EARLY DRAFT

Graph-Interpreter

a scheduler of DSP/ML nanoApps

Contents

[Use-cases 1](#_Toc170147185)

[Overview 3](#_Toc170147186)

[Implementation 4](#_Toc170147187)

[Graph Text 5](#_Toc170147188)

[Syntax of graphs 6](#_Toc170147189)

[Control of the scheduler 6](#_Toc170147190)

[Platform manifest 15](#_Toc170147191)

[Paths 15](#_Toc170147192)

[Digital manifests 15](#_Toc170147193)

[List IO manifest files 16](#_Toc170147194)

[List of nodes manifests 16](#_Toc170147195)

[Digital manifest 17](#_Toc170147196)

[Graph interfaces manifests 18](#_Toc170147197)

[Options syntax 18](#_Toc170147198)

[IO manifest Header 18](#_Toc170147199)

[IO configuration 18](#_Toc170147200)

[Domain-specific IO configuration 19](#_Toc170147201)

# Use-cases

**Graph-Interpreter** is a scheduler of **DSP/ML nanoApps** designed with three objectives:

1. **Accelerate time to market**

Made for system integrators and OEM who develop complex DSP/ML stream processing. Helps going fast from prototypes validated on a computer to final tuning steps on production boards, by loading a graph of computing nodes without device recompilation.

2. **NanoApps repositories**

Provide to the nodes an opaque interface of the platform memory hierarchy. Arranges the data flow is translated in the node desired formats. Prepares the conditions where node (nanoApps) will be delivered in a protected binary format.

3. **Portability, scalability**

Use the same stream-based processing methodology from devices using 1kBytes of internal RAM to multiprocessor heterogeneous architectures. Nodes can be produced in various programming languages. The Graph are portable when interpreted on another platform.

**Use-case examples :**

1. **Tuning the IO interfaces of closed embedded systems**

The graph implements a cascade of DSP/ML algorithms applied on analog sensors. The results are sent to the main application. The system integration consists in tuning the levels. A node is in charge of rescaling and resampling the input data rate for the next computing nodes in the graph. The system integrator can update the rescaler node and use a debug script to trigger a GPIO based on level detection.

2. **Tuning the algorithms of closed embedded systems**

A node incorporates filters and detection thresholds, the system integrator updates the parameters of the nodes without recompilation. The memory mapping of the node can also be tuned. The task dispatching between processors can be tuned for performance optimization, without code recompilation.

# Overview

An example of computing graph is given in the picture below. The “**nodes**” (or called “**nanoApps**”) are processing data provided through “**arcs**”. Each arc’s stream is characterized by its conveyed **data format** (raw format, number of channels, interleaving options, time-stamps, sampling-rate, frame size).

A computer generated diagram of a program

Description automatically generated with medium confidence

Graph of nodes for stream-based computing

The nodes at the boundary of the graph are called “**IOs**” (as Input / Output ports). The IOs are characterized by the physical **domain** of operation (connexion the application, image stream, audio, motion sensor, GPIO), the commander / servant protocol used with the platform **AL** (platform Abstraction Layer. When IOs are exchanging data they do it with a list of functions with an index named **IO\_AL\_idx**.

A **platform** is characterized by the list of IO interfaces the graph can use with the IO\_AL\_idx indexes, by the list of processors and their **memory banks** pre-reserved for the execution of the graph, by the list of the nodes pre-installed before graph execution. The details of a platform, nodes and IOs are recorded in respective “**manifests**”.

A graph description includes the **presets** and **parameters** of the nodes to use at graph reset time. When the parameter and states needs to be exchanged dynamically during the graph execution a **script** (picture) can be coupled before/after the execution of each node. The scripts consist in a compact **byte-code** language. A **global script** can be used for the interactions with the application (specific parameter settings during use-case transitions). A graph can be reused as a **sub-graph** of a more complex graph.

# Implementation

The graph to be interpreted is coded as a **binary graph** resulting from the “compilation” of the graph in text format (picture below). The graph compiler is a tool executed offline, it receives the characteristics of the platform, nodes and IOs in manifests.

**Graph Text**

**+**

**nanoApp manifests**

**Platform manifest**

**Graph interfaces manifests**

**List of installed nanoApps**

**List of app’s callbacks**

**Embedded scripts**

**Compact binary format of the graph**

1. **Header**.

2. **IO-interfaces**.

3. **Data and stream formats**.

4. **Scripts** of byte codes.

5. **nanoApps and parameters**

6. **Buffer description** of arcs.

7. **Memory sections**.

**GRAPH COMPILATION**

**GRAPH EXECUTION**

**Graph interpreter of the binary graph**

1. **Parse the graph IOs**

2. **Parse the graph of nanoApps**

3. lock the execution

4. **Update the FIFO** R/W

5. **Return** to application

The Graph Interpreter is using two interfaces : one from the application and one to notify data move on IOs are finished.

The first interface is : void arm\_graph\_interpreter (uint32\_t command, arm\_stream\_instance\_t \*instance, uint8\_t \*data, uint32\_t size)

Where “**command**” tells to reset the graph, execute it, check boundary IOs filling state to move data in/out, set parameters. **Instance** is the memory allocated for the execution of the graph : a structure of pointers to the binary graph, to the installed nodes, to the AL stream interfaces functions (indexed with IO\_AL\_idx), some debug control fields.

The second interface is a call-back used to notify the end data moves with IOs :

void arm\_graph\_interpreter\_io\_ack (uint8\_t IO\_AL\_idx, uint8\_t \*data, uint32\_t data\_size)

The parameters of this function tell the “data” pointer with an amount of “size” bytes have been exchanged on the graph boundary with the AL interface indexed by IO\_AL\_idx. This function will read the binary graph information to find which **arc circular buffer descriptor** needs to be updated with this data move.

## Graph Text

The graph text ends with the arcs between nodes. Before this it describes the list of stream formats, the graph boundaries and the parameter to set during the initialization of the nodes.

In the example below a stream of data from an A/D converter is filtered through a bandpass filter, the result is used for a signal detector. The graph syntax for declaring the detector node is “node arm\_stream\_detector 0”, the second field is the name of the node declared in the platform manifest and “0” is the instance index of the node in case several detector are used in the application.

The next paragraph details the syntax to use for the graph creation.

The compilation result is on the left column : a list of 32-bits integers using the C array syntax:

* a graph description header (words 0..6)
* the setting of the graph interface at initialization time
* the list of nodes and their parameters used at initialization time
* the graph IO parameters and state
* the stream format used
* the FIFO “descriptors” of the buffers used by the arcs (address and read/write indexes)

|  |  |
| --- | --- |
| ;----------------------------------------------------------------------  ; Stream-based processing using a graph interpreter :  ;  ; - The ADC detection is used to toggle a GPIO  ;  ; +----------+ +--------+ +--------+ +--------+  ; | ADC +-----> filter1+------>detect +-----> GPIO |  ; +----------+ +--------+ +--------+ +--------+  ;  ;----------------------------------------------------------------------  set\_file\_path 0 ./  set\_file\_path 1 ../../stream\_graph/  ;----------------------------------------------------------------------  format 0  format\_frame\_length 8  ;----------------------------------------------------------------------  stream\_io 0  stream\_io\_hwid 2 ; io\_platform\_analog\_sensor\_0.txt  stream\_io 1  stream\_io\_hwid 7 ; io\_platform\_gpio\_out\_0.txt  ;----------------------------------------------------------------------  node arm\_stream\_filter 0  node\_preset 1 Q15 filter  node\_map\_hwblock 1 5 TCM = VID5  node\_parameters 0 TAG = "all parameters"  1 u8; 2 Two biquads  1 u8; 0 postShift  5 s16; 9315 14928 9315 0 -0  5 s16; 9315 5736 9315 0 -0  \_end\_  ;---------------------------------------------------------------------  node arm\_stream\_detector 0  node\_preset 3 detector setting  ;---------------------------------------------------------------------  arc\_input 0 arm\_stream\_filter 0 0 0  arc\_output 1 arm\_stream\_detector 0 1 0  arc arm\_stream\_filter 0 1 0 arm\_stream\_detector 0 0 0 | //--------------------------------------  // DATE Mon Jun 24 14:42:00 2024  // AUTOMATICALLY GENERATED CODES  // DO NOT MODIFY !  //--------------------------------------  0x00000030, // ------- words in the graph  0x00000000, // 000 000 [0] Destination in RAM 0, and RAM split 0  0x00000041, // 004 001 [1] Number of IOs 2, Formats 1, Scripts 0  0x00000015, // 008 002 [2] LinkedList size 21, Collision table 0, Arc debug 0  0x00000003, // 00C 003 [3] Nb arcs 3 SchedCtrl 0 ScriptCtrl 0  0x00000001, // 010 004 [4] Processors allowed  0x00000000, // 014 005 [5] memory consumed 0,1,2,3  0x00000000, // 018 006 [6] memory consumed 4,5,6,7  0x00000000, // 01C 007 settings of io(graph 0) 2 arc 0 set0\_copy1=0 rx0tx1=0  0x00000000, // 020 008 settings of io(graph 1) 7 arc 1 set0\_copy1=0 rx0tx1=1  0x00C04807, // 024 009 ----- arm\_stream\_filter(0) idx:7 Nrx 1 Ntx 1 ArcFmt 1 lockArc 1  0x08020000, // 028 00A ARC 0 Rx0Tx1 0 dbgpage0 -- ARC 2 Rx0Tx1 1 dbgpage0  0x100000D8, // 02C 00B Reserved static memory bank(0) = bank 0 stat0work1ret2 = 0  0x0000004C, // 030 00C size 76  0x12000000, // 034 00D Scratch memory bank(1) = bank 2 stat0work1ret2 = 1  0x00000008, // 038 00E size 8  0x01000007, // 03C 00F ParamLen 6+1 Preset 1 Tag0ALL 0  0x24630002, // 040 010 (0)  0x24633A50, // 044 011 (1)  0x00000000, // 048 012 (2)  0x16682463, // 04C 013 (3)  0x00002463, // 050 014 (4)  0x00000000, // 054 015 (5)  0x00404808, // 058 016 ----- arm\_stream\_detector(0) idx:8 Nrx 1 Ntx 1 ArcFmt 0 lockArc 1  0x08010002, // 05C 017 ARC 2 Rx0Tx1 0 dbgpage0 -- ARC 1 Rx0Tx1 1 dbgpage0  0x10000118, // 060 018 Reserved static memory bank(0) = bank 0 stat0work1ret2 = 0  0x00000034, // 064 019 size 52  0x10000148, // 068 01A Reserved static memory bank(1) = bank 0 stat0work1ret2 = 2  0x00000020, // 06C 01B size 32  0x03000001, // 070 01C ParamLen 0+1 Preset 3 Tag0ALL 0  0x000003FF, // 074 01D ------------------- vvvvvvvvv RAM vvvvvvvvv  0x00201000, // 078 01E IO(graph0) 2 arc 0 set0\_copy1=0 rx0tx1=0 servant1 1 shared 0 domain 0  0x00701801, // 07C 01F IO(graph1) 7 arc 1 set0\_copy1=0 rx0tx1=1 servant1 1 shared 0 domain 0  0x00000008, // 080 020 Format 0 frameSize 8  0x00004400, // 084 021 nchan 1 raw 17  0x00000000, // 088 022 domain-dependent  0x00000000, // 08C 023 domain-dependent  0x000000C0, // 090 024 ARC descriptor(0) Base C0h (30h words) fmtProd 0  0x00000008, // 094 025 Size 8h fmtCons 0  0x00C00000, // 098 026  0x00400000, // 09C 027  0x000000C8, // 0A0 028 ARC descriptor(1) Base C8h (32h words) fmtProd 0  0x00000008, // 0A4 029 Size 8h fmtCons 0  0x00C00000, // 0A8 02A  0x00400000, // 0AC 02B  0x000000D0, // 0B0 02C ARC descriptor(2) Base D0h (34h words) fmtProd 0  0x00000008, // 0B4 02D Size 8h fmtCons 0  0x00C00000, // 0B8 02E  0x00400000, // 0BC 02F |

# Syntax of graphs

## Control of the scheduler

## 

|  |  |  |
| --- | --- | --- |
| Tags | Parameters (type) | Comments |
| graph\_location | int | 0: destination of the binary graph is in RAM (default)  1: keep the graph in Flash and copy in RAM the portion starting from the node linked-list  2: keep the graph in Flash and copy in RAM the portion starting from the arc descriptors  3: the graph is already in RAM provided by the application |
| debug\_script\_fields | int | LSB set means "call the debug script before each nanoAppsRT is called"  bit 1 (2) set means "call the debug script after each naonoAppsRT is called"  bit 2 (4) set means "call the debug script at the end of the loop"  bit 3 (8) set means "call the debug script is called when starting the graph scheduling"  bit 4 (16) set means "call the debug script is called when returning of the graph scheduling"  no bit is set (default) the debug script is not called (default 0) |
| scheduler\_return | int | 1: return to caller after each SWC calls  2: return to caller once all SWC are parsed  3: return to caller when all SWC are starving (default 3) |
| allowed\_processors | int | bit-field of the processors alloed to execute this graph, (default = 1 main processor) |
| graph\_map\_hwblock | 0 | index of the memory block VID indexes where to map the graph. Default VID's is 0 internal RAM |
| set\_file\_path | Int String | index and its file path, used for sub graphs and scripts |

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
| format | int | index used to start the declaration of a new format |
| format\_raw\_data | int | raw data of this format (17 : S16 is the default) |
| format\_frame\_length | int | frame length in number of bytes (default :1) |
| format\_interleaving | int | 0 means interleaved data, 1 means deinterleaved data by packets of "frame size" |
| format\_nbchan | int | number of channels in the stream (default 1) |
| format\_time\_stamp | int | time-stamp format 0:none, 1:absolute time-stamp, 2:relative time, 3:simple counter |
| format\_time\_stamp\_size | int | 0:16bits 1:32bits 2:64bits (see "STREAM\_TIME16D" for example) |
| format\_sdomain | int | subdomain type (for example see stream\_unit\_physical used for analog sensors)  general (a)synchronous sensor + rescaling, .. remote data, compress  audio\_in microphone, line-in, I2S, PDM RX  audio\_out line-out, earphone / speaker, PDM TX, I2S,  gpio\_in generic digital IO , control of relay,  gpio\_out generic digital IO , control of relay,  motion accelerometer, combined or not with pressure and gyroscope  2d\_in camera sensor  2d\_out display, led matrix,  analog\_in with aging control  analog\_out D/A, position piezzo, PWM converter  rtc ticks sent from a programmable timer  user\_interface\_in button, slider, rotary button  user\_interface\_out LED, digits, display,  platform\_3  platform\_2 platform-specific #2, decoded with callbacks  platform\_1 platform-specific #1, decoded with callbacks |
| format\_domain | float | IO\_DOMAIN defined in the platform IO manifest (0 means "any") |
| format\_sampling\_rate | int |  |
| format\_audio\_mapping | int | tbd |
| format\_motion\_mapping | int | tbd |
| format\_2d\_height | int | tbd |
| format\_2d\_width | int | tbd |
| format\_2d\_border | int | tbd |

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
| stream\_io | int | index used to start the declaration of a new IO |
| stream\_io\_format | int | index to the stream format (Index of the above table) (default #0) |
| stream\_io\_hwid | int | ID of the interface given in "files\_manifests\_computer" (default #0) |
| stream\_io\_setting1 | int | setting word32 (SETTINGS\_IOFMT2), the format depends on the IO domain (default #0) |
| stream\_io | int | index used to start the declaration of a new IO |

## Graph memory mapping split

Split the memory mapping to ease memory overlays between nodes and arcs

format : original memory bank ID

new ID to use in the node/arc declarations

start within the original ID

length of the new memory bank

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
| memory\_mapping | Int int int int | ORIGINAL\_ID NEW\_ID START LENGTH |

*Graph debug trace*

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
|  |  |  |

*Subgraphs*

subgraph name, used for name mangling of the nodes and arcs

path ID (set\_file\_path) and file name

list of indexes from "top\_graph\_interface" (or indexes if we are already in a subgraph)

memory\_mapping partitions, list of VIDs used in the subgraph

subgraph

sub1 ; subgraph name, used for name mangling

3 sub\_graph\_0.txt ; path and file name

5 i16: 0 1 2 3 4 ; 5 streaming interfaces data\_in\_0, data\_out\_0 ..

3 i16: 0 0 0 ; 3 partitions here assigned to VID0 : fast-working slow-working slow-static

*Nodes of the graph*

node\_parameters ; node parameters example (default : no parameter)

; Set\_parameter : the array of parameters starts on 32bits-aligned addresses

; The programmer must arrange the data are aligned with respect to the way parameters are read in

; the nanoApp (using pointers to 8/16/32bits fields).

1 i8; 0 ; TAG= 0 "load all parameters"

7 i8; 2 3 4 5 6 7 8 ; parameters

include 1 filter\_parameters.txt ; path + text file-name using parameter syntax

\_end\_

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
| node\_preset | 1 | parameter preset used at boot time, default = #0 |
| node\_malloc\_E | 12 | "E" parameter used in "Memory Size Bytes", default = #0 |
| node\_map\_hwblock | 2 3 | index of the memory block "node\_mem" and the VID indexes from "procmap\_manifest\_xxxx.txt" where to map it. Default VID's is 0. |
| node\_map\_copy | 2 3 | copy the indexed "node\_mem" to VID 3 (faster memory) before run |
| node\_map\_swap | 2 3 | swap the indexed "node\_mem" to VID 3 (faster memory) before run, and restored after |
| node\_trace\_id | 0 | IO port used to send the trace |
| node\_map\_proc | 0 | execute this nanoApp on this processor (0: any possible, default) |
| node\_map\_arch | 0 | execute this nanoApp on this architecture (0: any possible, default) |
| node\_map\_rtos | 0 | execute this nanoApp on this thread index (0: any possible, default) |
| node\_map\_verbose | 0 | level of debug trace, default = #0 |
| node\_script | <0..127> | index of the script to call before and after execution of this node |

Scripting **nos** of the graph

Example

;==============================

; SCRIPTING NODE IN GRAPHS (SCHEDULED LIKE OTHER NODES)

;

; Checks if the data it needs is available and returns to the scheduler

; Its single arc (TX) is always empty

node arm\_stream\_script 1 ; instance index of arm\_stream\_script

script\_registers 2 ; numer of registers used in this script , default 2

script\_pointers 2 ; numer of pointers used in this script , default 2

script\_stack 12 ; size of the stack in word64 (default = 0)

script\_mem\_shared 1 ; Is it a private RAM(0) or can it be shared with other scripts(1)

script\_mem\_map 0 ; Memory mapping to VID #0 (default)

; this declaration creates the transmit arc of the script-node pointing to the stack/buffer area

script\_code

2 h16; 2002 0001 ; movi int16 r0 1

1 h16; e810 ; equ r1,r0

1 h16; 0381 ; ccallsys 1

1 h16; C000 ; ret

\_end\_

script\_assembler ; start of assembler language (@@@ TBD)

pshc int8 1

gtr

cjmp #1

pshc int16

cals readparam

labl #1

ret

\_end\_

node\_parameters <ID2> ; node parameters and index to let the code addressing it

; Set\_parameter : the array of parameters starts on 32bits-aligned addresses

; The programmer must arrange the data are aligned with respect to the way parameters are read in

; the nanoApp (using pointers to 8/16/32bits fields).

1 i8; 0 ; TAG= 0 "load all parameters"

7 i8; 2 3 4 5 6 7 8 ; parameters

node\_parameters <ID2> ;

include 1 binary\_code.txt ; path ID and file name

\_end\_

\_end\_

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
| script\_registers | 2 | numer of registers used in this script , default 2 |
| script\_pointers | 2 | numer of pointers used in this script , default 2 |
| script\_stack | 12 | size of the stack in word64 (default = 0) |
| script\_mem\_shared | 1 | Is it a private RAM(0) or can it be shared with other scripts(1) |
| script\_mem\_map | 0 | Memory mapping to VID #0 (default) this declaration creates the transmit arc of the script-node pointing to the stack/buffer area |

*Shared scripts used for node control*

Example

; COMMON SCRIPTS IN GRAPHS (AND SCRIPT ID IN THE NODE HEADER "SCRIPT\_LW0")

;

; script instance #0 is the "main script" of the subgraph

script 0 ; index of the script, to be mapped to an index in the graph compiler

script\_registers 2 ; same as arm\_stream\_script

script\_pointers 2 ;

script\_stack 12 ;

script\_mem\_shared 1 ;

script\_mem\_map 0 ;

script\_code ; start of byte-codes of the script

2 h16; 2002 0001 ;

1 f64; 3.14159265359 ;

1 h16; e810 ;

1 i8 ; 0 ; parameters embedded in the code and addressed with Labels

7 i8 ; 2 3 4 5 6 7 8 ;

\_end\_

*Graph arcs*

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
|  |  | ---ARC CONNECTED TO GRAPH INTERFACE  in a subgraph the IDX interfaces are sequential 1,2,3.. and documented like function parameters  in the main graph the "top\_graph\_interface" have the indexes to use use in the first column  IO-ID NAME INST IO FORMAT | |
| arc\_input | 1  node\_name 2 0 0 | input arc index #1 connected to "node\_name" instance #2 and its arc index #0, Format #0 | |
| arc\_output | 2  node\_name 3 1 0 | output arc index #2 connected to "node\_name" instance #3 and its arc index #1, Format #0 | |
| arc | node1 1 2 0  node2 3 4 1 | ---ARC CONNECTION BETWEEN TWO NANOAPPS  NAME INST IO FMT NAME INST IO FMT  arc between node1 instance #1 arc index #2, producer format #0 to node2 instance #3 and its arc index #4, consumer format #1 | |
| arc\_flow\_error | 1 | #1 do something depending on domain when a flow error occurs, default #0 (no interpolation) | |
| arc\_debug\_cmd | 1 | debug action "ARC\_INCREMENT\_REG", default = #0 (no debug) | |
| arc\_debug\_reg | 3 | index of the 64bits result, default = #0 | |
| arc\_debug\_page | 0 | debug registers base address + 64bits x 16 registers = 32 word32 / page, default = #0 | |
| arc\_flush | 0 | control of register "MPFLUSH\_ARCW1" : forced flush of data in MProcessing and shared tasks | |
| arc\_extend\_addr | 1 | address range extension-mode of the arc descriptor "EXTEND\_ARCW2" for large NN models, default = #0 (no extension) | |
| arc\_map\_hwblock | 0 | mapping VID index from "procmap\_manifest\_xxxx.txt" to map the buffer, default = #0 (VID0) | |
| arc\_jitter\_ctrl | 1.5 | factor to apply to the minimum size between the producer and the consumer, default = 1.0 (no jitter) | |

nanoApp manifests

Identification

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters (type)** | **Comments** |
|  | ARM | developer name |
|  | arm\_stream\_detector | node name |

Graph parameters

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| node\_nb\_arcs | 1 1 | nb arc input, output, default values "1 1" |
| node\_arc\_parameter | 0 | SWC with extra-large amount of parameters (NN models) will declare it with extra arcs |
| node\_steady\_stream | 1 | (0) the data flow is variable (or constant, default value :1) on all input and output arcs |
| node\_same\_data\_rate | 1 | (0) the arcs have different data rates, (1) all arcs have the same data rate |
| node\_use\_dtcm | 1 | default 0 (no MP DTCM\_LW2), 1: fast memory pointer placed after the arc format |
| node\_use\_arc\_format | 0 | default 1 : the scheduler must push each arc format (LOADFMT\_LW0\_LSB) |
| node\_mask\_library | 15 | default 0 bit-field of dependencies to computing libraries |
| node\_subtype\_units | VRMS | triggers the need for rescaling and data conversion |
| node\_architecture | 0 | arch compatible with (default: 0 = source code) to merge and sort for ARCHID\_LW0 |
| node\_fpu\_used | 0 | fpu option used (default 0: none, no FPU assembly or intrinsic) |
| node\_use\_unlock\_key | 1 | a key-exchange protocol is initiated at reset time |
| node\_node\_version | 101 | version of the computing node |
| node\_stream\_version | 001 | version of the stream scheduler it is compatible with |

Node memory allocation

memory allocation size in bytes =

A : memory allocation in Bytes (default 0)

+ B x nb\_channels of arc(i) : addition memory as a number of channels in arc index i (default 0)

+ C x sampling\_rate of arc(j) ; .. as proportional to the sampling rate of arc index j (default 0)

+ D x frame\_size of arc(k) ; .. as proportional to the frame size used for the arc index k (default 0)

+ E x parameter from the graph ; optional field "malloc\_E" during the node declaration in the graph, for

; example the number of pixels in raw for a scratch area (default 0)

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| node\_mem | 2 | start the declaration of a new memory block with index 2 |
| node\_mem\_alloc | 32 | size = 32Bytes data memory, Static, Fast memory block |
| node\_mem\_nbchan | 4 0 | add in Bytes : 4 x nb of channels of arc 0 |
| node\_mem\_sampling\_rate | 0.1 1 | add in Bytes : 0.1 x sampling rate of arc 1 |
| node\_mem\_frame\_size | 1 0 | add in Bytes : 1 x frame size of arc 0 |
| node\_mem\_alignement | 4 | 4 bytes (default) |
| node\_mem\_retention | 1 | 0 for a Static memory allocation, preserved along the execution (default)  1 for Working (or Scratch) area which can be reused and overlaid by other nodes  2 for memory to be preserved (Retention) after a platform reboot |
| node\_mem\_speed | 2 | 0 for 'best effort' or 'no constraint' on speed access  1 for 'fast' memory selection when possible  2 for 'critical fast' section, to be in I/DTCM when available |
| node\_mem\_relocatable | 1 | Default 0 : not relocatable, 1: a command 'STREAM\_UPDATE\_RELOCATABLE' is  sent to the node to update the pointer to this memory allocation |
| node\_mem\_data0prog1 | 0 | selection data / program |

Node arcs configuration

memory allocation size in bytes =

A : memory allocation in Bytes (default 0)

+ B x nb\_channels of arc(i) : addition memory as a number of channels in arc index i (default 0)

+ C x sampling\_rate of arc(j) ; .. as proportional to the sampling rate of arc index j (default 0)

+ D x frame\_size of arc(k) ; .. as proportional to the frame size used for the arc index k (default 0)

+ E x parameter from the graph ; optional field "malloc\_E" during the node declaration in the graph, for

; example the number of pixels in raw for a scratch area (default 0)

all the nodes must have at least one TX-arc (even a dummy one) used to manage the lock field.

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| node\_arc | 2 | start the declaration of a new arc with index 2 |
| node\_arc\_rx0tx1 | 0 | followed by 0:input 1:output, default = 0 0 and 1 1 |
| node\_arc\_sampling\_rate | {1 16000 44100} | sampling rate options (enumeration in Hz), default "any" |
| node\_arc\_interleaving | 0 | multichannel intleaved (0, default), deinterleaved by frame-size (1) |
| node\_arc\_nb\_channels | {1 1 2} | options for the number of channels (default 1) |
| node\_arc\_raw\_format | {1 17} | options for the raw arithmetics computation format here STREAM\_S16, , default values "1 S16" |
| node\_arc\_frame\_length | {1 1 2 16} | options of possible frame\_size in number of sample (can mono or multi-channel) |
| node\_arc\_frame\_duration | {1 10 22.5} | options of possible frame\_size in [milliseconds] |
|  |  | (one sample can mono or multi-channel), default is "any length" |
|  |  |  |
| node\_arc\_sampling\_period\_s | {1 0.1 0.2 0.4} | sampling period options (enumeration in [second]) |
| node\_arc\_sampling\_period\_day | {1 0.25 1 7} | sampling period options (enumeration in [day]) |
| node\_arc\_sampling\_accuracy | 0.8 | sampling rate accuracy in percent |
|  |  |  |
| node\_arc\_inPlaceProcessing | 1 0 | index of the output arc sharing the same interface buffer as one |
| node\_arc |  | input arc buffer (default: all output buffers are separated from the input buffers) |
|  |  | start the declaration of a new arc with index 2 |

## Platform manifest

The “**VID**” (Virtual Identifier) index is used to translate the graph memory map addresses to physical addresses. This is a memory plane used to have compact representation of 64bits addresses and to help multiprocessors pointing to the same physical addresses even if they have address translators.

The platform **digital manifest** gives the base address and sizes of the memory planes addressed with up to **8 IDs**, each memory plane has multiple VID corresponding to physical memory blocks. By convention the VID index 0 is used for the shared RAM holding the graph’s arc FIFO descriptors (read and write index indexes to buffers).

A system integrator can avoid specifying the VID memory mapping and let the graph compiler manage. Tuning the performance means taking care of overlays, or arranging processors don’t have simultaneous access to the same physical memory banks, and this is where VID indexes are used.

### Paths

3

../../stream\_platform/

../../stream\_platform/windows/manifest/

../../stream\_nodes/

### Digital manifests

1 procmap\_manifest\_computer.txt path index + file name

### List IO manifest files

10

;Path Manifest IO\_AL\_idx ProcCtrl clock-domain

1 io\_platform\_data\_in\_0.txt 0 1 0

1 io\_platform\_data\_in\_1.txt 1 1 0

1 io\_platform\_analog\_sensor\_0.txt 2 1 0

1 io\_platform\_motion\_in\_0.txt 3 1 0

1 io\_platform\_audio\_in\_0.txt 4 1 0

1 io\_platform\_2d\_in\_0.txt 5 1 0

1 io\_platform\_line\_out\_0.txt 6 1 0

1 io\_platform\_gpio\_out\_0.txt 7 1 0

1 io\_platform\_gpio\_out\_1.txt 8 1 0

1 io\_platform\_data\_out\_0.txt 9 1 0

### List of nodes manifests

20

2 swc\_manifest\_none.txt

2 Basic/arm/script/swc\_manifest\_graph\_control.txt

2 Basic/arm/script/swc\_manifest\_script.txt

2 Basic/arm/router/swc\_manifest\_router.txt

2 Basic/arm/converter/swc\_manifest\_converter.txt

2 Basic/arm/amplifier/swc\_manifest\_amplifier.txt

2 Basic/arm/mixer/swc\_manifest\_mixer.txt

2 Audio/arm/filter/swc\_manifest\_filter.txt

2 Audio/arm/detector/swc\_manifest\_detector.txt

2 Basic/arm/rescaler/swc\_manifest\_rescaler.txt

2 Audio/arm/compressor/swc\_manifest\_compressor.txt

2 Audio/arm/decompressor/swc\_manifest\_decompressor.txt

2 Basic/arm/modulator/swc\_manifest\_modulator.txt

2 Basic/arm/demodulator/swc\_manifest\_demodulator.txt

2 Basic/arm/resampler/swc\_manifest\_resampler.txt

2 Basic/arm/qos/swc\_manifest\_qos.txt

2 Basic/arm/split/swc\_manifest\_split.txt

2 image/arm/detector2D/swc\_manifest\_detector2D.txt

2 image/arm/filter2D/swc\_manifest\_filter2D.txt

2 Basic/arm/analysis/swc\_manifest\_analysis.txt

## Digital manifest

; ------------------------------------------------------------------------------------------------------------

; Processor and memory configuration + default memory mapping

; ------------------------------------------------------------------------------------------------------------

;

1 1 9 number of architectures, number of processors, number of memory banks

; memory banks:

; - ID base offset ID reference above

; - VID virtual ID used in the graph for manual mapping, must stay below 99 for swap controls (see NodeTemplate.txt)

; - S 0=any/1=normal/2=fast/3=critical-Fast,

; - W static0/working1/retention2,

; - P shared0/private1,

; - H DMAmemHW1

; - D Data0/Prog1/Both2

; - Size minimum sizes guaranteed per VID starting from @[ID]+offset below

; - Offset maximum offset from the base offset ID, (continuous banks means = previous size + previous offset)

; the memory is further split in the graph "top\_memory\_mapping" to ease mapping and overlays

; ID VID S W P H D Size offset from offsetID

0 0 1 0 0 0 0 95526 10 VID0=DEFAULT flat memory bank, can overlap with the others

0 1 0 0 0 0 0 65526 10 SRAM0 static, hand tuned memory banks

0 2 0 0 0 0 0 30000 65536 SRAM1 static

0 3 0 1 0 0 0 15000 95536 SRAM1 working at application level

0 4 0 1 0 0 0 256000 262144 DDR working at application level

2 5 3 1 1 0 0 1024 262144 DTCM Private memory of processor 1

1 10 0 2 0 0 0 1024 524288 Retention memory

3 20 0 0 0 0 0 200000 10 Data in Flash

2 8 3 1 1 0 1 16384 0 ITCM Private memory of processor 1

; memory offsets ID used by all processors and physical address seen from the main processor

; 0 h20000000 image of "platform\_specific\_long\_offset(intPtr\_t long\_offset[])"

; 1 h28000000 in stream\_al/platform\_XXXXX.c

; 2 h2C000000 TCM Private memory of processor 1

; 3 h08000000 Internal Flash

;----------------------------------------------------------------------------------------

; all architectures

; all processors (processor IDs >0)

;------------------------------------------------------------

1 1 15 processor ID, boolean "I am the main processor" allowed to boot the graphs

; Bit-field computation firmware extensions, on top of the basic one, embedded in Stream services

; EXT\_SERVICE\_MATH 1, EXT\_SERVICE\_DSPML 2, EXT\_SERVICE\_AUDIO 3, EXT\_SERVICE\_IMAGE 4

## Graph interfaces manifests

The concept of Domains is used to select specific parameters (analog, audio, 2D, motion).

Parameters are set from a list of “options” : from list or from a range.

Example of IO manifest. The file starts with the name and domain of the IO, followed by “options” corresponding to the physical domain.

; ------------------------------------------------------------------------------------------------------------

; Manifest of interface abstraction to ADC converter and analog sensor

; ------------------------------------------------------------------------------------------------------------

io\_platform\_sensor\_in\_0 ; name for the tools

analog\_in ; domain name, unit: dB, Vrms, mV/Gauss, dps, kWh, ...

io\_commander0\_servant1 1 ; commander=0 servant=1 (default is servant)

io\_direction\_rx0tx1 1 ; direction of the stream 0:input 1:output from graph point of view

io\_subtype\_units 104 ; depending on the domain. Here Units\_Vrms of the "general" domain (0 = any or underfined)

io\_analogscale 0.55 ; 0.55V is corresponding to full-scale (0x7FFF or 1.0f) with the default setting

io\_sampling\_rate {1 16000 44100 48000} ; sampling rate options (enumeration in Hz)

io\_rescale\_factor 12.24 -44.3 ; [1/a off] analog\_input = invinterpa x ((samples/Full\_Scale\_Digital) - interpoff)

\_end\_

### Options syntax

An option can have several fields each in the list or the range format. The separation of fields is made with “{“ and “}”, the first integer selects the format of the field.

options sets : { index list } { index list }

when the list has one single element "X", this is the value to consider : {X} <=> {1 X} <=> X

when index == 0 it means "any", the list can be empty, the default value is not changed from reset

when index > 0 the list gives the allowed values the scheduler can select

The Index tells the default "value" to take at reset time and to put in the graph

the combination of index give the second word of stream\_format\_io[]

when index < 0 a list of triplets follows to describe a combination of data intervals : A1 B1 C1 A2 B2 C2 ...

A is starting value, B is the increment step, C is the included maximum value

The absolute index value selects the default value in this range

### IO manifest Header

io\_platform\_sensor\_in\_0 ; IO name for the tools

analog\_in ; domain name, among the list below :

domain name description and examples

----------- ------------------------

general (a)synchronous sensor , electrical, chemical, color, .. remote data

audio\_in microphone, line-in, I2S, PDM RX

audio\_out line-out, earphone / speaker, PDM TX, I2S,

gpio\_in generic digital IO , control of relay,

gpio\_out generic digital IO , control of relay,

motion accelerometer, combined or not with pressure and gyroscope

2d\_in camera sensor

2d\_out display, led matrix,

analog\_in with aging control

analog\_out D/A, position piezzo, PWM converter

rtc ticks sent from a programmable timer

user\_interface\_in button, slider, rotary button

user\_interface\_out LED, digits, display,

platform\_x platform-specific #x, decoded with callbacks

### IO configuration

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| io\_commander0\_servant1 | 1 | commander=0 servant=1 (default is servant)  IO stream are managed from the graph scheduler with the help of one subroutine per IO using  the template : typedef void (\*p\_io\_function\_ctrl) (uint32\_t command, uint8\_t \*data, uint32\_t length);  The "command" parameter can be : STREAM\_SET\_PARAMETER, STREAM\_DATA\_START, STREAM\_STOP, STREAM\_SET\_BUFFER.  And one subroutine for all IOs in charge of acknowledge the end of the data move,  to update the circular buffer, manage overflows. This subroutine can be called from ISR  void arm\_graph\_interpreter\_io\_ack (uint8\_t fw\_io\_idx, uint8\_t \*data, uint32\_t data\_size);  Where fw\_io\_idx is the index given in "top\_manifest\_xxxx.txt"  When the IO is "Commander" it calls arm\_graph\_interpreter\_io\_ack() when data is read  When the IO is "Servant" the scheduler call p\_io\_function\_ctrl(STREAM\_DATA\_START, ..) to ask for  data move. Once the move is done the IO driver calls arm\_graph\_interpreter\_io\_ack() |
| io\_buffer\_allocation | 2.1 | default is 0, which means the buffer is declared outside of the graph  The floating-point number is a multiplication factor of the frame size (here 2 frames),  the buffer size is computed with rounding (n = floor(X+0.5))  When more than one byte are exchanged, the IO driver needs a temporary buffer. This buffer  can be allocated "outside(0)" by the IO driver, or ">1" during the graph memory mapping preparation  The memora mapping of this allocation is decided in the graph and can be in general-purpose or  any RAM "0" or specific memory bank for speed reason or reserved for DMA processing, etc .. |
| io\_direction\_rx0tx1 | 1 | direction of the stream 0:input 1:output from graph point of view |
| io\_raw\_format | S16 | options for the raw arithmetics computation format here STREAM\_S16 |
| io\_interleaving | 1 | multichannel intleaved (0), deinterleaved by frame-size (1) |
| io\_nb\_channels | 1 | options for the number of channels |
| io\_frame\_length | {1 1 2 16 } | options of possible frame\_size in number of sample (can mono or multi-channel). |
| io\_frame\_duration | {1 10 22.5} | options of possible frame\_size in [milliseconds]. The default frame length is 1 sample |
| io\_subtype\_units | VRMS | depending on the domain. Here Units\_Vrms of the "general" domain (0 = any or underfined) |
| io\_subtype\_multiple | {DPS GAUSS} | example for multi domain sensor : motion can have up to 4 data units for accelerometer, gyroscope, magnetometer, temperature |
| io\_power\_mode | 0 | to set the device at boot time in stop / off (0)  running mode(1) : digital conversion (BIAS always powered for analog peripherals )  running mode(2) : digital conversion BIAS shut-down between conversions  Sleep (3) Bias still powered but not digital conversions |
| io\_position meter | 1.1 -2.2 0.01 | unit and relative XYZ position with the platform reference point |
| io\_euler\_angles | 10 20 90 | Euler angles with respect to the platform reference orientation, in degrees |
| io\_sampling\_rate | {1 16000 44100 48000} | sampling rate options (enumeration in Hz) |
| io\_sampling\_period\_s | {1 1 60 } | sampling period options (enumeration in [second]) |
| io\_sampling\_period\_day | {1 0.25 1 7} | sampling period options (enumeration in [day]) |
| io\_sampling\_rate\_accuracy | 0.1 | in percentage |
| io\_time\_stamp\_format | {1 1} | 0 no time-stamp, 1 absolute time, 2 relative time from last frame, 3 frame counter |
| io\_time\_stamp\_length | {1 1} | 0/1/2/3 corresponding to 16/32/64/64 bits time formats (default : STREAM\_TIME32) |

### Domain-specific IO configuration

#### general

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| io\_commander0\_servant1 | 1 | commander=0 servant=1 (default is servant)  IO stream are managed from the graph scheduler with the help of one subroutine per IO using  the template : typedef void (\*p\_io\_function\_ctrl) (uint32\_t command, uint8\_t \*data, uint32\_t length);  The "command" parameter can be : STREAM\_SET\_PARAMETER, STREAM\_DATA\_START, STREAM\_STOP, STREAM\_SET\_BUFFER.  And one subroutine for all IOs in charge of acknowledge the end of the data move,  to update the circular buffer, manage overflows. This subroutine can be called from ISR  void arm\_graph\_interpreter\_io\_ack (uint8\_t fw\_io\_idx, uint8\_t \*data, uint32\_t data\_size);  Where fw\_io\_idx is the index given in "top\_manifest\_xxxx.txt"  When the IO is "Commander" it calls arm\_graph\_interpreter\_io\_ack() when data is read  When the IO is "Servant" the scheduler call p\_io\_function\_ctrl(STREAM\_DATA\_START, ..) to ask for  data move. Once the move is done the IO driver calls arm\_graph\_interpreter\_io\_ack() |
| io\_buffer\_allocation | 2.1 | default is 0, which means the buffer is declared outside of the graph  The floating-point number is a multiplication factor of the frame size (here 2 frames),  the buffer size is computed with rounding (n = floor(X+0.5))  When more than one byte are exchanged, the IO driver needs a temporary buffer. This buffer  can be allocated "outside(0)" by the IO driver, or ">1" during the graph memory mapping preparation  The memora mapping of this allocation is decided in the graph and can be in general-purpose or  any RAM "0" or specific memory bank for speed reason or reserved for DMA processing, etc .. |
| io\_direction\_rx0tx1 | 1 | direction of the stream 0:input 1:output from graph point of view |
| io\_raw\_format | S16 | options for the raw arithmetics computation format here STREAM\_S16 |
| io\_interleaving | 1 | multichannel intleaved (0), deinterleaved by frame-size (1) |
| io\_nb\_channels | 1 | options for the number of channels |

#### audio\_in

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| io\_nb\_channels | {1 1 2} | options for the number of channels |
| io\_channel\_mapping | 1 | mono (Front Left), 18 channels can be controlled :  Front Left FL bit0  Front Right FR 1  Front Center FC 2  Low Frequency LFE 3  Back Left BL 4  Back Right BR 5  Front Left of Center FLC 6  Front Right of Center FRC 7  Back Center BC 8  Side Left SL 9  Side Right SR 10  Top Center TC 11  Front Left Height TFL 12  Front Center Height TFC 13  Front Right Height TFR 14  Rear Left Height TBL 15  Rear Center Height TBC 16  Rear Right Height TBR 17 |
| io\_analog\_gain | {1 0 12 24 } | analog gain (PGA) |
| io\_digital\_gain | {-1 -12 1 12 } | digital gain range |
| io\_hp\_filter | {1 1 20 50 300 } | high-pass filter (DC blocker) ON(1)/OFF(0) followed by cut-off frequency options |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### audio\_out

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameter** | **Comments** |
| **io\_subtype\_units** | 87 | Units is [mV] |
| **io\_analog\_scale** | 1400 | 1400nV is corresponding to full-scale with the default setting |
| **io\_sampling\_rate** | {1 16000 44100 48000} | sampling rate options (enumeration in Hz) |
| **io\_nb\_channels** | {1 1 1 2 | ; multichannel intleaved (0), deinterleaved by frame-size (1) + options for the number of channels |
| **io\_channel\_mapping** | 1 | mono (Front Left), 18 channels can be controlled : |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Data types

Raw data types

STREAM\_DATA\_ARRAY 0 see stream\_array: [0NNNTT00] 0, type, nb

STREAM\_S1 1 S, one signed bit, "0" = +1 one bit per data

STREAM\_U1 2 one bit unsigned, boolean

STREAM\_S2 3 SX two bits per data

STREAM\_U2 4 XX

STREAM\_Q1 5 Sx ~stream\_s2 with saturation management

STREAM\_S4 6 Sxxx four bits per data

STREAM\_U4 7 xxxx

STREAM\_Q3 8 Sxxx

STREAM\_FP4\_E2M1 9 Seem micro-float [8 .. 64]

STREAM\_FP4\_E3M0 10 Seee [8 .. 512]

STREAM\_S8 11 Sxxxxxxx eight bits per data

STREAM\_U8 12 xxxxxxxx ASCII char, numbers..

STREAM\_Q7 13 Sxxxxxxx arithmetic saturation

STREAM\_CHAR 14 xxxxxxxx

STREAM\_FP8\_E4M3 15 Seeeemmm NV tiny-float [0.02 .. 448]

STREAM\_FP8\_E5M2 16 Seeeeemm IEEE-754 [0.0001 .. 57344]

STREAM\_S16 17 Sxxxxxxx.xxxxxxxx 2 bytes per data

STREAM\_U16 18 xxxxxxxx.xxxxxxxx Numbers, UTF-16 characters

STREAM\_Q15 19 Sxxxxxxx.xxxxxxxx arithmetic saturation

STREAM\_FP16 20 Seeeeemm.mmmmmmmm half-precision float

STREAM\_BF16 21 Seeeeeee.mmmmmmmm bfloat

STREAM\_Q23 22 Sxxxxxxx.xxxxxxxx.xxxxxxxx 24bits 3 bytes per data

STREAM\_Q23\_32 23 SSSSSSSS.Sxxxxxxx.xxxxxxxx.xxxxxxxx 4 bytes per data

STREAM\_S32 24 one long word

STREAM\_U32 25 xxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx UTF-32, ..

STREAM\_Q31 26 Sxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx

STREAM\_FP32 27 Seeeeeee.mmmmmmmm.mmmmmmmm.mmmmmmmm FP32

STREAM\_CQ15 28 Sxxxxxxx.xxxxxxxx Sxxxxxxx.xxxxxxxx (I Q)

STREAM\_CFP16 29 Seeeeemm.mmmmmmmm Seeeeemm.mmmmmmmm (I Q)

STREAM\_S64 30 long long 8 bytes per data

STREAM\_U64 31 unsigned 64 bits

STREAM\_Q63 32 Sxxxxxxx.xxxxxx ....... xxxxx.xxxxxxxx

STREAM\_CQ31 33 Sxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx Sxxxx..

STREAM\_FP64 34 Seeeeeee.eeemmmmm.mmmmmmm ... double

STREAM\_CFP32 35 Seeeeeee.mmmmmmmm.mmmmmmmm.mmmmmmmm Seee.. (I Q)

STREAM\_FP128 36 Seeeeeee.eeeeeeee.mmmmmmm ... quadruple precision 16 bytes per data

STREAM\_CFP64 37 fp64 fp64 (I Q)

STREAM\_FP256 38 Seeeeeee.eeeeeeee.eeeeemm ... octuple precision 32 bytes per data

STREAM\_TIME16 39 ssssssssssssqqqq q14.2 1 hour + 8mn +/- 0.0625

STREAM\_TIME16D 40 qqqqqqqqqqqqqqqq q15 [s] time difference +/- 15us

STREAM\_TIME32 41 ssssssssssssssssssssssssssssqqqq q28.4 [s] (8.5 years +/- 0.0625s)

STREAM\_TIME32D 42 ssssssssssssssssqqqqqqqqqqqqqqqq q17.15 [s] (36h, +/- 30us) time difference

STREAM\_TIMESTMP 43 ssssssssssssssssssssqqqqqqqqqqqq q20.12 [s] (12 days, +/- 0.25ms)

STREAM\_TIME64 44 \_\_\_\_ssssssssssssssssssssssssssssssssqqqqqqqqqqqqqqqqqqqqqqqqqqqq q32.28 [s] 140 Y +Q28 [s]

STREAM\_TIME64MS 45 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm ms

STREAM\_TIME64ISO 46 \_\_\_..YY..YY..YY..YY..MM..MM..DD..DD..SS..SS.....offs..MM..MM..MM ISO8601 signed offset 2024-05-04T21:12:02+07:00

STREAM\_WGS84 47 <--LATITUDE 32B--><--LONGITUDE 32B--> lat="52.518611" 0x4252130f lon="13.376111" 0x4156048d - dual IEEE754

STREAM\_HEXBINARY 48 UTF-8 lower case hexadecimal byte stream

STREAM\_BASE64 49 RFC-2045 base64 for xsd:base64Binary XML data

STREAM\_STRING8 50 UTF-8 string of char terminated by 0

STREAM\_STRING16 51 UTF-16 string of char terminated by 0

Physical units (RFC8428 RFC8798)

STREAM\_SUBT\_ANA\_ANY 0 any

STREAM\_SUBT\_ANA\_METER 1 m meter

STREAM\_SUBT\_ANA\_KGRAM 2 kg kilogram

STREAM\_SUBT\_ANA\_GRAM 3 g gram\*

STREAM\_SUBT\_ANA\_SECOND 4 s second

STREAM\_SUBT\_ANA\_AMPERE 5 A ampere

STREAM\_SUBT\_ANA\_KELVIB 6 K kelvin

STREAM\_SUBT\_ANA\_CANDELA 7 cd candela

STREAM\_SUBT\_ANA\_MOLE 8 mol mole

STREAM\_SUBT\_ANA\_HERTZ 9 Hz hertz

STREAM\_SUBT\_ANA\_RADIAN 10 rad radian

STREAM\_SUBT\_ANA\_STERADIAN 11 sr steradian

STREAM\_SUBT\_ANA\_NEWTON 12 N newton

STREAM\_SUBT\_ANA\_PASCAL 13 Pa pascal

STREAM\_SUBT\_ANA\_JOULE 14 J joule

STREAM\_SUBT\_ANA\_WATT 15 W watt

STREAM\_SUBT\_ANA\_COULOMB 16 C coulomb

STREAM\_SUBT\_ANA\_VOLT 17 V volt

STREAM\_SUBT\_ANA\_FARAD 18 F farad

STREAM\_SUBT\_ANA\_OHM 19 Ohm ohm

STREAM\_SUBT\_ANA\_SIEMENS 20 S siemens

STREAM\_SUBT\_ANA\_WEBER 21 Wb weber

STREAM\_SUBT\_ANA\_TESLA 22 T tesla

STREAM\_SUBT\_ANA\_HENRY 23 H henry

STREAM\_SUBT\_ANA\_CELSIUSDEG 24 Cel degrees Celsius

STREAM\_SUBT\_ANA\_LUMEN 25 lm lumen

STREAM\_SUBT\_ANA\_LUX 26 lx lux

STREAM\_SUBT\_ANA\_BQ 27 Bq becquerel

STREAM\_SUBT\_ANA\_GRAY 28 Gy gray

STREAM\_SUBT\_ANA\_SIVERT 29 Sv sievert

STREAM\_SUBT\_ANA\_KATAL 30 kat katal

STREAM\_SUBT\_ANA\_METERSQUARE 31 m2 square meter (area)

STREAM\_SUBT\_ANA\_CUBICMETER 32 m3 cubic meter (volume)

STREAM\_SUBT\_ANA\_LITER 33 l liter (volume)

STREAM\_SUBT\_ANA\_M\_PER\_S 34 m/s meter per second (velocity)

STREAM\_SUBT\_ANA\_M\_PER\_S2 35 m/s2 meter per square second (acceleration)

STREAM\_SUBT\_ANA\_M3\_PER\_S 36 m3/s cubic meter per second (flow rate)

STREAM\_SUBT\_ANA\_L\_PER\_S 37 l/s liter per second (flow rate)\*

STREAM\_SUBT\_ANA\_W\_PER\_M2 38 W/m2 watt per square meter (irradiance)

STREAM\_SUBT\_ANA\_CD\_PER\_M2 39 cd/m2 candela per square meter (luminance)

STREAM\_SUBT\_ANA\_BIT 40 bit bit (information content)

STREAM\_SUBT\_ANA\_BIT\_PER\_S 41 bit/s bit per second (data rate)

STREAM\_SUBT\_ANA\_LATITUDE 42 lat degrees latitude[1]

STREAM\_SUBT\_ANA\_LONGITUDE 43 lon degrees longitude[1]

STREAM\_SUBT\_ANA\_PH 44 pH pH value (acidity; logarithmic quantity)

STREAM\_SUBT\_ANA\_DB 45 dB decibel (logarithmic quantity)

STREAM\_SUBT\_ANA\_DBW 46 dBW decibel relative to 1 W (power level)

STREAM\_SUBT\_ANA\_BSPL 47 Bspl bel (sound pressure level; log quantity)

STREAM\_SUBT\_ANA\_COUNT 48 count 1 (counter value)

STREAM\_SUBT\_ANA\_PER 49 / 1 (ratio e.g., value of a switch; [2])

STREAM\_SUBT\_ANA\_PERCENT 50 % 1 (ratio e.g., value of a switch; [2])\*

STREAM\_SUBT\_ANA\_PERCENTRH 51 %RH Percentage (Relative Humidity)

STREAM\_SUBT\_ANA\_PERCENTEL 52 %EL Percentage (remaining battery energy level)

STREAM\_SUBT\_ANA\_ENERGYLEVEL 53 EL seconds (remaining battery energy level)

STREAM\_SUBT\_ANA\_1\_PER\_S 54 1/s 1 per second (event rate)

STREAM\_SUBT\_ANA\_1\_PER\_MIN 55 1/min 1 per minute (event rate, "rpm")\*

STREAM\_SUBT\_ANA\_BEAT\_PER\_MIN 56 beat/min 1 per minute (heart rate in beats per minute)

STREAM\_SUBT\_ANA\_BEATS 57 beats 1 (Cumulative number of heart beats)\*

STREAM\_SUBT\_ANA\_SIEMPERMETER 58 S/m Siemens per meter (conductivity)

STREAM\_SUBT\_ANA\_BYTE 59 B Byte (information content)

STREAM\_SUBT\_ANA\_VOLTAMPERE 60 VA volt-ampere (Apparent Power)

STREAM\_SUBT\_ANA\_VOLTAMPERESEC 61 VAs volt-ampere second (Apparent Energy)

STREAM\_SUBT\_ANA\_VAREACTIVE 62 var volt-ampere reactive (Reactive Power)

STREAM\_SUBT\_ANA\_VAREACTIVESEC 63 vars volt-ampere-reactive second (Reactive Energy)

STREAM\_SUBT\_ANA\_JOULE\_PER\_M 64 J/m joule per meter (Energy per distance)

STREAM\_SUBT\_ANA\_KG\_PER\_M3 65 kg/m3 kg/m3 (mass density, mass concentration)

STREAM\_SUBT\_ANA\_DEGREE 66 deg degree (angle)\*

STREAM\_SUBT\_ANA\_NTU 67 NTU Nephelometric Turbidity Unit

dary Unit (rfc8798) Description SenML Unit Scale Offset

STREAM\_SUBT\_ANA\_MS 68 millisecond s 1/1000 0 1ms = 1s x [1/1000]

STREAM\_SUBT\_ANA\_MIN 69 minute s 60 0

STREAM\_SUBT\_ANA\_H 70 hour s 3600 0

STREAM\_SUBT\_ANA\_MHZ 71 megahertz Hz 1000000 0

STREAM\_SUBT\_ANA\_KW 72 kilowatt W 1000 0

STREAM\_SUBT\_ANA\_KVA 73 kilovolt-ampere VA 1000 0

STREAM\_SUBT\_ANA\_KVAR 74 kilovar var 1000 0

STREAM\_SUBT\_ANA\_AH 75 ampere-hour C 3600 0

STREAM\_SUBT\_ANA\_WH 76 watt-hour J 3600 0

STREAM\_SUBT\_ANA\_KWH 77 kilowatt-hour J 3600000 0

STREAM\_SUBT\_ANA\_VARH 78 var-hour vars 3600 0

STREAM\_SUBT\_ANA\_KVARH 79 kilovar-hour vars 3600000 0

STREAM\_SUBT\_ANA\_KVAH 80 kilovolt-ampere-hour VAs 3600000 0

STREAM\_SUBT\_ANA\_WH\_PER\_KM 81 watt-hour per kilometer J/m 3.6 0

STREAM\_SUBT\_ANA\_KIB 82 kibibyte B 1024 0

STREAM\_SUBT\_ANA\_GB 83 gigabyte B 1e9 0

STREAM\_SUBT\_ANA\_MBIT\_PER\_S 84 megabit per second bit/s 1000000 0

STREAM\_SUBT\_ANA\_B\_PER\_S 85 byteper second bit/s 8 0

STREAM\_SUBT\_ANA\_MB\_PER\_S 86 megabyte per second bit/s 8000000 0

STREAM\_SUBT\_ANA\_MV 87 millivolt V 1/1000 0

STREAM\_SUBT\_ANA\_MA 88 milliampere A 1/1000 0

STREAM\_SUBT\_ANA\_DBM 89 decibel rel. to 1 milliwatt dBW 1 -30 0 dBm = -30 dBW

STREAM\_SUBT\_ANA\_UG\_PER\_M3 90 microgram per cubic meter kg/m3 1e-9 0

STREAM\_SUBT\_ANA\_MM\_PER\_H 91 millimeter per hour m/s 1/3600000 0

STREAM\_SUBT\_ANA\_M\_PER\_H 92 meterper hour m/s 1/3600 0

STREAM\_SUBT\_ANA\_PPM 93 partsper million / 1e-6 0

STREAM\_SUBT\_ANA\_PER\_100 94 percent / 1/100 0

STREAM\_SUBT\_ANA\_PER\_1000 95 permille / 1/1000 0

STREAM\_SUBT\_ANA\_HPA 96 hectopascal Pa 100 0

STREAM\_SUBT\_ANA\_MM 97 millimeter m 1/1000 0

STREAM\_SUBT\_ANA\_CM 98 centimeter m 1/100 0

STREAM\_SUBT\_ANA\_KM 99 kilometer m 1000 0

STREAM\_SUBT\_ANA\_KM\_PER\_H 100 kilometer per hour m/s 1/3.6 0

STREAM\_SUBT\_ANA\_GRAVITY 101 earth gravity m/s2 9.81 0 1g = m/s2 x 9.81

STREAM\_SUBT\_ANA\_DPS 102 degrees per second 1/s 360 0 1dps = 1/s x 1/360

STREAM\_SUBT\_ANA\_GAUSS 103 Gauss Tesla 10-4 0 1G = Tesla x 1/10000

STREAM\_SUBT\_ANA\_VRMS 104 Volt rms Volt 0.707 0 1Vrms = 1Volt (peak) x 0.707

STREAM\_SUBT\_ANA\_MVPGAUSS 105 Hall effect, mV/Gauss millivolt 1 0 1mV/Gauss

List of installed nanoApps

;----------------------------------------------------------------------------------------

; List of nodes

;----------------------------------------------------------------------------------------

;

1 arm\_stream\_graph\_control, /\* scheduler control : lock, bypass, loop, if-then \*/

2 arm\_stream\_script, /\* byte-code interpreter, index "arm\_stream\_script\_INDEX" \*/

3 arm\_stream\_router, /\* copy input arcs and subchannel and output arcs and subchannels \*/

4 arm\_stream\_converter, /\* raw data format converter \*/

5 arm\_stream\_amplifier, /\* amplifier mute and un-mute with ramp and delay control \*/

6 arm\_stream\_mixer, /\* multichannel mixer with mute/unmute and ramp control \*/

7 arm\_stream\_filter, /\* cascade of DF1 filters \*/

8 arm\_stream\_detector, /\* estimates peaks/floor of the mono input and triggers a flag on high SNR \*/

9 arm\_stream\_rescaler, /\* raw data values remapping using "interp1" \*/

10 arm\_stream\_compressor, /\* raw data compression with adaptive prediction \*/

11 arm\_stream\_decompressor, /\* raw data decompression \*/

12 arm\_stream\_modulator, /\* signal generator with modulation \*/

13 arm\_stream\_demodulator, /\* signal demodulator, frequency estimator \*/

14 arm\_stream\_resampler, /\* asynchronous sample-rate converter \*/

15 arm\_stream\_qos, /\* raw data interpolator with synchronization to one HQoS stream \*/

16 arm\_stream\_split, /\* let a buffer be used by several nodes \*/

17 arm\_stream\_detector2D, /\* activity detection, pattern detection \*/

18 arm\_stream\_filter2D, /\* Filter, rescale, rotate, exposure compensation \*/

19 arm\_stream\_analysis, /\* arm\_stream\_analysis, \*/

List of app’s callbacks

## Node design

Calling sequence

The main CMSIS-Stream instance (the one located in the main process or processor) is called by the application to compute the amount of memory needed to execute the graph : the buffers of the arcs, the SWC instances of the graph, the buffers used for IOs (command “STREAM\_MEMREQ” below).

In a second step, the application provides the memory pointers to the requested memory banks. The CMSIS-Stream instances are now allowed to activate of the IOs at the boundary of the graph, do the memory initialization of all SWC (command “STREAM\_RESET” below).

Finally, the application lets the graph being scheduled by CMSIS-Stream (command “STREAM\_RUN” above).

The first control API has four parameters, three data parameters and a command with values :

STREAM\_MEMREQ : the application asks for the amount of memory needed to schedule the graph; the function returns the needed amount of memory for each memory bank (see “1.1.2 processor characteristics”). The parameters are :

• A function pointer to the firmware of the platform, in charge of the low-level abstraction of the hardware controls:

o Returns the details of current processor: its index in the manifest table, its architecture and the FPU options

o Call one on the three functions used to control the device drivers : “set”, “start”, “stop” (see “2) The graph boundaries”)

o Read the time information, for example computed from a SYSTICK global counter.

• A pointer to the list of SWC entry points, and a pointer to their respective manifests (see “1.3 SWC manifests”)

• A pointer to the “graph description text” to be compiled to “binary graph structure”

a) STREAM\_RESET : pointers memory banks are provided to “arm\_stream()” which can initialize its instances and the SWC instances of the graph. In a similar way described for SWC (see “3.1.2 SWC parameter RESET”) the application is providing to the CMSIS-Stream a callback mechanism. Each CMSIS-Stream instance is stored in the shared memory of “binary graph structure” with the format:

• 7 offsets to the physical memory banks (see “1.1.2 processor characteristics”). This information lets the arc’s buffer being address with indexes instead of physical pointers, it allows sharing the same arc’s descriptors among processors having different memory address decoding (including arch64 processors).

• Debug informations, the execution state of the instance, the current SWC under processing.

• A 32bits-field of the graph IO ports to have look to. Most CMSIS-Stream instances will not be given access to the peripherals. Those indexes are used to address the platform IO manifest and checking the associated graph’s ring buffers are not having flow problems.

• A function pointer to the firmware of the platform (see above).

• The list of SWC entry points

b) STREAM\_RUN : the graph of components is scheduled (the linked-list of the “binary graph structure” is parsed, see “3) Linked list of SWC”)

c) STREAM\_END : command forwarded to each SWC to release memory allocated with stdlib.

CMSIS-Stream is scheduling the software components of a graph. The nodes of the graph are software components (“SWC ”) independent of the platform capabilities.

The graph description is a text file (example here) and is the result from the translation made in a GUI tool, using :

• a manifest of the platform (details on processors, memory, peripherals)

• a manifest of each SWC : description of the data formats of the interfaces

CMSIS-Stream is translating the graph description text file to a binary graph structure, with the help of the data in the manifests. This result is placed in shared memory area to all processes and processors.

This shared binary graph structure consists in :

• the linked list of arcs and nodes (the SWC) of the graph

• the arcs descriptors (read and write indexes to circular buffers)

• the memory of the CMSIS-Stream instances scheduled the graph.

• the structure describing the operations at the boundary of the graph (the graph “IOs”)

• registers used to synchronize the different CMSIS-Stream instances, if any

Two entry-points

CMSIS-Stream has two entry-points, one for controling and asking for services, and a second one used as callback for notifications of data transfers :

void arm\_stream (uint32\_t command, PTR\_INT ptr1, PTR\_INT ptr2, PTR\_INT ptr3);

void arm\_stream\_io (uint32\_t fw\_io\_idx, void \*data, uint32\_t length);

The second control API (arm\_stream\_io) has three parameters : the index of the device driver calling this function, the base address of the buffer, the size of the buffer. The “index” is given in the platform IO manifest (see ”1.2.4 The ID of the hardware port”). Data format and interleaving is described at “A.3.1 Data format fields common to all streams”.

The description of the scheduling of the graph consists in :

• the content of the manifests of the platform and the manifests of the SWC

o paragraph below “1) Platform and SWC manifests”

• the way the IOs are sharing data with the ring buffers at the boundary of the graph

o paragraph below “2) The graph boundaries”

• the description of the linked-list and the connexions between arcs and nodes

o paragraph below “3) Linked list of SWC”

3.1 SWC interface

The SWC have a single entry point in the format “func (int, \*,\*,\*)”. The first parameter is the execution command (memory request, reset, set parameters, run, stop). The SWC can call CMSIS-Stream through a dedicated function pointer provided at reset time. An example of SWC API here.

3.1.1 SWC parameter “MEMREQ”

The first operation asked by the scheduler is to ask the SWC for memory allocation with respect to parameters associated to the input and output stream format (the SWC may ask for working memory buffer size in relation with the frame size of the streams).

A SWC delivered in source-code, or as library object using the same compilation tool chain as the application, can use memory allocation function from the standard C library (malloc(), realloc(), calloc()), and will have to manage the “free()” deallocation upon reception of the command “STREAM\_END”.

The format is : func (STREAM\_MEMREQ, \*ptr1, \*ptr2, \*ptr3):

• The first pointer is a memory space of 7 words of 32bits. The SWC will fill this area with up to 6 memory allocation requests terminated with “0”. Each word is a bit-field (description in “A.4 Memory types”) giving the size of the memory buffer, the byte alignment and the recommended speed. The memory can be declared “static” or “working”(or “scratch memory area”), depending if the content needs to be preserved between two calls. The first memory request is the “instance”, which holds pointers to static and working memory buffers. The pointer to this memory area is reused in all the other SWC commands.

• The second parameter is pointing to a table of the stream formats used (see “A.3 Stream digital “data formats"”). This information (buffer size, sampling rate, interleaving scheme) can be used by the SWC to adjust the request to the minimum amount of memory.

• The last parameter is TBD and reserved for a SWC activation protocol with key exchanges

3.1.2 SWC parameter “RESET”

The second operation of the scheduler is to provide the SWC with the memory being allocated.

The format is : func (STREAM\_RESET, \*ptr1, \*ptr2, \*ptr3):

• The first parameter points to the SWC instance, with memory allocation corresponding to the first word of the STREAM\_MEMREQ. The following data is a vector of pointers corresponding the memory allocation requested in the same following order provided by the STREAM\_MEMREQ.

• The second parameter is a pointer to the entry point of CMSIS-Stream, and giving access to optional services in computing, signal compression. There is a protocol TBD to activate this link : the SWC will use a single subroutine as calling address and will register the return address (seen by CMSIS-Stream) with a dummy call during this initialization sequence.

• The last parameter is unused.

3.1.3 SWC parameter “SET\_PARAMETER”

CMSIS-Stream is setting the SWC parameter at reset time with the default reset parameter vector provided in its manifest. This API allows to change a single parameter or the full set.

The format is : func (STREAM\_SET\_PARAMETERS, \*ptr1, \*ptr2, \*ptr3):

• The first parameter “ptr1” points to the SWC instance.

• The second parameter points to the parameters to be updated

• The last parameter will be casted to integer and the LSB 9 bits tell the index or the tag used by the SWC documentation to change one parameter. The value 256 tells the full parameter list will be set.

The scheduler has no way to decide to change a parameter during the execution of the graph. The Scripts are used for this purpose.

3.1.4 SWC parameter “READ\_PARAMETER”

The scheduler is reading the SWC parameter at reset time with the default reset parameter vector provided in its manifest. This API allows to change a single parameter or the full set.

The format is : func (STREAM\_READ\_PARAMETERS, \*ptr1, \*ptr2, \*ptr3):

• The first parameter “ptr1” points to the SWC instance.

• The second parameter points to the parameters to be updated

• The last parameter will be casted to integer and the LSB 9 bits tell the index or the tag used by the SWC documentation to change one parameter. The value 256 tells the full parameter list will be set.

3.1.5 SWC parameter “RUN”

The scheduler launches the execution of the SWC when the conditions, found in the linked-list fields (processor architecture, arc’s ready-flags) are set. The stream of data from the arcs are exchanged in the format detailed in the SWC manifest (see “A.3.1 Data format fields common to all streams”).

The calling format is : func (STREAM\_RUN, \*ptr1, \*ptr2, \*ptr3).

As previously “ptr1” is the instance pointer of the component. “ptr2” is a pointer to a structure.

The "\*ptr2" field points to a structure : [{data pointer stream1} {data pointer stream2} .. ]

The "\*ptr3" field points to a similar structure : [{parameter of stream1} {parameter of stream2} .. ]

Simple components have two streams : one as input, the other as output other can have up to 4 streams (several input/output combinations).

A data buffer is combination of a pointer and size. For input streams the size is the amount of data in the buffer. For output streams this is the amount of free space in the output buffer starting from the pointer address,

The SWC is updating the base address pointers and data sizes before returning to caller.

When a stream data format is FMT\_INTERLEAVED, (for example Left and Right audio samples are in this order : LRLRLRLRLRLR .. ) then {data stream} is a pointer to the base address, {parameter stream} is the number bytes in the buffer.

When a stream data format is FMT\_DEINTERLEAVED\_1PTR, (for example Left and Right audio samples are in this order : LLLL..LLLLRRRR...RRRR ) the size of the first buffer (the "frame") then {data stream} is a pointer to the base address, {parameter stream} is the number bytes for a single frame (the size of the Left sample portion). The SWC will address the second and following channels by incrementing the base pointer address with the size of the frame.

When a stream data format is FMT\_DEINTERLEAVED\_NPTR. The buffers have independent positions (for example color planes of images). The {data stream} is a combination of the pointer to the base address, and the size of the corresponding buffer. {parameter stream} is unused. For example with stereo audio : {data stream} points to a structure : [ {\*ptr\_L, size L}, {\*ptr\_R, size R} ], this is the format used in EEMBC-audiomark.

3.1.6 SWC parameter “END”

This command is used to free the allocated memory. The format is : func (STREAM\_END, instance pointer, 0, 0).

3.2 ARCs descriptors

Two types of arcs are used in the graph : simple linear buffers and ring buffers. Simple arcs are described with two words of 32bits, Ring descriptors are using 6 words (the first 2 words are identical to the ones of simple arcs) with the purpose of managing complex situation with peripherals, multiprocessing, drift compensations, data monitoring. Ring buffers are used at the boundary of the graph, their content is realigned to the base address to avoid the SWC to manage folding addresses. The data format is:

• Word 0 : 27 bits for the buffer base address and 5bits for the data format

o The base address is computed in 3 sub-fields of a linear 22bits address shifted with an exponent of E = 2bits and 3bits selecting one of the 7 memory banks (see “1.1.2 processor characteristics”). Each CMSIS-Stream instance (see “Two entry points b) STREAM\_RESET”) has its table of offset to physical memory banks. The base address value is physical\_address[offset] + (linear address << (E<<2)).

o The 5bits data format field is a compact way to give the data format details (see “A.3.1 Data format fields common to all streams”) which is 128bits long. This data is an index to the table of data formats used in the graph. This table is part of the shared binary graph structure.

• Word 1 : 24bits buffer size, 2bits to select the arc type, 8 bits for debug

o The buffer length is computed with a 22bits linear address and a 2 bits shifter to extend the length.

o The 2 bits selector give the indication of linear or ring descriptor format

o For debug, 4bits are ging the debug task to proceed, and 4 bits to select the debug register of the result. The debug register array is in the shared binary graph structure. The debug tasks are TBD and could be : estimation of the data rate, the time-stamp of the last acces, the peak / min / absmax data values with different forgetting factors.

• Word 2: read index on 24bits, flow management on 4bits, data alignment decision bit

o The read index is a plain 24bit index in Bytes, and allows to manage 16MB buffers

o There are 2bits for underflow and 2bits for overflow management during read/write access. The decision thresholds (“crumb\_in”, “crumb\_out”) are given in the Word-3. The underflow options are : repeat last frames, generate zeroes (default), extrapolate last frame. The overflow options are : skip last frame, interpolate last frame.

o To avoid data to fold when the write pointer is too close to the top of the buffer a data realignement will be proceeded based on a selection of ‘0’ the crumb-in threshold, ‘1’ the crumb-out threshold.

• Word 3: write index on 24bits, 4 bits of crumb\_in, crumb\_out

o The write index is a plain 24bit index in Bytes. Buffer full condition corresponds to (W-R) = (Size-1).

o Crumb\_in (2bits) is used to set the “ready flag” (see “3) Linked list of SWC”) of the consumer SWC when the amount of data in the buffer is either larger the buffer size /2 /4 /8 /16.

o Crumb\_out (2bits) is used to set the “ready flag” (see “3) Linked list of SWC”) of the producer SWC when the amount of free area in the buffer is either larger the buffer size /2 /4 /8 /16.

• Word 4: offset (19 bits) to the linked-list header of the SWC consumer of this arc.

• Word 5: Debug address field (27bits, format of the base address)

A Data types

A.1 Raw data types

Sample of raw data type here.

A.2 Array of Raw data types

When the raw data type is null, the next byte is the raw data type and the next 2 bytes are the number of raw data as array.

A.3 Stream digital “data formats"

The description of a full data format is made on 16 Bytes (4 words of 32bits): 8 Bytes for the digital format (frame-size, data interleaving, the raw data format in the frame, the number of channels, the sampling rate..) and 8 Bytes for domain-specific data format and mixed-signal sensor/transducer peripheral control.

A.3.1 Data format fields common to all streams

The first word of the common data format has the fields:

• Frame size in Bytes, on 24bits, allowing data frames of 16MB. The value 0 means “any size”. SWC manifests are using the data format to for each of their input and output ports. In this case, the field “frame size” means the minimum of input data needed by the SWC before any processing is feasible. And for the output ports: the minimum of free area to have in the output buffer to avoid overflow due to the data being produced by the SWC.

• Interleaving scheme, on 2 bits, three options being used:

o “FMT\_INTERLEAVED” : data is interleaved within the frame, in sequence. For example left and right audio samples are found like “LRLRLRLR..” or gyroscope data is provided like “XYZXYZXYZ..”

o “FMT\_DEINTERLEAVED\_1PTR” : interleaved format of frame-size (“LLLLLRRRRR LLLLLRRRRR..”)

o “FMT\_DEINTERLEAVED\_NPTR” : the data addressed with pairs of pointers and frame-size, the buffer do not need to be contiguous.

• Raw data format, on 6 bits, see “A.1 Raw data types”.

The second word of the common data format has the fields:

• Number of channels minus one, on 5 bits, allowing up to 32 channels.

• Sampling rate, on 21 bits, the value 0 means “asynchronous or slave IO port”. The data format is F=M x 2\*\*E. With 19 bits for the unsigned mantissa “M”, 2 bits for the exponents (values of “E” = 0,-8,-16,-24), allowing to set the rate from 1 week period up to 524kHz, with all the common audio and voice sampling rate with full accuracy.

• Time-stamp format, on 2bits:

o No timestamp on the data frames

o Absolute value of the time (64bits) on each frame, 2 MSB to control the format, 40 bits counting the SYSTICK periods of 1ms, 16 bits computed from SYST\_RVR to have the fraction of 1ms.

o Relative time from last frame (32 bits), 30bits of data and 2 MSB to control the format (0: seconds spent from reset, 1: seconds spent from January 1st 2023, 2: seconds interval in U14.16 format, 3: seconds interval in U24.4 format).

• Four bits unused

A.3.2 Domain-specific stream data format

The third word of the domain-specific data format has the fields :

• Domain code on 6 bits

• Data mapping of the channels on 24 bits. Example of 7.1 format audio format (8 channels) FrontLeft, FrontRight, FrontCenter, LowFrequency, BackLeft, BackRight, SideLeft, SideRight. The code 0101b of a 2-channel stream, means data for FrontLeft and FrontCenter.

• Direction of the flow, 1 bit, “0” means data flow sent to the graph and “1” means generated by the graph processing

• Hashing of the stream, TBD