# EARLY DRAFT

# Graph-Interpreter
# a scheduler of DSP/ML nanoApps

Contents

## Use-cases

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

1. **Accelerate time to market**

For system integrators and OEM : develop complex DSP/ML stream processing. Go fast from prototypes validated on a computer to final tuning steps on board, 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. Arrange the data flow is translated in the node desired formats. Prepare the conditions where node (nanoApps) will be delivered in ciphered binary format.

3. **Portability, scalability**

Use the same stream-based processing methodology from devices using 1kBytes of internal RAM to multiprocessor heterogeneous architectures.

**Use-case examples :**

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

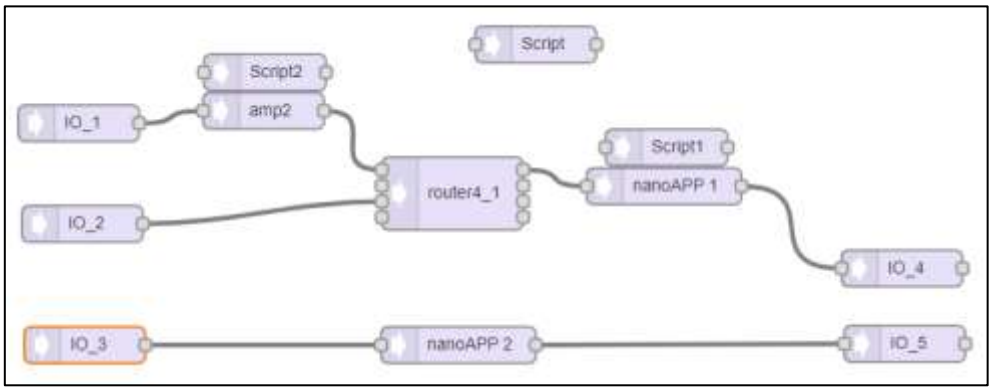
A block of Flash is reserved for the graph in a device. The graph implements a cascade of ML algorithms from analog sensing. The results are sent to the main application. The system integration consists in tuning the levels before data is shared with the application. A node is in charge of rescaling and resampling the input data rate for the next computing nodes in the graph. The system integrator updates the rescaler node and a debug script to trigger a GPIO based on level detection. The flash is updated without recompilation.

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

A node incorporates filters and detection thresholds, the system integrator updates the parameters of the node without recompilation. The memory mapping of the node can also be tuned or the dispatching of tasks assigned to other processors for performance optimization.

## 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).




Graph of nodes for stream-based computing

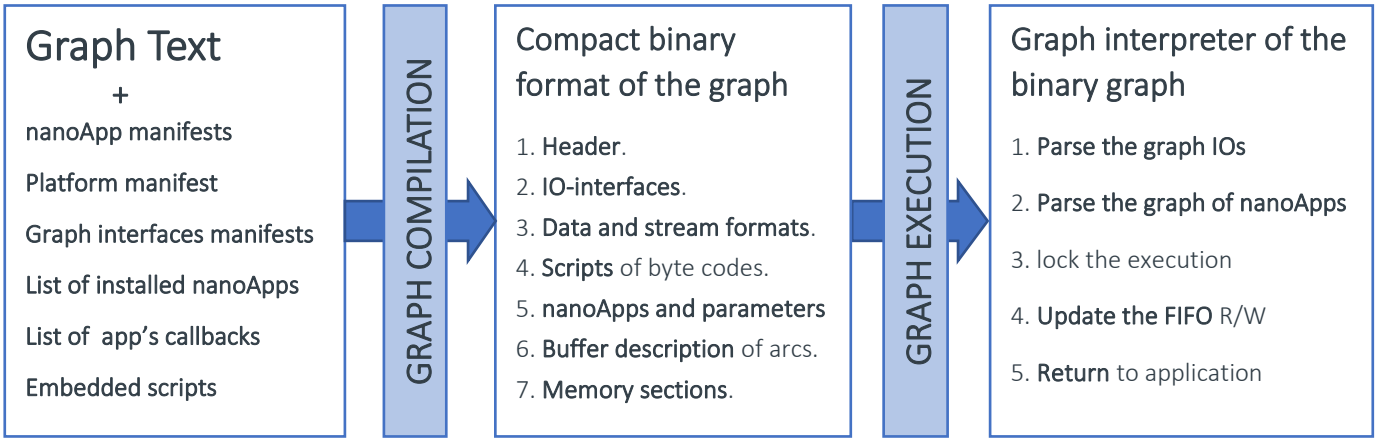
The nodes at the boundary of the graph are called "**IOs**" (as Input / Output ports). The IOs are characterized by the physical **domain** of operation (image, audio, motion sensor, GPIO, connexion the application), the commander / servant protocol used with the platform **AL** (platform abstraction layer), and the FIFO buffer declaration (buffer declared inside or outside of the graph). When IOs are exchanging data they call functions in the list of platform AL functions with an index named **IO_AL_idx**.

A **platform** is characterized by the list of software and hardware IO interface the graph can use with the IO_AL_idx index, 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**".

The graph description incorporates 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** (see picture) can be coupled before/after the execution of each node. The scripts consist in a compact **byte-code** language similar to the ones of old pocket calculators. A **global script** can be used for the interactions with the application (specific parameter settings during use-case transitions). A graph 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 on a computer, it receives the characteristics of the platform, nodes and IOs in manifests.




The Graph Interpreter is used from 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 other interface is the 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.

The graph interpreter the platform AL (abstraction layer) to manage the IO data moves, read the time or a counter, have access to a short list of critical DSP/ML subroutines optimized for the instruction-set of the platform and some functions of the standard library (memory allocation, math).

# Details

## Graph Text

### Example of graph

```
;-----------------------------------------------------------------
;   Stream-based processing using a graph interpreter :
;
;   The main script receives a code for the implementation of :
;       - 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
format                  1
format_frame_length     12
;-----------------------------------------------------------------
stream_io               0
stream_io_hwid          0       ; io_platform_data_in_0.txt
stream_io_format        0

stream_io               1
stream_io_hwid          7       ; io_platform_gpio_out_0.txt
stream_io_format        1
;
;-----------------------------------------------------------------
script 0
    script_code
        2 h16; 2002 0001        movi int16 r0 1
        1 h16; e810             equ r0,#0
        1 h16; 0381             ccallsys 1
        1 h16; C000             ret
    _end_
;-----------------------------------------------------------------
node arm_stream_filter 0

    node_preset       1         ; Q15 filter
    node_malloc_E     1         ; for debug
    node_map_hwblock  1  5      ; TCM = VID5
    node_map_swap     1  3      ; SRAM1
    node_parameters
        1  u8;  0               TAG = "all parameters"
        1  u8;  2               Two biquads
        1  u8;  1               postShift
        5 s16; 9315, 14928,  9315, 25576, -11147,
        5 s16; 9315,  5736,  9315,  9328, -26591,
    _end_
_end_
;-----------------------------------------------------------------
node arm_stream_detector 0       ; arm_stream_detector
_end_
;--------------- LAST SECTION OF THE GRAPH-----------------------
arc_section

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
arc_map_hwblock 0
```

CORRESPONDING BINARY GRAPH :

```
0x00000091,
0x00000000,
0x00009123,
0x00000010,
0x00000006,
0x00000001,
0x000000FF,
0x00000000,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFF3000,
0x00000000,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFF3802,
0x00000000,
0xFFFFFFFF,
0xFFFFFFFF,
0xFFFF1803,
0x00000000,
0x22000008,
0x467A0000,
0x00000003,
0x2200000C,
0x467A0000,
0x00000004,
0x22000008,
0x0000001F,
0x00000000,
0x00000003,
0x00030000,
0x05900004,
0xAD020200,
0x90B19261,
0x02020005,
0x000261AD,
0x01020102,
0x90B17100,
0x00029801,
0x48010000,
0x1000006F,
0x00000090,
0x00300007,
0x12310002,
0x12311D28,
0xD47563E8,
0x0B341231,
0x24701231,
0x00009821,
```

# Syntax of graphs

## *Graph scheduler control*

| Tags | Parameters (type) | Comments |
|---|---|---|
| graph_location | int | 0: location of the binary graph is all in ram (default)<br>1: keep the graph in Flash and copy in RAM the portion starting from the node linked-list<br>2: keep the graph in Flash and copy in RAM the portion starting from the arc descriptors<br>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"<br>bit 1 (2) set means "call the debug script after each naonoAppsRT is called"<br>bit 2 (4) set means "call the debug script at the end of the loop"<br>bit 3 (8) set means "call the debug script is called when starting the graph scheduling"<br>bit 4 (16) set means "call the debug script is called when returning of the graph scheduling"<br>no bit is set (default) the debug script is not called (default 0) |
| scheduler_return | int | 1: return to caller after each SWC calls<br>2: return to caller once all SWC are parsed<br>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 |

## *Graph stream formats*

| 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)<br>general (a)synchronous sensor + rescaling, .. remote data, compress<br>audio_in microphone, line-in, I2S, PDM RX<br>audio_out line-out, earphone / speaker, PDM TX, I2S,<br>gpio_in generic digital IO , control of relay,<br>gpio_out generic digital IO , control of relay,<br>motion accelerometer, combined or not with pressure and gyroscope<br>2d_in camera sensor<br>2d_out display, led matrix,<br>analog_in with aging control<br>analog_out D/A, position piezzo, PWM converter<br>rtc ticks sent from a programmable timer<br>user_interface_in button, slider, rotary button<br>user_interface_out LED, digits, display,<br>platform_3<br>platform_2 platform-specific #2, decoded with callbacks<br>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 |

## *Graph main IOs*

| 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 nodes 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<br>in a subgraph the IDX interfaces are sequential 1,2,3.. and documented like function parameters<br>in the main graph the "top_graph_interface" have the indexes to use use in the first column<br><br>IO-ID NAME INST IO FORMAT |
| arc_input | 1<br>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<br>node_name 3 1 0 | output arc index #2 connected to "node_name" instance #3 and its arc index #1, Format #0 |
| arc | <br><br>node1 1 2 0<br>node2 3 4 1 | ---ARC CONNECTION BETWEEN TWO NANOAPPS<br>NAME INST IO FMT NAME INST IO FMT<br>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)<br>1 for Working (or Scratch) area which can be reused and overlaid by other nodes<br>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<br>1 for 'fast' memory selection when possible<br>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<br>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)<br>IO stream are managed from the graph scheduler with the help of one subroutine per IO using<br>the template : typedef void (*p_io_function_ctrl) (uint32_t command, uint8_t *data, uint32_t length);<br>The "command" parameter can be : STREAM_SET_PARAMETER, STREAM_DATA_START, STREAM_STOP, STREAM_SET_BUFFER.<br><br>And one subroutine for all IOs in charge of acknowledge the end of the data move,<br>to update the circular buffer, manage overflows. This subroutine can be called from ISR<br>void arm_graph_interpreter_io_ack (uint8_t fw_io_idx, uint8_t *data, uint32_t data_size);<br>Where fw_io_idx is the index given in "top_manifest_xxxx.txt"<br><br>When the IO is "Commander" it calls arm_graph_interpreter_io_ack() when data is read<br>When the IO is "Servant" the scheduler call p_io_function_ctrl(STREAM_DATA_START, ..) to ask for<br>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))<br><br>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<br>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)<br>running mode(1) : digital conversion (BIAS always powered for analog peripherals )<br>running mode(2) : digital conversion BIAS shut-down between conversions<br>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)<br>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);<br>The "command" parameter can be : STREAM_SET_PARAMETER, STREAM_DATA_START, STREAM_STOP, STREAM_SET_BUFFER.<br><br>And one subroutine for all IOs in charge of acknowledge the end of the data move,<br>to update the circular buffer, manage overflows. This subroutine can be called from ISR<br>void arm_graph_interpreter_io_ack (uint8_t fw_io_idx, uint8_t *data, uint32_t data_size);<br>Where fw_io_idx is the index given in "top_manifest_xxxx.txt"<br><br>When the IO is "Commander" it calls arm_graph_interpreter_io_ack() when data is read<br>When the IO is "Servant" the scheduler call p_io_function_ctrl(STREAM_DATA_START, ..) to ask for<br>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<br>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))<br><br>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<br>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 :<br>Front Left FL bit0<br>Front Right FR 1<br>Front Center FC 2<br>Low Frequency LFE 3<br>Back Left BL 4<br>Back Right BR 5<br>Front Left of Center FLC 6<br>Front Right of Center FRC 7<br>Back Center BC 8<br>Side Left SL 9<br>Side Right SR 10<br>Top Center TC 11<br>Front Left Height TFL 12<br>Front Center Height TFC 13<br>Front Right Height TFR 14<br>Rear Left Height TBL 15<br>Rear Center Height TBC 16<br>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 : |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

## *gpio_in*

## *gpio_out*

control of time-stamps, sampling rate,

## *motion*

```
io_sampling_rate        1 16000 44100 48000   ; sampling rate options (enumeration in Hz)
io_sampling_period_s    1 0.01 0.02 0.04 ; sampling period options (enumeration in [second])

io_sampling_accuracy    0.8             ; sampling rate accuracy in percent

                                        ; subtype_motion aXg0m0 1                       /* only accelerometer */
                                        ; subtype_motion a0gXm0 2                       /* only gyroscope */
                                        ; subtype_motion a0g0mX 3                       /* only magnetometer */
                                        ; subtype_motion aXgXm0 4                       /* A + G */
                                        ; subtype_motion aXg0mX 5                       /* A + M */
                                        ; subtype_motion a0gXmX 6                       /* G + M */
                                        ; subtype_motion aXgXmX 7                       /* A + G + M */
io_motion_format   4                    ; imu_channel_format

io_motion_sensitivity acc   1  2 4 8 16 ; sensitivity options of accelerometer
io_motion_sensitivity gyro  1  2 4 8 16 ; sensitivity options of gyroscope [dps]
io_motion_sensitivity mag   1  2 4 8 16 ; sensitivity options of magnetometer

io_motion_averaging   acc   1  1 4 16 32; averaging in nb of samples
io_motion_averaging   gyro  1  1 4 16 32; averaging in nb of samples

io_high_pass   0 /1                     ; remove clicks
io_DC-canceller
```

## *2d_in*

```
io_raw_format_2d           (U16 + RGB16) (U8 + Grey) (U8 + YUV422)
io_trigger flash
io_synchronize with IR transmitter    https://developer.android.com/reference/android/hardware/HardwareBuffer
io_frame rate per second
io_exposure time           The amount of time the photosensor is capturing light, in seconds.
io_image size
io_modes                   portrait, landscape, barcode, night modes
io_Gain                    Amplification factor applied to the captured light. 1.0 is the default gain; more than 1.0 is brighter; less than 1.0 is darker.
io_WhiteBalanceColorTemp         Temperature parameter when using the regular HDRP color balancing.
io_WhiteBalanceColorTint         Tint parameter when using the regular HDRP color balancing.
io_MosaicPattern              Color Filter Array pattern for the colors.
io_WhiteBalanceRGBCoefficients   Custom RGB scaling values for white balance, used only if EnableWhiteBalanceRGBCoefficients is selected.
io_EnableWhiteBalanceRGBCoefficients       Enable using custom RGB scaling values for white balance instead of temperature and tint.
io_Auto White Balance     Assumes the camera is looking at a white reference, and calibrates the WhiteBalanceRGBCoefficients. Refer to the API for more
details.
io_time-stamp (none)
io_wdr;                    wide dynamic range flag (tuya)
io_watermark;              watermark insertion flag (tuya)
io_flip;                   image format (portrait, panoramic)
io_night_mode;             motion detection sensitivity (low, medium, high)
io_detection_zones;        + {center pixel (in %) radius}, {}, {}
io_focus_area
io_auto exposure on focus area
io_focus_distance          forced focus to infinity or xxx meters
io_get_distance            from focus area
io_zoom_area
```

```
io_time_stamp;             detection time-stamp format
io_light_detection;
io_jpeg_quality
io_sound_detection;        sound level
io_other sensors;          humidity, battery%
```

## *2d_out*

```
8b backlight brightness control
```

# 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  sssssssssssqqqq q14.2   1 hour + 8mn +/- 0.0625
STREAM_TIME16D   40  qqqqqqqqqqqqqqqq q15 [s] time difference +/- 15us
STREAM_TIME32    41  sssssssssssssssssssssssssqqqq q28.4  [s] (8.5 years +/- 0.0625s)
STREAM_TIME32D   42  sssssssssssssssqqqqqqqqqqqqqqqq q17.15 [s] (36h, +/- 30us) time difference
STREAM_TIMESTMP  43  sssssssssssssssssssqqqqqqqqqqq q20.12 [s] (12 days, +/- 0.25ms)
STREAM_TIME64    44  ____sssssssssssssssssssssssssssssssqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq q32.28 [s] 140 Y +Q28 [s]
STREAM_TIME64MS  45  _____________________mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm 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

### Use-case

## Embedded scripts

A virtual machine with 6 registers, 6 pointers, loop counter and a stack. Instructions are coded with 2 bytes.

## 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:
    - Returns the details of current processor: its index in the manifest table, its architecture and the FPU options
    - Call one on the three functions used to control the device drivers : “set”, “start”, “stop” (see “2) The graph boundaries”)
    - 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 lis**t 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
  - paragraph below “1) Platform and SWC manifests”
- the way the **IOs** are sharing data with the ring buffers at the boundary of the graph
  - paragraph below “2) The graph boundaries”
- the description of the **linked-list** and the connexions between arcs and nodes
  - 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
  - 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)).
  - 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
  - The buffer length is computed with a 22bits linear address and a 2 bits shifter to extend the length.
  - The 2 bits selector give the indication of linear or ring descriptor format
  - 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
  - The read index is a plain 24bit index in Bytes, and allows to manage 16MB buffers
  - 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.
    - 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
    - The write index is a plain 24bit index in Bytes. Buffer full condition corresponds to (W-R) = (Size-1).
    - 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.
    - 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:
    - "`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.."
    - "`FMT_DEINTERLEAVED_1PTR`" : interleaved format of frame-size ("LLLLLRRRRR LLLLLRRRRR..")
    - "`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:
    - No timestamp on the data frames
    - 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.
    - 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*