

Raw Data Interface, Quick documentation

Changes to Version 3.65.1

- The “struct RawData” is now “class RawDataBlock”
- The data type is now unsigned int
- Instead of .numInts you have to use .GetLength()
- Instead of .data.get() you have to use .GetPointer()

Functions

```
void CTimeTag::StartTimetagsRaw();
```

Use this function before using ReadTagsRaw();

```
void CTimeTag::StopTimetagsRaw();
```

Please use this function after using the raw interface.

```
RawDataBlock CTimeTag::ReadTagsRaw();
```

This function returns a struct RawDataBlock. This structure encapsulates the pointer, length and other useful information.

When no data is present, the length will be 0.

The amount of data depends on the data rate. At large data rate, the length will be 32 Mtags; At slow data rates the data will span about 100 ms.

This function can be called by multiple threads. Please note, that the other functions in the CTimeTag class are not thread safe.

RawDataBlock

This class encapsulates the address and length or the timedata in one data structure and combines it with some other useful information.

```
//This are the two main functions
unsigned int *GetPointer(); //This returns the pointer to the actual data
int GetLength();           //This is the number of 32 bit tags in the data

//This are support functions
int GetTimeBits();         //This returns 25 on TDM-800
int GetTimeMask();        //This always returns 0x01ffff on TDM-800
int FreeMemory();          //Please ignore, only needed for Python interface
```

Memory Management

When the RawDataBlock goes out of scope, the memory is marked free and will be reused by the DMA controller.

You can call FreeMemory() when

Multitasking Considerations

The system is designed to have one thread in on one RawDataBlock. If you want, you can have multiple threads working on the same data. In this case we recommend to make a copy or the RawDataBlock. The memory is reused only when **all** copies go out of scope.

TT32Reader

The class TT32Reader is added to your convenience. You can use it but you don't have to. It makes the code easier to read and write. Since the functions are declared inline, the execution is very fast.

```
TT32Reader(RawDataBlock &b); // It is instantiated by giving it a RawDataBlock
bool ReadNextTag(int & channel, int & time); // Returns the next tag in channel and time.
Returns false when there was no tag to read
bool ReadNextTime(int & channel, int & time); //Returns only real time tags, no special tags
```

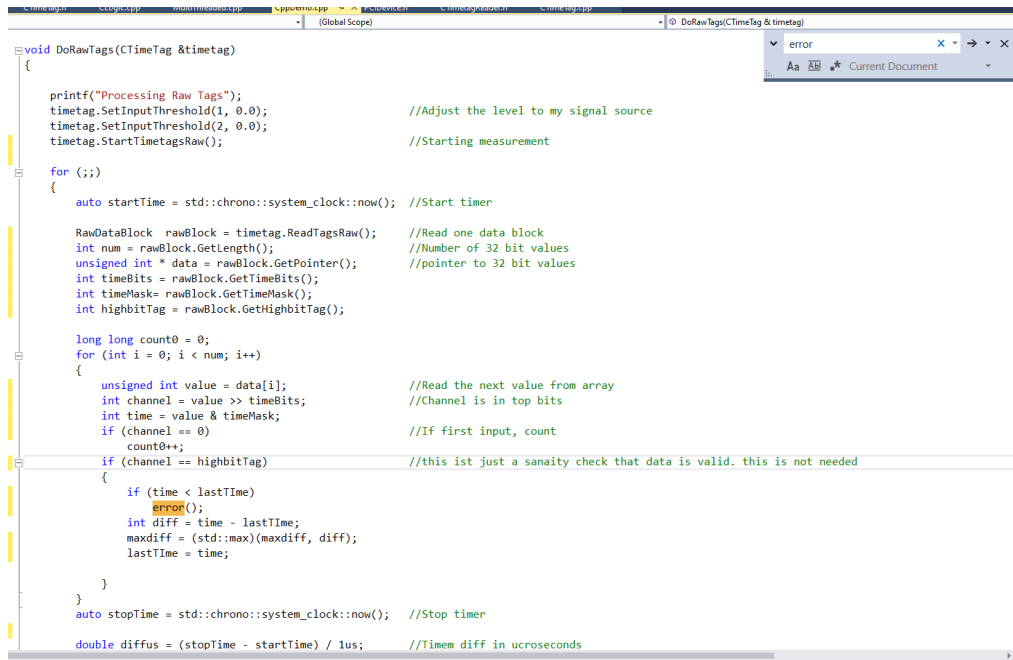
This is an example how TT32Reader can be used:

```
void ProcessBlock(RawDataBlock raw)
{
    TT32Reader reader(raw);
    int channel;
    int time;
    while (reader.ReadNextTag(channel, time))
    {
        if (channel == description.input)
            result.singleCount++;
        if (channel == raw.GetHighbitTag())
            CheckHighbit(time);
    }
}
```

Demo Program

This is a simple demo program. It can be found in CppDemo.cpp

It counts the frequency of input 1.



```
void DoRawTags(CTimeTag &timetag)
{
    printf("Processing Raw Tags");
    timetag.SetInputThreshold(1, 0.0); //Adjust the level to my signal source
    timetag.SetInputThreshold(2, 0.0);
    timetag.StartTimeTagsRaw(); //Starting measurement

    for (;;)
    {
        auto startTime = std::chrono::system_clock::now(); //Start timer

        RawDataBlock rawBlock = timetag.ReadTagsRaw(); //Read one data block
        int num = rawBlock.GetLength(); //Number of 32 bit values
        unsigned int * data = rawBlock.GetPointer(); //pointer to 32 bit values
        int timeBits = rawBlock.GetTimeBits();
        int timeMask = rawBlock.GetTimeMask();
        int highbitTag = rawBlock.GetHighbitTag();

        long long count0 = 0;
        for (int i = 0; i < num; i++)
        {
            unsigned int value = data[i]; //Read the next value from array
            int channel = value >> timeBits; //Channel is in top bits
            int time = value & timeMask;
            if (channel == 0) //If first input, count
                count0++;
            if (channel == highbitTag) //this ist just a sanity check that data is valid. this is not needed
            {
                if (time < lastTime)
                {
                    error();
                    int diff = time - lastTime;
                    maxdiff = (std::max)(maxdiff, diff);
                    lastTime = time;
                }
            }
        }
        auto stopTime = std::chrono::system_clock::now(); //Stop timer

        double diffus = (stopTime - startTime) / 1us; //Time diff in ucroseconds
    }
}
```

This is the expected output when 100 MHz on input 1 and input 2.

```
C:\Projekte\PCI\Software\Work\Visu\CppDemo\Debug\CppDemo.exe
tags: 33554432 k count0 : 16776280 time 168 ms, Mhz = 99.8 maxdiff 1
tags: 33554432 k count0 : 16776284 time 168 ms, Mhz = 100.0 maxdiff 1
tags: 33554432 k count0 : 16776288 time 169 ms, Mhz = 99.5 maxdiff 1
tags: 33554432 k count0 : 16776230 time 166 ms, Mhz = 100.8 maxdiff 1
tags: 33554432 k count0 : 16776240 time 168 ms, Mhz = 99.8 maxdiff 1
tags: 33554432 k count0 : 16776264 time 169 ms, Mhz = 99.4 maxdiff 1
tags: 33554432 k count0 : 16776278 time 167 ms, Mhz = 100.2 maxdiff 1
tags: 33554432 k count0 : 16776192 time 166 ms, Mhz = 101.1 maxdiff 1
tags: 33554432 k count0 : 16776276 time 168 ms, Mhz = 100.1 maxdiff 1
tags: 33554432 k count0 : 16776308 time 169 ms, Mhz = 99.1 maxdiff 1
tags: 33554432 k count0 : 16776202 time 166 ms, Mhz = 100.9 maxdiff 1
tags: 33554432 k count0 : 16776288 time 169 ms, Mhz = 99.5 maxdiff 1
tags: 33554432 k count0 : 16776242 time 166 ms, Mhz = 101.1 maxdiff 1
tags: 33554432 k count0 : 16776224 time 168 ms, Mhz = 99.8 maxdiff 1
tags: 33554432 k count0 : 16776278 time 169 ms, Mhz = 99.3 maxdiff 1
tags: 33554432 k count0 : 16776254 time 168 ms, Mhz = 100.0 maxdiff 1
tags: 33554432 k count0 : 16776220 time 167 ms, Mhz = 100.7 maxdiff 1
tags: 33554432 k count0 : 16776302 time 168 ms, Mhz = 99.8 maxdiff 1
tags: 33554432 k count0 : 16776194 time 167 ms, Mhz = 100.2 maxdiff 1
tags: 33554432 k count0 : 16776304 time 167 ms, Mhz = 100.7 maxdiff 1
tags: 33554432 k count0 : 16776208 time 169 ms, Mhz = 99.2 maxdiff 1
tags: 33554432 k count0 : 16776286 time 166 ms, Mhz = 101.1 maxdiff 1
tags: 33554432 k count0 : 16776226 time 168 ms, Mhz = 100.0 maxdiff 1
tags: 33554432 k count0 : 16776266 time 168 ms, Mhz = 99.9 maxdiff 1
tags: 33554432 k count0 : 16776222 time 167 ms, Mhz = 100.4 maxdiff 1
tags: 33554432 k count0 : 16776264 time 168 ms, Mhz = 99.8 maxdiff 1
tags: 33554432 k count0 : 16776278 time 167 ms, Mhz = 100.3 maxdiff 1
tags: 33554432 k count0 : 16776230 time 168 ms, Mhz = 99.8 maxdiff 1
tags: 33554432 k count0 : 16776230 time 167 ms, Mhz = 100.4 maxdiff 1
tags: 33554432 k count0 : 16776270 time 169 ms, Mhz = 99.4 maxdiff 1
```

Multithreading Framework

The Multithreading Framework is also a system you can use if you like but you don't have to.

It makes it easier to process time-tags in several threads. The multithreading framework does all synchronisation for you. Also memory management, starting and stopping of the threads and proper cleanup is done by the framework.

It is contained in the CppDemo project, in the files Multithreaded.cpp and Multithreading.h. The multithreading framework is provided in source code. Copy it or change it and use it however you like.

WorkDescription

This struct is used to parameterise your function. Every worker thread gets a reference to this struct.

Here you could add channel numbers, coincidence times and so on. Whatever is relevant to your application.

```
struct ExampleWorkDescription
{
    int input;
    //Here you can add the description of the work to do.
};
```

WorkResult

This struct contains the result of processing. This approach is only useful, when your data can be added as in a histogram or coincidence counters.

```
struct ExampleWorkResult
{
    long totalCount = 0;
    long singleCount = 0;
    // Add your variables here

    void operator += (ExampleWorkResult &other)
    {
        totalCount += other.totalCount;
        singleCount += other.singleCount;
        //Add two results together. The lock is done outside.
    }
};
```

TimetagWorker

```
class SimpleWorkerThread :public TimetagWorker
{
public:
    SimpleWorkerThread(CTimeTag & timetag, ExampleWorkDescription &description) :
        TimetagWorker(timetag),
        description(description)
    {
    }
private:
    ExampleWorkDescription &description;
    int lastTime = 0;
    //Add your variables here. Every worker thread has its own copy

    //This is your main function. You get a RawDataBlock and have to produce a result
    //Do not loop for extendet amount of time without checking the abort flag.
}
```

ProcessDataBlock

This is the actual processing. Use this function and insert whatever you need in your application. It gets a RawDataBlock and produces a WorkResult.

```
virtual void ProcessDataBlock(RawDataBlock raw) override
{
    ExampleWorkResult result;
    TT32Reader reader(raw);
    int channel;
    int time;
    while (reader.ReadNextTag(channel, time))
    {
        //include your actual algorithm here
        result.totalCount++;
        if (channel == description.input)
            result.singleCount++;
        if (channel == raw.GetHighbitTag())
            CheckHighbit(time);
    }
    //The result is combined with the other threads
    lock_guard<mutex> l(ResultMutex);
    totalResult += result;
    //The functino returns after processing one block
}
```