Mode*，它的持续时间为 **20ms**，这个时间可以通过调用 API 接口 *setEventDuration* 来修改。

Loop Mode 中的第 3 个模式（Mode_3）为 *Single Full-frame Optical-flow Mode*，它的持续时间为 3**0ms**，这个时间可以通过调用 API 接口 *setPictureNumber* 来修改。

1.2.3. CeleX-5 Sensor 的数据格式

1.2.3.1. MIPI 数据格式

对于 MIPI 接口的数据，Full-Frame 和 Event 模式下的数据格式差异比较大，下面将分别介绍这两种类型的数据。Full-Frame 模式的数据格式比较简单，它就是连续灰度值信息（ADC），如下图所示。从连续的 3 个字节中，获取两个像素的 ADC 值。

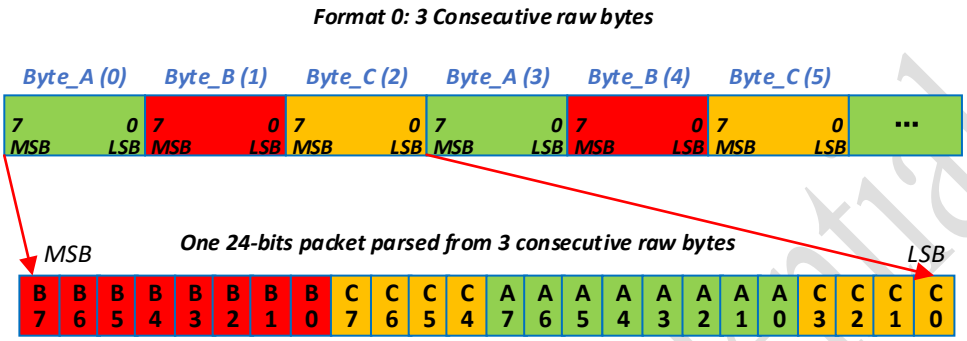


Fig. 1-9

在 Event 模式下，CeleX-5 Sensor 共支持 3 种数据格式，用户可以通过 *API 接口*来设置使用哪种数据格式，需要注意的是：这个设置只能在调用 *openSensor* 之前使用，Sensor 一旦被启用后，调用该接口修改数据格式将无效。

如果不关心如何解析数据，可以跳过下面的章节，因为本 SDK 中已经解析好这些数据，并以图像和(x, y, A, T)等信息的方式提供给用户。

下面将介绍这 3 种数据格式以及如何从这些数据种解析出来：row address，column address，ADC， row timestamp，temperature 等信息。

1. Format 0

Format 0 的数据格式如下图所示，它共有 4 种 Package，每个 Package 为 24-bits（3 个字节）。最低的 2 个 bit 表示 Package 的类型（ID），例如，当 ID = 2b’10 时，表示是 Package_A，从中可以解析出来 row address 和 row timestamp。在该模式下，ADC 信息只有 8 bits，row timestamp 也只有 12 bits。

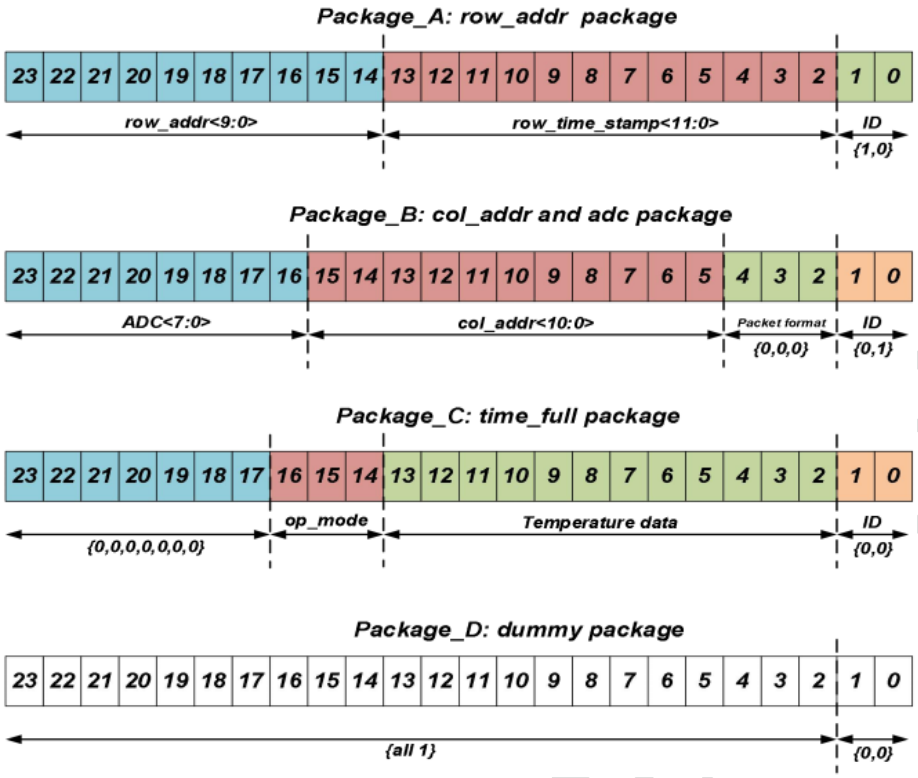


Fig. 1-10 Format 0: 24-bit packet with ADC data

需要注意的是，这 24-bits 的数据不是原始数据中的连续的 24-bits，而是需要重新排列它们的顺序才能得到，下图给出如何拼接这个 24-bits 数据的方法。

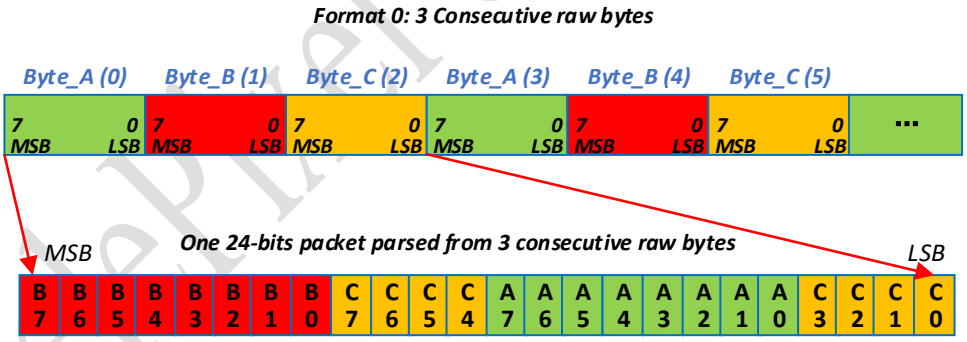


Fig. 1-11

2. Format 1

Format 1 的数据格式如下图所示，它也有 4 种 Package，每个 Package 为 28- bits。同样，最低的 2 个 bit 表示 Package 的类型（ID），例如，当 ID = 2b'10 时，表示是 Package_A，从中可以解析出来 row address 和 column address。与 Format 0 不同的是，该模式下，ADC 信息只有 12 bits，row timestamp 有 16 bits。

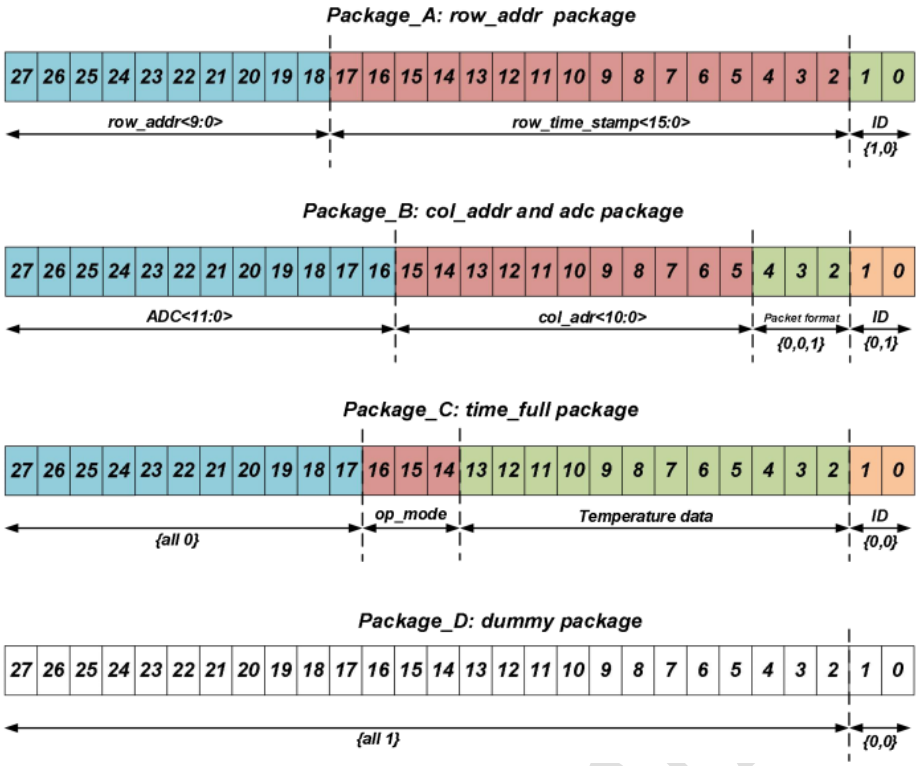


Fig. 1-12 Format 1: 28-bit packet with ADC data

同样，这 28-bits 的数据也不是原始数据中的连续的 28-bits，而是需要重新排列它们的顺序才能得到，下图给出如何拼接这个 28-bits 数据的方法。它是用连续的 56-bits（7 个字节）的原始数据，按照下图的方法拼接出 2 个 28-bits 的 Package。

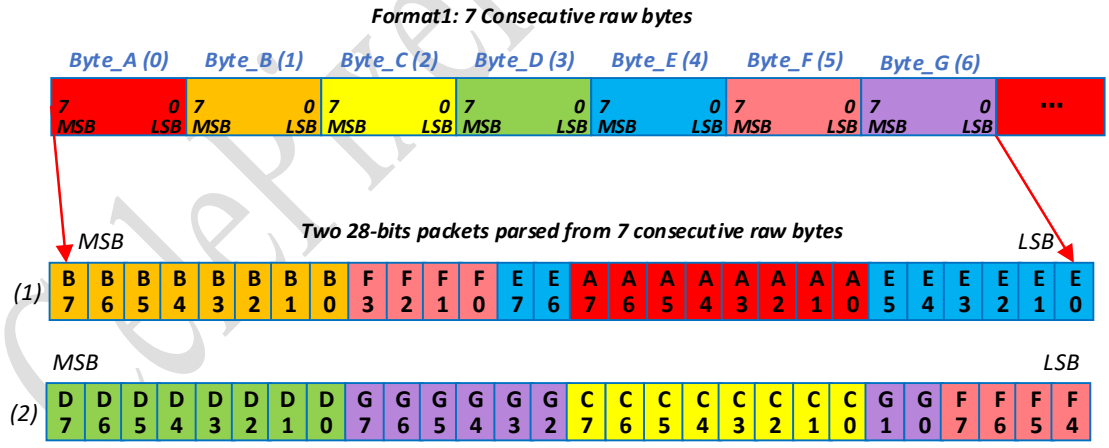


Fig. 1-13

2. Format 2

Format 2 的数据格式如下图所示，它共有 5 种 Package，每个 Package 为 14- bits。同样，最低的 2 个 bit 表示 Package 的类型（ID），例如，当 ID = 2b'10 时，表示是 Package_A，从中可以解析出来 row address。与前两种数据格式不同的是，在模式下没有 ADC 信息。如果

想获取 Event 的 ADC 信息，则不能使用该模式。

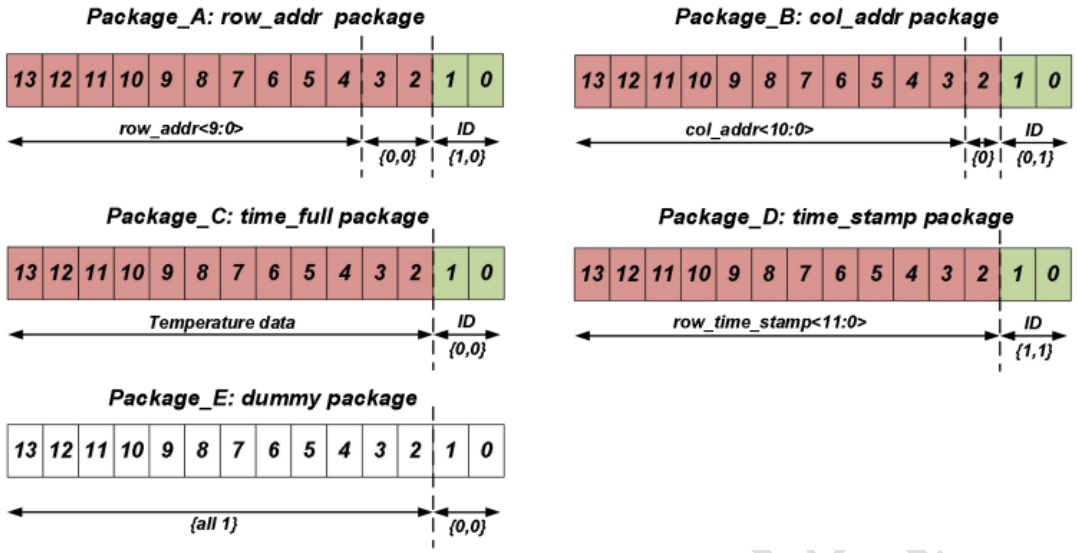


Fig. 1-14 Format 2: 14-bit packet without ADC data

同样，这 14-bits 的数据也不是原始数据中的连续的 14-bits，而是需要重新排列它们的顺序才能得到，下图给出如何拼接这个 14-bits 数据的方法。它是用连续的 56-bits（7 个字节）的原始数据，按照下图的方法拼接出 4 个 14-bits 的 Package。

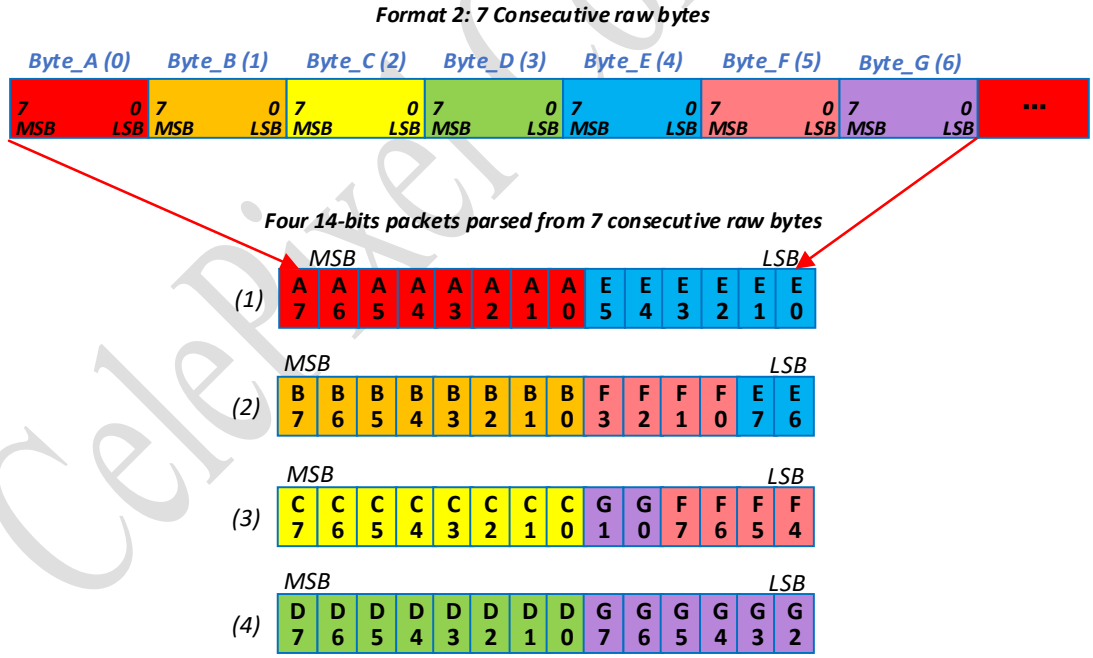


Fig. 1-15

1.2.3.2. SDK 输出的数据格式

CeleX-5 Sensor 工作在不同的模式，SDK 输出的数据类型也有所不同，下表给出了在不同的模式下，用户可以获取到的数据类型。

Table 1-1：SDK 输出的数据类型

Sensor 的工作模式	SDK 输出的图像数据类型
Full-frame Picture Mode	Full Pic Buffer/Mat
Event Address-only Mode	Event Binary Pic Buffer/Mat Event Denoised Pic Buffer/Mat Event Count Pic Buffer/Mat Event Vector<row, col, timestamp>
Event Optical-flow Mode	Event Optical-flow Pic Buffer/Mat Event Binary Pic Buffer/Mat Event Vector<row, col, optical-flow info, timestamp>
Event Intensity Mode	Event Binary Pic Buffer/Mat Event Gray Pic Buffer/Mat Event Count Pic Buffer/Mat Event Accumulated Pic Buffer/Mat Event Superimposed Pic Buffer/Mat Event Vector<row, col, brightness, polarity, timestamp>
Single Full-frame Optical-flow Mode	Event Optical-flow Pic Buffer/Mat Event Optical-flow Direction Pic Buffer/Mat Event Optical-flow Speed Pic Buffer/Mat Event Binary Pic Buffer/Mat

其中，***xx Pic Buffer*** 为 row = 1280，column = 800 的单通道图像数组，而 ***xx Pic Mat*** 为 row = 1280，column = 800 的单通道的 OpenCV mat。

需要特别指出的是，Event Vector 中的 timestamp 在不同的模式或频率下，其范围或对应的绝对时间会有所不同，具体如下所述。

Event data format = 1（Event Intensity Mode 和 Event Optical-flow Mode 采用的数据格式）

（1）Sensor 默认的工作频率为 70MHz，可以调节的范围：20MHz ~ 70MHz

（2）Sensor 输出 event 的时间戳范围：0 ~ 65535（16 bits）

（3）SDK 默认的收集 30ms 的数据建立一帧图像，对应的时间戳范围：0 ~ 2142

Event data format = 2（Event Address Only Mode 采用的数据格式）

（1）Sensor 默认的工作频率为 100MHz，可以调节的范围：20MHz ~ 100MHz

（2）Sensor 输出 event 的时间戳范围：0 ~ 4095（12 bits）

（3）SDK 默认的收集 30ms 的数据建立一帧图像，对应的时间戳范围：0 ~ 1500

Table 1-2：不同频率下的时间精度和时间戳范围

Clock (MHz)	T-Unit (us)	30ms 对应的时间单位个数
20	T_Unit = 25	30*1000 / 25 = 1200
30	T_Unit = 22	30*1000 / 22 = 1363
40	T_Unit = 25	30*1000 / 25 = 1200
50	T_Unit = 20	30*1000 / 20 = 1500
60	T_Unit = 17	30*1000 / 17 = 1764
70	T_Unit = 14	30*1000 / 14 = 2142
80	T_Unit = 25	30*1000 / 25 = 1200
90	T_Unit = 22	30*1000 / 22 = 1363
100	T_Unit = 20	30*1000 / 20 = 1500

1.2.4. 创建 Full-frame 和 Event 图像帧的方法

CeleX-5 Sensor 套件上电之后，就可以输出连续的像素事件。如上一节所述，我们可以从事件中解码 X，Y，A，T 信息。接下来将介绍如何通过（X, Y, A, T）信息来创建图像帧。

1.2.4.1. 创建 Full-Frame 图像帧的方法

1) 调用 API 接口 *setClockRate* 设置 Sensor 的工作频率，指示 Sensor 每经过一定的时间间隔，产生一个完整的图像帧（Full Frame）。这是一种兼容传统 Sensor 的工作模式，也即不管像素上有没有发生光线的变化，都能获取到它的灰度值。

2) 把 Sensor 的工作模式切换至 Full-frame 模式。

3) 构建一个二维数组来表示 Full Pic，假设 M [800] [1280]，它有 800 行，每行有 1280 个像素将每个像素的亮度值初始化为 0。

解析从 USB driver 中获得的数据。每个事件 E 被解码为（X,Y,A,T）时，M [Y] [X]上的像素的亮度值为 A；在 Full-Picture 模式下，（T）信息是无效的。

4) 在新的 Full-frame Time 中，重复步骤 4。

上述建帧的过程是在 API 内部实现的，用户可以直接调用 getFullPicBuffer 这个函数来获得上述的 Full Picture 图像帧的内容。

1.2.4.2. 创建 Event 图像帧的方法

在 Event 模式下，创建图像帧的方法有很多，下面仅介绍一下本 SDK 内部是如何创建 Event 二值图像帧（Event Binary Pic），Event 灰度图像帧（Event Gray Pic）以及 Event 累加灰度图像帧（Event Accumulated Gray Pic）的创建方法

1) 调用 API 接口 setEventFrameTime 设置 Event 建帧时间，指示本 SDK 将会把 Sensor 每经过一定的时间间隔（即用户设置的 Event Frame Time）内输出的 Event，组合起来产生一个 Event 二值图像帧。
2) 把 Sensor 的工作模式切换至 Event 模式。
3) 创建 3 个二维数组 M1 [800] [1280]，M2 [800] [1280]以及 M3 [800] [1280]，分别用来表示 Event Binary Pic，Event Gray Pic 以及 Event Accumulated Gray Pic。并将 3 个二维数组中的每个像素的亮度值初始化为 0。
4) 从 FPGA 获得的数据流中，解析获得的数据。每个事件 E 被解码为（X,Y,A,T）时，M 1[Y] [X]上的像素的亮度值为 255，M 2[Y] [X]上的像素的亮度值为 A，M 3[Y] [X]上的像素的亮度值为 A。
5) 在新的 Event Frame Time 中，首先将二维数组 M1 和 M2 中的每个像素的亮度值为设置为 0，M3 保持不变；然后再重复步骤 4。也就是说，在每个 Event Frame Time 中，M1 和 M2 都是每次清零的，而 M3 数组则是在上一个 Event Frame Time 建立的数组的基础上更新的。

上述建帧的过程是在 API 内部实现的，用户可以直接调用 getEventPicBuffer 这个函数来获得上述的各图像帧的内容。为了帮助用户理解 Event Gray Pic 和 Event Accumulated Gray Pic 的区别，下面用一个 5 * 5 阵列来进一步说明上述过程，本文档仅列出前五帧作为范例。

在图 1-16 中的每一帧表示 Event 模式下的 Gray Pic，在图 1-17 中的每一帧表示 Event 模式下的 Accumulated Gray Pic。

其中“x”表示 Event 模式下发生变化个的像素的灰度值，红色的“x”表示在第一个 Event Frame Time 中变化的像素，蓝色的“x”表示在第二个 Event Frame Time 中变化的像素，绿色的“x”表示在第三个 Event Frame Time 中变化的像素，紫色的“x”表示在第四个 Event Frame Time 中变化的像素，黑色的“x”表示在第五个 Event Frame Time 中变化的像素。

在图 1-17 中，应该注意的是，当一个像素先前发生了变化，然后它又发生了变化，我们直接用新的灰度值代替旧的灰度值，例如，图示的位置（row0，col0）和位置（row2，col4）。

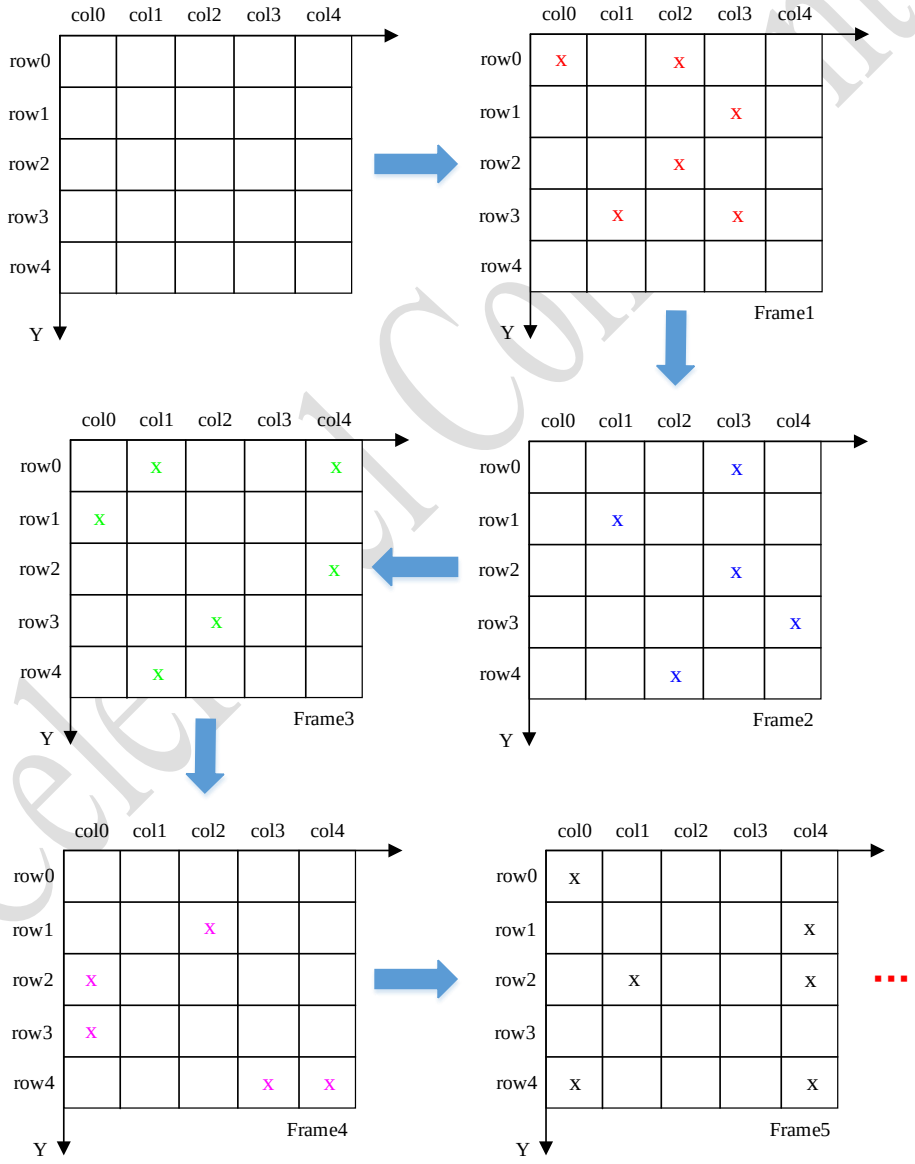


Fig. 1-16 Event 模式下的灰度图像帧（Event Gray Pic）

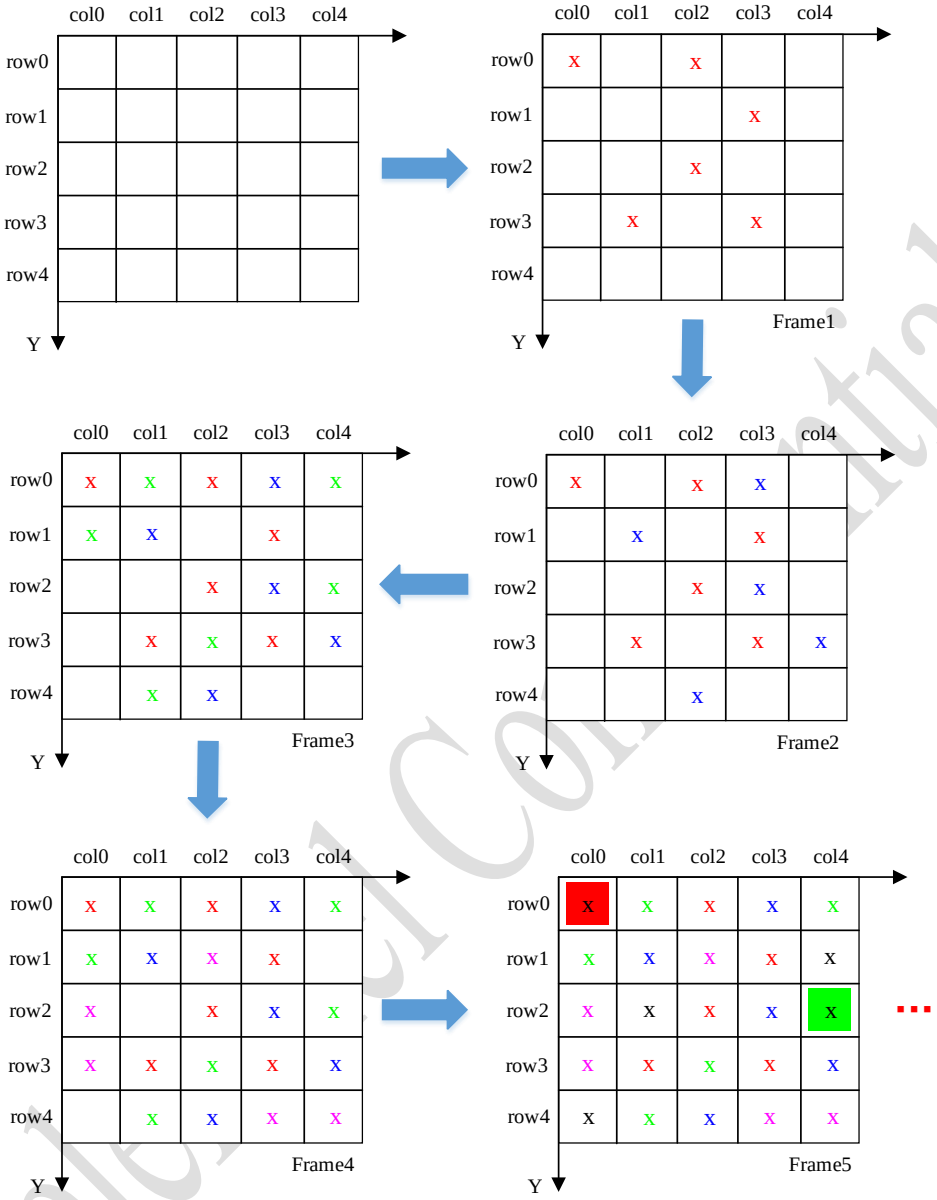


Fig. 1-17 Event 模式下的累加灰度图像帧（Event Accumulated Gray Pic）

1.3. CeleX-5 Sensor 录制的 bin 文件数据结构

1.3.1. 无 IMU 数据的 bin 文件数据结构

无 IMU 数据的 bin 文件数据结构包括文件头（Bin Header）和具体的图像数据包组成（Image Package），如图 1-18 所示。

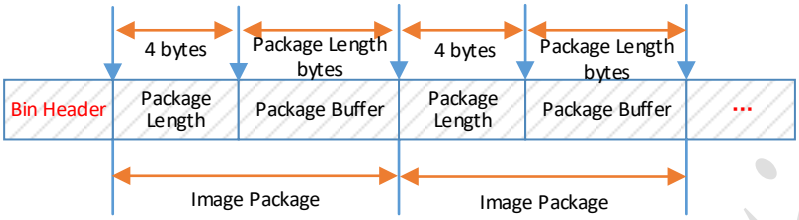


Fig. 1-18 无 IMU 数据的 bin 文件数据结构

Bin Header 的格式如下面的结构体所示，其中，*data_type* 的 bit1 用来表示是否含有 IMU 数据。

```
typedef struct BinFileAttributes
{
    uint8_t     data_type; //bit0: 0: fixed mode; 1: loop mode
                           // bit1: 0: no IMU data; 1: has IMU data
    uint8_t     loopA_mode;
    uint8_t     loopB_mode;
    uint8_t     loopC_mode;
    uint8_t     event_data_format;
    uint8_t     hour;
    uint8_t     minute;
    uint8_t     second;
    uint32_t    package_count;
} BinFileAttributes;
```

Package Length是指一个Package Buffer的大小，而Package Buffer存放的是如1.2.3.1章节所介绍的MIPI格式的数据（Event数据的默认大小为357001，Full-frame数据的默认大小为1536001）。

1.3.2. 有 IMU 数据的 bin 文件结构

有 IMU 数据的 bin 文件数据结构包括文件头（Bin Header），具体的图像数据包组成（Image Package）以及 IMU 数据，如图 1-19 所示。

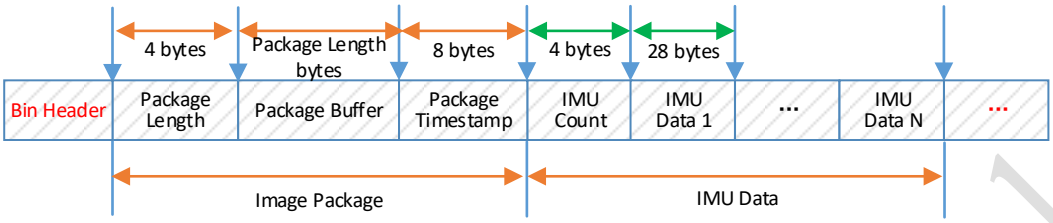


Fig. 1-19 有 IMU 数据的 bin 文件数据结构

Bin Header，Package Length，Package Buffer 的格式如上一章节所述，Package Timestamp 为收到该 Image Package 的时间戳。

IMU Data 包括 IMU 数据的个数以及每笔 IMU 数据的具体信息，其中 IMU Count 为个数，IMU Data N 为第 N 笔 IMU 数据（包括具体的 IMU 数据以及时间戳信息）。

由于用户不需要自己解析 IMU 数据，可以通过 API 接口 getIMUData 获取 IMU 数据，所以这里就不介绍 IMU Data N 的数据格式。

2. CeleX-5 API Reference

2.1. 概述

CeleX API 提供了与 CeleX-5 传感器通信的 C++接口。要使用此 API 构建应用程序，您需要将 include 目录中的文件拷贝到您的项目中，这些头文件中包含了我们提供的所有接口。同时，还需指定 CeleX.dll 和 CeleX.lib（Windows）或 libCeleX.so（Linux）（例如/ usr / local / lib）的路径。

CeleX5 封装了与 CeleX-5 传感器相关的接口，所以要使用该库，需要创建一个 CeleX5 实例，然后调用 *openSensor* 打开连接到 PC 上的传感器，可以调用 *getFullPicBuffer* 和 *getEventPicBuffer* 来获取使用原始数据计算得到的各种图像帧。

CeleX5DataManager 是一个数据通知类，重载 *onFrameDataUpdated* 方法以接收各种图像帧已经建立完成的通知。

文档中用到的结构体以及枚举值的简单介绍：

enum CeleX5Mode
{
Unknown_Mode = -1,
Event_Address_Only_Mode = 0,
Event_Optical_Flow_Mode = 1,
Event_Intensity_Mode = 2,
Full_Picture_Mode = 3,
Full_Optical_Flow_S_Mode = 4,
Full_Optical_Flow_M_Mode = 6,
};
	Unknown_Mode – 未知模式
	Event_Address_Only_Mode - 仅有地址信息的 Event 模式(*Event Address Only Mode*)
	Event_Optical_Flow_Mode - 带 Optical-flow 信息的 Event 模式(*Event Optical-flow Mode*)
	Event_Intensity_Mode - 带亮度信息的 Event 模式(*Event Intensity Mode*)
	Full_Picture_Mode - Full-frame picture 模式
	Full_Optical_Flow_S_Mode - Single full-frame optical-flow 模式
	Full_Optical_Flow_M_Mode - Multiple full-frame optical-flow 模式

enum DeviceType
{
Unknown_Devive = 0,
CeleX5_MIPI = 1,
CeleX5_OpalKelly = 2
};
	Unknown_Device - 未知设备
	CeleX5_MIPI - CeleX5 MIPI 设备
	CeleX5_OpalKelly - CeleX5 OpalKelly 设备

enum SensorAttribute { Master_Sensor = 0,	**Master_Sensor** - 主设备 **Slave_Sensor** - 从设备

Slave_Sensor = 1, };	

enum emEventPicType
{
EventBinaryPic = 0,
EventAccumulatedPic = 1,
EventGrayPic = 2,
EventCountPic = 3,
EventDenoisedBinaryPic = 4,
EventSuperimposedPic = 5,
EventDenoisedCountPic = 6
};
	EventBinaryPic - Event 二值图像帧
	EventAccumulatedPic - Event 累加灰度图像帧
	EventGrayPic - Event 灰度图像帧
	EventCountPic - Event 计数图像帧
	EventDenoisedBinaryPic - Event 四邻域去噪图像帧
	EventSuperimposedPic - Event 叠加图像帧，即将 Event 二值图像帧叠加到 Event 累加灰度图像帧上形成的图像帧
	EventDenoisedCountPic - 去噪后的 Event 计数图像帧

enum emFullPicType
{
Full_Optical_Flow_Pic = 0,
Full_Optical_Flow_Speed_Pic = 1,
Full_Optical_Flow_Direction_Pic = 2
};
	Full_Optical_Flow_Pic - Full 图像的光流图像帧
	Full_Optical_Flow_Speed_Pic - Full 光流速度图像帧
	Full_Optical_Flow_Direction_Pic – Full 光流方向图像帧

typedef struct BinFileAttributes { uint8_t data_type; uint8_t loopA_mode; uint8_t loopB_mode; uint8_t loopC_mode; uint8_t event_data_format; uint8_t hour; uint8_t minute; uint8_t second;	**data_type** - bit0: 0: fixed mode; 1: loop mode. - bit1: 0: no IMU data; 1: has IMU data. **loopA_mode** - Fixed 模式下模式或是 Loop 模式中的第 1 个 Loop 的模式 **loopB_mode** - Loop 模式中的第 2 个 Loop 的模式 **loopC_mode** - Loop 模式中的第 3 个 Loop 的模式 **event_data_format** - event 数据使用的数据格式 **hour, minute, second** – 录制的 bin 文件的时间 **package_count** - 录制的 bin 文件的 Package 总数

uint32_t package_count; } BinFileAttributes;	

typedef enum EventShowType
{
EventShowByTime = 0,
EventShowByCount = 1,
EventShowByRowCycle = 2,
};
	EventShowByTime - 按时间建帧
	EventShowByCount - 按 Event 数量建帧
	EventShowByRowCycle - 按从上至下扫描的轮回数量建帧

typedef enum PlaybackState
{
NoBinPlaying = 0,
Playing,
BinReadFinished,
PlayFinished,
Replay
};
	NoBinPlaying - 无 bin 文件播放
	Playing - 正在播放 bin 文件
	BinReadFinished - 读取 bin 文件结束
	PlayFinished - bin 文件播放结束
	Replay - bin 文件正在重播

typedef struct EventData { uint16_t col; uint16_t row; uint16_t brightness; int16_t polarity; uint32_t t; long t_pc; }EventData;	**col** - 像素所在的列 **row** - 像素所在的行 **brightness** - 像素所在的亮度值 **p** - 极性（-1：变弱，1：变强，0：不变） **t** - 像素变化的时间 t_pc – PC 世界时间戳
typedef struct IMUData { double x_GYROS; double y_GYROS; double z_GYROS; uint32_t t_GYROS; double x_ACC;	**x_GYROS** - 绕 X 轴方向旋转的偏转角速度（gyroscopes） **y_GYROS** - 绕 Y 轴方向旋转的偏转角速度（gyroscopes） **z_GYROS** - 绕 Z 轴方向旋转的偏转角速度（gyroscopes） **t_GYROS** - 偏转角速度对应的时间戳 **x_ACC** - X 轴方向的加速度（accelerometer）分量

double y_ACC; double z_ACC; uint32_t t_ACC; double x_MAG; double y_MAG; double z_MAG; uint32_t t_MAG; double x_TEMP; uint64_t frameNo; std::time_t time_stamp; } IMUData;	**y_ACC** - Y 轴方向的加速度（accelerometer）分量 **z_ACC** - Z 轴方向的加速度（accelerometer）分量 **t_ACC** - 加速度分量对应的时间戳 **x_MAG** - X 轴方向的磁力计（magnetometer）分量 **y_MAG** - Y 轴方向的磁力计（magnetometer）分量 **z_MAG** - Z 轴方向的磁力计（magnetometer）分量 **t_MAG** - 磁力计分量对应的时间戳 **frameNo** - 图像帧编号 **time_stamp** - 收到该数据包的世界时间

2.2. CeleX5DataManager Class Reference

该类用来通知图像帧数据已经处理完成，此时，可以获取 API 已经处理好的数据来做各种算法或显示。要接收这些通知，需要从此类派生自己的 *DataManager* 子类，并重载 *onFrameDataUpdated* 方法，具体的实例代码如下：

```
#include <opencv2/opencv.hpp>
#include <celex5/celex5.h>
#include <celex5/celex5datamanager.h>
#include <celex5/celex5processeddata.h>

#ifdef _WIN32
#include <windows.h>
#else
#include<unistd.h>
#endif

#define FPN_PATH       "./FPN.txt"

class SensorDataObserver : public CeleX5DataManager
{
public:
    SensorDataObserver(CX5SensorDataServer* pServer)
    {
        m_pServer = pServer;
        m_pServer->registerData(this, CeleX5DataManager::CeleX_Frame_Data);
    }
    ~SensorDataObserver()
    {
        m_pServer->unregisterData(this, CeleX5DataManager::CeleX_Frame_Data);
    }
```

```
    virtual void onFrameDataUpdated(CeleX5ProcessedData* pSensorData);//overrides Observer operation
    CX5SensorDataServer* m_pServer;
};

void SensorDataObserver::onFrameDataUpdated(CeleX5ProcessedData* pSensorData)
{
    if (NULL == pSensorData)
        return;
    CeleX5::CeleX5Mode sensorMode = pSensorData->getSensorMode();
    if (CeleX5::Full_Picture_Mode == sensorMode)
    {
        //get fullpic when sensor works in FullPictureMode
        if (pSensorData->getFullPicBuffer())
        {
            cv::Mat matFullPic(800, 1280, CV_8UC1, pSensorData->getFullPicBuffer()); //full pic
            cv::imshow("FullPic", matFullPic);
            cv::waitKey(1);
        }
    }
    else if (CeleX5::Event_No_ADC_Mode == sensorMode)
    {
        //get buffers when sensor works in EventMode
        if (pSensorData->getEventPicBuffer(CeleX5::EventBinaryPic))
        {
            cv::Mat matEventPic(800, 1280, CV_8UC1,
            pSensorData->getEventPicBuffer(CeleX5::EventBinaryPic)); //event binary pic
            cv::imshow("Event Binary Pic", matEventPic);
            cvWaitKey(1);
        }
```

```
        }
        else if (CeleX5::Event_Intensity_Mode == sensorMode)
        {
            //get buffers when sensor works in FullPic_Event_Mode
            if (pSensorData->getEventPicBuffer(CeleX5::EventBinaryPic))
            {
                cv::Mat matEventPic1(800, 1280, CV_8UC1,
                pSensorData->getEventPicBuffer(CeleX5::EventBinaryPic)); //event binary pic
                cv::Mat matEventPic2(800, 1280, CV_8UC1,
                pSensorData->getEventPicBuffer(CeleX5::EventGrayPic)); //event gray pic
                cv::imshow("Event Binary Pic", matEventPic1);
                cv::imshow("Event Gray Pic", matEventPic2);
                cvWaitKey(1);
            }
        }
}

int main()
{
        CeleX5 *pCeleX = new CeleX5;
        if (NULL == pCeleX)
            return 0;
        pCeleX->openSensor(CeleX5::CeleX5_MIPI);
        pCeleX->setFpnFile(FPN_PATH);
        CeleX5::CeleX5Mode sensorMode = CeleX5::Event_Intensity_Mode;
        pCeleX->setSensorFixedMode(sensorMode);
        SensorDataObserver* pSensorData = new SensorDataObserver(pCeleX->getSensorDataServer());

        while (true)
```

```
        {
#ifdef _WIN32
                Sleep(5);
#else
                usleep(1000 * 5);
#endif
        }
        return 1;
}
```

2.3. CeleX5 Class Reference

CeleX5 类封装跟 CeleX 第 5 代传感器有关的所有接口，如获取传感器输出的数据，切换传感器的工作模式和配置寄存器参数以控制传感器等功能。

Public Member Functions:

No.	API Name	Description
1	openSensor	打开 CeleX-5 Sensor 设备
2	isSensorReady	判断 CeleX-5 Sensor 初始化是否成功
3	getMIPIData	从 USB3.0 driver 中读取 MIPI 数据
4	getFullPicBuffer	获取 *Full-frame Picture Mode* 模式下的全幅图像帧
5	getFullPicMat	获取 *Full-frame Picture Mode* 模式下的全幅图像帧 mat
6	getEventPicBuffer	获取指定的 Event 图像类型的图像帧
7	getEventPicMat	指定输出的 Event 图像类型，获取到相应图像类型 cv::Mat 格式的图像帧
8	getOpticalFlowPicBuffer	获取 full-frame 光流图像帧
9	getOpticalFlowPicMat	获取一帧 cv::Mat 格式的光流图像
10	getEventDataVector	获取到每一帧 Event 数据的数组
11	getIMUData	获取 IMUData 数据包
12	setSensorFixedMode	设置 CeleX-5 Sensor 在单模式下（非 Loop 模式）的工作模式
13	getSensorFixedMode	获取 CeleX-5 sensor 在单模式下（非 Loop 模式）的工作模式
14	setFpnFile	设置创建图像时要使用的 FPN 文件
15	generateFPN	生成 FPN 文件
16	setClockRate	设置 CeleX-5 Sensor 的工作频率
17	getClockRate	获取 CeleX-5 Sensor 当前的工作频率，默认值 100 MHz
18	setThreshold	设置 Event 数据触发的阈值
19	getThreshold	获取已设置的触发 Event 数据的阈值大小
20	setBrightness	设置一个跟图像亮度有关的配置值的大小
21	getBrightness	获取跟图像亮度有关的配置值的大小
22	setISOLevel	设置 CeleX-5 Sensor 的 ISO 档位

23	getISOLevel	获取 CeleX-5 Sensor 当前的 ISO 档位
24	getFullPicFrameTime	获取当 CeleX-5 Sensor 工作在 *Full-frame Picture Mode* 时生成一个 full-frame 图像帧的时间
25	setEventFrameTime	设置当 CeleX-5 Sensor 工作在 Event 模式下产生一帧 Event 图像的时间
26	getEventFrameTime	获取 CeleX-5 Sensor 工作在 Event 模式时的帧长
27	setOpticalFlowFrameTime	设置当 CeleX-5 Sensor 工作在 Full-frame Optical-flow 模式下产生一帧 Optical-flow 图像的时间（包括积累时间和输出时间）
28	getOpticalFlowFrameTime	获取当 CeleX-5 Sensor 工作在 *Full-frame Optical-flow Mode* 模式下产生一帧 Optical-flow 图像的时间
29	reset	重置 Sensor 以及清除缓存里面的数据
30	pauseSensor	暂停 Sensor 的数据输出
31	restartSensor	恢复 Sensor 的数据输出
32	stopSensor	结束所有 Sensor 的数据输出
33	setSensorAttribute	设置当前设备的属性（Master/Slave）
34	getSensorAttribute	获取当前设备的属性（Master/Slave）
35	getSerialNumber	获取 CeleX-5 Sensor 的序列号
36	getFirmwareVersion	获取 CeleX-5 Sensor 的固件版本号
37	getFirmwareDate	获取 CeleX-5 Sensor 的固件发布日期
38	setEventShowMethod	设置 CeleX-5 Sensor 建帧方式
39	getEventShowMethod	获取当前 CeleX-5 Sensor 的建帧方式
40	setRotateType	设置当前图像帧显示的旋转参数
41	getRotateType	获取当前图像帧的旋转参数
42	setEventCountStepSize	设置 Event Count Pic 图像的计数步长
43	getEventCountStepSize	获取当前 Event Count Pic 图像的步长
44	setEventDataFormat	设置要使用的 Event 数据格式（MIPI）
45	getEventDataFormat	获取当前使用的 Event 数据格式（MIPI）
46	startRecording	开始录制 Sensor 的原始数据（未经任何处理的数据）
47	stopRecording	停止录制 Sensor 的数据，并停止向 bin 文件中写数据

48	openBinFile	打开一个用户指定的 bin 文件
49	readBinFileData	从打开的 bin 文件中读取指定字节长度的数据
50	getBinFileAttributes	获取到 bin 文件的相关属性
51	setRowDisabled	修改 Sensor 输出的图像的分辨率（关闭指定行的数据输出）

2.3.1 openSensor

bool CeleX5::openSensor(DeviceType type)

Parameters

[in] **type** CeleX-5 sensor 的设置类型

Returns

CeleX-5 sensor 是否成功开启的状态

该方法用于打开 CeleX-5 Sensor 设备，如果当前使用的 CeleX-5 芯片套件是支持串口输出的，则 type 传入参数为 *CeleX5_MIPI*，如果支持并口输出，则 type 传入的参数为 *CeleX5_Parallel*。关于 type 更多内容见 *DeviceType*。

```
#include <celex5/celex5.h>

{

    CeleX5 *pCeleX = new CeleX5;

    if (pCeleX == NULL)

        return 0;

    pCeleX->openSensor(CeleX5::CeleX5_MIPI);

}
```

See also

isSensorReady

2.3.2 isSensorReady

bool CeleX5::isSensorReady(int device_index)

Parameters

[in] **device_index** 设备的索引值（如果只连接单个设备，该参数可忽略，索引默认设置为 0，当连接多个设备时，则需要通过 **device_index** 来进行区分，0 表示 Master Sensor，1 表示 Slave Sensor）

Returns

CeleX-5 Sensor 的初始化状态

该方法用于判断 CeleX-5 Sensor 初始化是否成功。返回 true 表示 Sensor 已初始化成功，否则返回 false。

See also

openSensor

2.3.3 getMIPIData

void CeleX5::getMIPIData(vector<uint8_t> &buffer,
std::time_t& time_stamp_end,
vector<IMURawData>& imu_data,
int device_index)

Parameters

[out] **buffer** 存储要读的数据的 vector buffer

[out] **time_stamp_end** 用于记录获取到数据包的时间戳

[out] **imu_data** 用于存储 IMUData 数据包

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于从 USB3.0 driver 中读取 MIPI 数据。

2.3.4 getFullPicBuffer

void getFullPicBuffer(unsigned char* buffer, int device_index)

Parameters

[in] **buffer** *Full-frame Picture Mode* 模式下的 frame buffer (大小为 1280 * 800)

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于获取*Full-frame Picture Mode*模式下的全幅图像帧，只要Sensor被成功启动后，即可调用该方法，实例代码如下：

```
#include <opencv2/opencv.hpp>

#include <celex5/celex5.h>

int main()

{

    CeleX5 *pCeleX = new CeleX5;

    if (NULL == pCeleX)

        return 0;

    pCeleX->openSensor(CeleX5::CeleX5_MIPI);

    pCeleX->setFpnFile(FPN_PATH, 0);

    pCeleX->setSensorFixedMode(CeleX5::Full_Picture_Mode, 0);

    int imgSize = 1280 * 800;

    unsigned char* pBuffer1 = new unsigned char[imgSize];
```

```
    while (true)
    {
        if (sensorMode == CeleX5::Full_Picture_Mode)
        {
            //get fullpic when sensor works in Full-frame Picture Mode
            pCeleX->getFullPicBuffer(pBuffer1, 0); //full pic
            cv::Mat matFullPic(800, 1280, CV_8UC1, pBuffer1);
            cv::imshow("FullPic", matFullPic);
            cvWaitKey(10);
        }
    }
}
```

See also

getFullPicMat

2.3.5 getFullPicMat

cv::Mat CeleX5::getFullPicMat(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Full-frame Picture Mode 模式下的 frame buffer mat (cv::Mat(800, 1280))

该方法用于获取*Full-frame Picture Mode*模式下的全幅图像帧mat，它的使用方法与*getFullPicBuffer*相同，实例代码如下：

```
#include <opencv2/opencv.hpp>
#include <celex5/celex5.h>
int main()
{
    CeleX5 *pCeleX = new CeleX5;
    if (pCeleX == NULL)
        return 0;
```

```
        pCeleX->openSensor(CeleX5::CeleX5_MIPI);
        pCeleX->setFpnFile(FPN_PATH, 0);
        pCeleX->setSensorFixedMode(CeleX5:: Full_Picture_Mode, 0);
        while (true)
        {
            if (sensorMode == CeleX5::Full_Picture_Mode)
            {
                if (!pCeleX->getFullPicMat(0).empty())
                {
                    cv::Mat fullPicMat = pCeleX->getFullPicMat(0);
                    cv::imshow("FullPic", fullPicMat);
                    cv::waitKey(10);
                }
            }
        }
}
```

See also

getFullPicBuffer

2.3.6 getEventPicBuffer

void getEventPicBuffer(unsigned char* buffer, emEventPicType type, int device_index)

Parameters

[in] **buffer** Event 模式下指定图像类型的图像帧（大小为 1280 * 800）

[in] **type** Event 模式下可输出的图像类型

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于获取指定的 Event 图像类型的图像帧，它可以获取多种不同类型的 Event 图像，包括二值图、灰度图等等。更多图像类型可见 *emEventPicType*。示例代码如下：

```
#include <opencv2/opencv.hpp>
#include <celex5/celex5.h>
int main()
```

```
{
    CeleX5 *pCeleX = new CeleX5;
    if (NULL == pCeleX)
        return 0;
    pCeleX->openSensor(CeleX5::CeleX5_MIPI);
    pCeleX->setFpnFile(FPN_PATH, 0);
    pCeleX->setSensorFixedMode(CeleX5:: Event_Address_Only_Mode, 0);
    int imgSize = 1280 * 800;
    unsigned char* pBuffer1 = new unsigned char[imgSize];
    while (true)
    {
        if (sensorMode == CeleX5:: Event_Address_Only_Mode)
        {
            //get buffers when sensor works in Event Mode
            pCeleX->getEventPicBuffer(pBuffer1, CeleX5::EventBinaryPic, 0);
            cv::Mat matEventPic(800, 1280, CV_8UC1, pBuffer1);
            cv::imshow("Event-EventBinaryPic", matEventPic);
            cvWaitKey(10);
        }
    }
}
```

See also

getEventPicMat

2.3.7 getEventPicMat

cv::Mat CeleX5::getEventPicMat(emEventPicType type, int device_index)

Parameters

[in] **type** Event模式下可输出的图像类型

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

指定类型的 event frame mat（cv::Mat(800, 1280)）

该方法用于指定输出的 Event 图像类型，获取到相应图像类型 cv::Mat 格式的图像帧。该方法可以获取多种不同类型的 Event 图像，包括二值图、灰度图等等。更多图像类型可见 *emEventPicType*。

See also

getEventPicBuffer

2.3.8 getOpticalFlowPicBuffer

void CeleX5::getOpticalFlowPicBuffer(unsigned char* buffer,

emFullPicType type,

int **device_index)**

Parameters

[in] **buffer** Optical-frame buffer (大小为 1280 * 800).

[in] **type** Full Optical 模式下可输出的图像类型

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于获取 full-frame 光流图像帧，光流速度图像已经光流方向图。具体图像类型可见 *emFullPicType*。

使用实例如下：

```
#include <opencv2/opencv.hpp>

#include <celex5/celex5.h>

int main()

{

    CeleX5 *pCeleX = new CeleX5;

    if (NULL == pCeleX)

        return 0;

    pCeleX->openSensor(CeleX5::CeleX5_MIPI);

    pCeleX->setSensorFixedMode(CeleX5::Full_Optical_Flow_S_Mode, 0);

    int imgSize = 1280 * 800;

    unsigned char* pOpticalFlowBuffer = new unsigned char[imgSize];

    while (true)
```

```
    {
        //get optical-flow data when sensor works in EventMode
        //optical-flow raw data - display gray image
        pCeleX->getOpticalFlowPicBuffer(pOpticalFlowBuffer, CeleX5:: Full_Optical_Flow_Pic, 0);
        cv::Mat matOpticalRaw(800, 1280, CV_8UC1, pOpticalFlowBuffer);
        cv::imshow("Optical-Flow Buffer - Gray", matOpticalRaw);
        cvWaitKey(10);
    }
    return 1;
}
```

See also

getOpticalFlowPicMat

2.3.9 getOpticalFlowPicMat

cv::Mat CeleX5:: getOpticalFlowPicMat (emFullPicType type, int device_index)

Parameters

[in] **type** Full Optical 模式下可输出的图像类型

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

cv::Mat 格式的光流图像帧 (cv::Mat(800, 1280)).

该方法用于获取一帧 cv::Mat 格式的光流图像。具体图像类型可见 *emFullPicType*。

See also

getOpticalFlowPicBuffer

2.3.10 getEventDataVector

bool CeleX5::getEventDataVector(std::vector<EventData>& data, int device_index)

bool CeleX5::getEventDataVector(std::vector<EventData> &vector, uint64_t& frameNo, int device_index)

Parameters

[out] **data** 用于存储一帧Event数据的vector容器

[out] **frameNo** 图像帧编号

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

如果 data 不为空则返回 true，否则返回 false。

该方法用于获取到每一帧 Event 数据的数组。默认的帧长为 30 毫秒。每个 Event 数据包含行、列、亮度以及时间信息。更多解释可见 *EventData*。该方法在实时获取数据或是离线读取 bin 文件时皆可用。

2.3.11 getIMUData

int CeleX5::getIMUData(std::vector<IMUData>& data, int device_index)

Parameters

[out] **data** 获取到的 IMUData 数据包列表

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

返回实际获取到 IMUData 的个数

该方法用于获取 IMUData 数据包，关于 IMU 数据的详细解释见 *IMUData*。

2.3.12 setSensorFixedMode

void CeleX5::setSensorFixedMode(CeleX5Mode mode, int device_index)

Parameters

[in] **mode** CeleX-5 sensor 的固定模式

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方式用于设置 CeleX-5 Sensor 在单模式下（非 Loop 模式）的工作模式，包括 *Event Address-only Mode, Event Optical-flow Mode, Event Intensity Mode, Full-frame Picture Mode, Single Full-frame Optical-flow Mode and Multiple Full-frame Optical-flow Mode*。关于 mode 更多内容见 *CeleX5Mode*。

See also

getSensorFixedMode

2.3.13 getSensorFixedMode

CeleX5Mode CeleX5::getSensorFixedMode(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

CeleX-5 sensor 的固定模式.

该方式用于获取 CeleX-5 sensor 在单模式下（非 Loop 模式）的工作模式。

See also

setSensorFixedMode

2.3.14 setFpnFile

bool CeleX5::setFpnFile(const string &fpnFile, int device_index)

Parameters

[in] **fpnFile** 要使用的 FPN 文件的路径+名称

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

加载 FPN 文件成功与否的状态

该方法用于设置创建图像时要使用的 FPN 文件，返回 true 表示成功读取 FPN 文件中的数据，否则返回 false。

See also

generateFPN

2.3.15 generateFPN

void CeleX5::generateFPN(std::string fpnFile, int device_index)

Parameters

[in] **fpnFile** 生成 FPN 文件的路径+名称

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于生成 FPN 文件，固定模式噪声（FPN, Fixed Pattern Noise）是数字图像传感器上的特定噪声模式的术语，在较长的曝光镜头中经常可见，其中特定像素易于在一般背景噪声之上提供较亮的强度。

See also

setFpnFile

2.3.16 setClockRate

void CeleX5::setClockRate(uint32_t value, int device_index)

Parameters

[in] **value** CeleX-5 Sensor 的时钟工作频率，单位为 MHz

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置的 CeleX-5 Sensor 的工作频率，Sensor 默认是工作在 100 MHz，它可以调节的范围为：20 ~ 100 MHz，步长为 10。该值越大表示 Sensor 可以检测到像素点的光强变化的速度越快。

See also

getClockRate

2.3.17 getClockRate

uint32_t CeleX5::getClockRate(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

CeleX-5 Sensor 的时钟工作频率，单位为 MHz

该方法用于获取的 CeleX-5 Sensor 的工作频率，默认值 100 MHz。

See also

setClockRate

2.3.18 setThreshold

void CeleX5::setThreshold(uint32_t value, int device_index)

Parameters

[in] **value** 触发 Event 数据的阈值

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置Event数据触发的阈值，当某个像素的光强度变化超过该阈值时，则该像素可以被标记为一个Event或激活的像素。该阈值越大，则被触发的像素会越少，反之会越多，它支持调节的范围为：50~ 511。

该方法只有在 CeleX-5 Sensor 输出 Event 数据时有效，如果 sensor 工作在 *Full-frame Picture Mode*，不论该值多大，Sensor 都会输出一帧完整的图像帧。而且，由于该方法修改的是硬件的参数，所有只有在实时获取 Sensor 数据时有效，在回播 bin 文件时无效。

See also

getThreshold

2.3.19 getThreshold

uint32_t CeleX5::getThreshold(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

触发 Event 数据的阈值

该方法用于获取已设置的触发 Event 数据的阈值大小。

See also

setThreshold

2.3.20 setBrightness

void CeleX5::setBrightness(uint32_t value, int device_index)

Parameters

[in] **value** 跟图像亮度有关的配置值

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置一个跟图像亮度有关的配置值的大小，它支持调节的范围为 0 ~ 1023，默认值为 150。该值设置的越大，Sensor 输出的图像越亮，越小，图像则越暗。由于该方法修改的是硬件的参数，所有只有在实时获取 Sensor 数据时有效，在回播 bin 文件时无效。

See also

getBrightness

2.3.21 getBrightness

uint32_t CeleX5::getBrightness(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

跟图像亮度有关的配置值

该方法用户获取跟图像亮度有关的配置值的大小。

See also

setBrightness

2.3.22 setISOLevel

void CeleX5::setISOLevel(uint32_t value, int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置 CeleX-5 Sensor 的 ISO 档位，ISO 共有 6 档，默认档位是 4。

See also

getISOLevel

2.3.23 getISOLevel

uint32_t CeleX5::getISOLevel(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

当前 Sensor 设置的 ISO 档位

该方法用于获取的 CeleX-5 Sensor 的 ISO 档位，默认档位是 4。

See also

setISOLevel

2.3.24 getFullPicFrameTime

uint32_t CeleX5::getFullPicFrameTime(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

产生一个 full-frame 图像帧的时间，单位为 ms

该方法用于获取当 CeleX-5 Sensor 工作在 *Full-frame Picture Mode* 时生成一个 full-frame 图像帧的时间。

2.3.25 setEventFrameTime

void CeleX5::setEventFrameTime(uint32_t msec, int device_index)

Parameters

[in] **msec** Event 模式下的帧长，单位为 ms

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置 CeleX-5 Sensor 工作在 Event 模式时的帧长。需要注意的是，该设置并不改变硬件参数，它仅仅是改变了软件用 Event 数据建帧的时间长度。

See also

getEventFrameTime

2.3.26 getEventFrameTime

uint32_t CeleX5::getEventFrameTime(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Event 模式下的帧长，单位为 ms

该方法用于获取 CeleX-5 Sensor 工作在 Event 模式时的帧长。

See also

setEventFrameTime

2.3.27 setOpticalFlowFrameTime

void CeleX5::setOpticalFlowFrameTime(uint32_t msec, int device_index)

Parameters

[in] **msec** CeleX-5 Sensor 工作在 Full-frame Optical-flow 模式下产生一帧 Optical-flow 图像的时间，单位为 ms

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置当 CeleX-5 Sensor 工作在 Full-frame Optical-flow 模式下产生一帧 Optical-flow 图像的时间（包括积累时间和输出时间）。它修改的是硬件参数，例如，msec 为 30ms，表示 Sensor 产生一帧 Optical-flow 图像的时间为 30ms，也即帧率为 33fps。关于这个时间的更多描述，见 1.3.1.5.

See also

getOpticalFlowFrameTime

2.3.28 getOpticalFlowFrameTime

uint32_t CeleX5::getOpticalFlowFrameTime(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

当 CeleX-5 Sensor 工作在 *Full-frame Optical-flow Mode* 模式下产生一帧 Optical-flow 图像的时间，单位为 ms

该方法用于获取当 CeleX-5 Sensor 工作在 *Full-frame Optical-flow Mode* 模式下产生一帧 Optical-flow 图像的时间。

See also

setOpticalFlowFrameTime

2.3.29 reset

void CeleX5::reset (int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方式用于重置 Sensor 以及清除缓存里面的数据。

2.3.30 pauseSensor

void CeleX5::pauseSensor()

该方法用于暂停所有Sensor的数据输出。

2.3.31 restartSensor

void CeleX5::restartSensor()

该方法用于重新启动所有Sensor的数据输出。

2.3.32 stopSensor

void CeleX5::stopSensor()

该方法用于结束所有Sensor的数据输出。

2.3.33 setSensorAttribute

void CeleX5::setSensorAttribute(SensorAttribute attribute, int device_index)

Parameters

[in] **attribute** Sensor 的属性（Master/Slave）

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置 CeleX-5 Sensor 的属性，可以指定 **device_index** 设备设置成 Master 或是 Slaver。但是默认 0 表示 Master Sensor，1 表示 Slave Sensor。

See also

getSensorAttribute

2.3.34 getSensorAttribute

SensorAttribute CeleX5::getSensorAttribute(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Sensor 的属性（Master/Slave）

该方法用于获取 CeleX-5 Sensor 的属性，默认 0 表示 Master Sensor，1 表示 Slave Sensor。

See also

setSensorAttribute

2.3.35 getSerialNumber

std::string CeleX5::getSerialNumber (int device_index)
Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Sensor 的序列号

该方法用于获取设备的序列号。

See also

getFirmwareVersion

getFirmwareDate

2.3.36 getFirmwareVersion

std::string CeleX5::getFirmwareVersion (int device_index)
Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Sensor 的固件版本号

该方法用于获取设备的固件版本号。

See also

getSerialNumber

getFirmwareDate

2.3.37 getFirmwareDate

std::string CeleX5::getFirmwareDate(int device_index)
Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Sensor 的固件发布时间

该方法用于获取设备的固件发布日期。

See also

getSerialNumber

getFirmwareVersion

2.3.38 setEventShowMethod

void CeleX5::setEventShowMethod(EventShowType type, int value, int device_index)
Parameters

[in] **type** 图像建帧的方式

[in] **value** 不同建帧方式对应的参数

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置当前图像建帧的方式，目前有按时间建帧、按 Event 数量建帧以及按顺序扫描轮回数（Sensor 从上至下顺序扫描）三种方式。对应 value 参数的设置分别是时间（us）、数量以及扫描的轮回的次数。

See also

getEventShowMethod

2.3.39 getEventShowMethod

EventShowType CeleX5::getEventShowMethod(int device_index)
Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

当前图像建帧的方式

该方法用于获取 Sensor 当前图像建帧的方式。

See also

setEventShowMethod

2.3.40 setRotateType

void CeleX5::setRotateType(int type, int device_index)

Parameters

[in] **type** 旋转的类型

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置图像的旋转类型。可用于实现图像的上下左右翻转。

See also

getRotateType

2.3.41 getRotateType

int CeleX5::getRotateType(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

旋转类型

该方法用于获取当前图像的旋转类型。

See also

setRotateType

2.3.42 setEventCountStepSize

void CeleX5::setEventCountStepSize(uint32_t size, int device_index)

Parameters

[in] **size** 计数步长

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于设置 Event Count Pic 图像的计数步长。由于有些位置 Event 变化次数较少，通过设置每次变化的步长可以获得更好的可视图像。

See also

getEventCountStepSize

2.3.43 getEventCountStepSize

uint32_t CeleX5::getEventCountStepSize(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

计数步长

该方法用于获取当前 Event Count Pic 的计数步长。

See also

setEventCountStepSize

2.3.44 setEventDataFormat

void CeleX5::setEventDataFormat(int format, int device_index)

Parameters

[in] **format** 要使用的 Event 数据格式（MIPI）

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方式用于设置要使用的 Event 数据格式（MIPI）。

See also

getEventDataFormat

2.3.45 getEventDataFormat

int CeleX5::getEventDataFormat(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

Event 数据格式（MIPI）

该方法用于获取当前使用的 Event 数据格式（MIPI）。

See also

setEventDataFormat

2.3.46 startRecording

void CeleX5::startRecording(std::string filePath, int device_index)

Parameters

[in] **filePath** 存放录制的 bin 文件的路径

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

该方法用于开始录制 Sensor 的原始数据（未经任何处理的数据），并存储在用户指定的文件中。录制时 Sensor 工作在什么模式，那么录制的 bin 文件就是该模式的数据。

See also

stopRecording

2.3.47 stopRecording

void CeleX5::stopRecording()

该方法用于停止录制 Sensor 的数据，并停止向 bin 文件中写数据。

See also

startRecording

2.3.48 openBinFile

bool CeleX5::openBinFile(string filePath, int device_index)

Parameters

[in] **filePath** 要回播的 bin 文件的路径+名称

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

是否成功打开要回播的 bin 文件的状态

该方法用于打开用户指定的单个或两个 bin 文件，成功打开文件返回 true，否则返回 false。

See also

readBinFileData

2.3.49 readBinFileData

bool CeleX5::readBinFileData(int device_index)

Parameters

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

是否已经读到文件结尾

该方法用于从打开的 bin 文件中读取指定字节长度的数据，并存入 API 内部的数据缓存中，如果已经读到文件末尾处，则返回 true，否则返回 false。需要注意的是，调用该接口之前，要先调用 *openBinFile* 接口打开一个 bin 文件。

See also

openBinFile

2.3.50 getBinFileAttributes

BinFileAttributes CeleX5:: getBinFileAttributes(int device_index)

Parameters

[in] **binFile** bin文件的路径

[in] **device_index** 设备的索引值，0 表示 Master Sensor，1 表示 Slave Sensor

Returns

bin 文件属性的结构体

调用此方法可以获取到当前播放的 bin 文件的相关属性。主要包括 bin 文件的录制时长、bin 文件的类型、文件的长度。更多关于 bin 文件的属性返回值可见 *BinFileAttributes*。

2.3.51 setRowDisabled

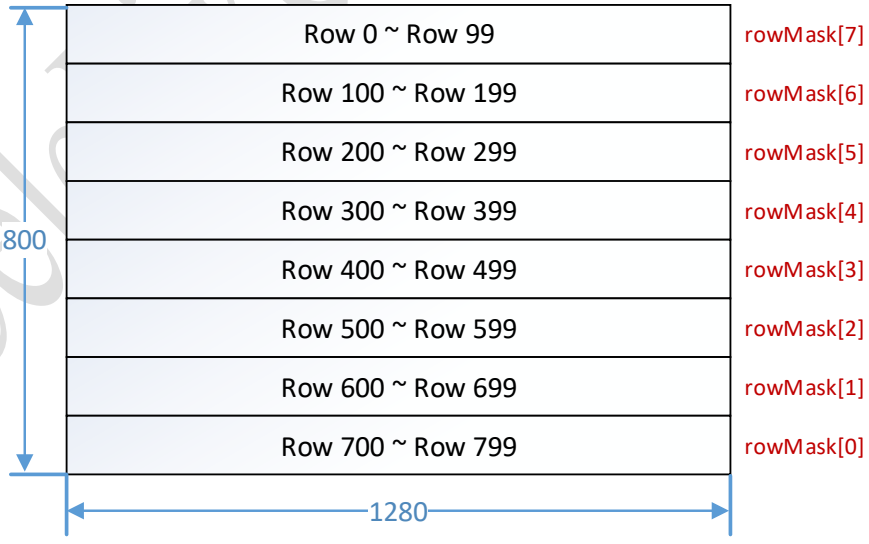
void setRowDisabled(uint8_t rowMask);

Parameters

[out] **rowMask** 要关闭指定行数据输出的控制位参数

该方法用于修改 Sensor 输出的图像的分辨率，即关闭指定行的数据输出。

如下图所示，Sensor 默认输出的图像分辨率为 1280*800，它可以按行被分割成 8 个区块，每一区块的大小都是 1280*100，且每个区块都有对应的控制位（从上到下分别为 rowMask[7]到 rowMask[0]）。例如，要关闭最上面的 100 行，那么只需将 rowMask[7]设置为 1，其他的控制位都设置为 0，即 rowMask = 128（b'10000000）。



3. Appendix

Table 3-1: Sensor Operation Mode Control Parameters

Addr	Name	Width	Default	Note
50	SWITCH_RESET_GAPA	[7:0]	200	Set gap time during mode switching
51	SWITCH_RESET_GAPB	[7:0]	250	Set gap time during mode switching
52	SWITCH_RESET_GAPC	[7:0]	200	Set gap time during mode switching
53	SENSOR_MODE_1	[2:0]	0	The operation mode in fixed mode, or the first operation mode in loop mode
54	SENSOR_MODE_2	[2:0]	3	The second operation mode in loop mode
55	SENSOR_MODE_3	[2:0]	4	The third operation mode in loop mode
57	EVENT_DURATION	[7:0]	20	Duration of event mode when sensor operates in loop mode -- Low byte
58	EVENT_DURATION	[1:0]	0	Duration of event mode when sensor operates in loop mode -- High byte
59	PICTURE_NUMBER_1	[7:0]	1	Number of pictures to acquire in Mode_D
60	PICTURE_NUMBER_2	[7:0]	1	Number of pictures to acquire in Mode_E
61	PICTURE_NUMBER_3	[7:0]	1	Number of pictures to acquire in Mode_F
62	PICTURE_NUMBER_4	[7:0]	3	Number of pictures to acquire in Mode_G
63	PICTURE_NUMBER_5	[7:0]	3	Number of pictures to acquire in Mode_H
64	SENSOR_MODE_SELECT	[0]	0	Sensor operation mode select: 0: fixed mode / 1: loop mode

Table 3-2: Sensor Data Transfer Parameters

Addr	Name	Width	Default	Note
70	EXTERNAL_DATA	[5:0]	0	Data from external sensor --- high byte
71	EXTERNAL_DATA	[7:0]	0	Data from external sensor --- middle byte
72	EXTERNAL_DATA	[7:0]	0	Data from external sensor --- low byte
73	EVENT_PACKET_SELECT	[1:0]	2	Event packet format select
74	MIPI_PIXEL_NUM_EVENT	[7:0]	254	Number of pixels in one row at event

				mode = 4*(this value+1)
75	ADC_RESOLUTION_SEL	[0]	1	Sensor ADC resolution select: 1: 12bit / 0: 8bit
76	MIPI_PIXEL_NUM_FRAME	[6:0]	5 (fixed)	Number of pixels in one row at full-frame mode --- high byte
77	MIPI_PIXEL_NUM_FRAME	[7:0]	0 (fixed)	Number of pixels in one row at full-frame mode --- low byte
78	MIPI_DATA_SOURCE_SELECT	[0]	1	For event mode, MIPI data source select: 1: internal data / 0: external data
79	MIPI_ROW_NUM_EVENT	[7:0]	0	Number of rows in one frame at event mode --- high byte
80	MIPI_ROW_NUM_EVENT	[7:0]	200	Number of rows in one frame at event mode --- low byte
81	MIPI_VIR_CHANNEL_ID	[5:0]	0	MIPI parameter, virtual channel ID
82	MIPI_HD_GAP_FULLFRAME	[2:0]	2	In full-frame mode, the interval between the last long packet and EOF -- high byte
83	MIPI_HD_GAP_FULLFRAME	[7:0]	132	In full-frame mode, the interval between the last long packet and EOF -- low byte
84	MIPI_HD_GAP_EVENT	[2:0]	2	In event mode, the interval between the last long packet and EOF -- high byte
85	MIPI_HD_GAP_EVENT	[7:0]	89	In event mode, the interval between the last long packet and EOF -- low byte
86	MIPI_GAP_EOF_SOF	[2:0]	0	High byte
87	MIPI_GAP_EOF_SOF	[7:0]	100	Low byte