Table of content

Stream-based processing with a graph interpreter

What

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

1. **Accelerate time to market**

Graph-Interpreter helps system integrators and OEM who develop complex DSP/ML stream processing. It allows going fast from prototypes validated on a computer to the final tuning steps on production boards, by updating a graph of computing nodes and their coefficients without device recompilation.

2. **NanoApps repositories**

It provides an opaque interface of the platform memory hierarchy to the computing nodes. It arranges the data flow is translated to the desired formats of each node. It prepares the conditions where nodes will be delivered from a Store.

3. **Portability, scalability**

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

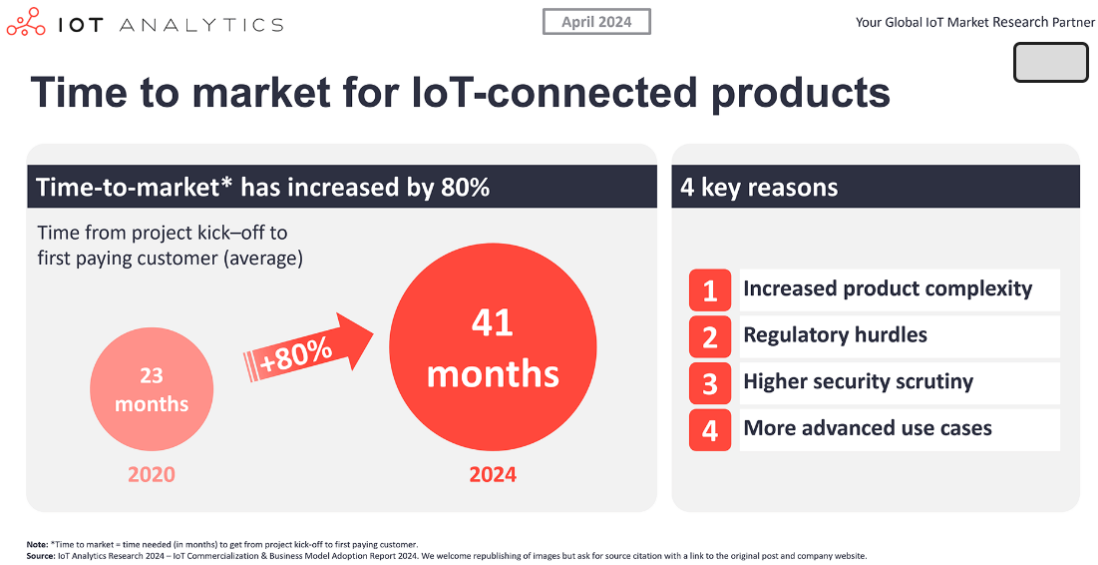
Why

The complexity of IoT systems using signal processing and machine-learning is continuously rising. In four years (picture below) we have seen the average time-to-market going from months to years. We must ease the tasks of the integrators by **splitting the problems in small pieces**, which translates in the definition of standard interfaces between those pieces.

Here are some examples of signal processing “pieces” and software portability issues :

- an algorithm is extracting metadata from a pressure sensor, the samples of which are a stream of floating-point data at 10Hz sampling rate. Can the algorithm be ported as-is to a platform using a pressure sensor using 16bits integers at 25Hz sampling rate ?
- a pattern recognition algorithm is using images of format 300x300 pixels RGB888. What happens when the platform is using a sensor with VGA image format ?
- an industrial proximity detector is using a 25kHz wave generator and an ultrasound echo detector using a stream of Q15 samples at 96kHz normalized at 120dBSPL full-scale. Can we manage the same behavior and performance with a 88.1kHz sampling-rate ?
- an audio algorithm using 50kB of RAM from which 4kB are critical on speed access and 25kB have no speed constraint. What happens when several algorithms, or several instances of the same, want to use the fast tightly-coupled memory bank (TCM), how do we manage data swapping before/after calling the algorithms ?
- a motion sensor subsystem is designed to integrate components from different silicon vendors. How do we manage automatically the scaling factors associated with the sensors, to have the same dynamic range and sampling-rates in the data stream ?
- a microprocessor has a dot-product and an FFT accelerator. Can we offer an abstraction layer to the algorithm designers for such coprocessors : the developer will release one single software. The computation of the FFT will use software libraries when there is no coprocessor.

Creating standard interfaces allows software component developers to deliver their IP without having to care about the capabilities of the platform used during system integration.



We want the algorithms developer to focus on their domain of expertise without creating a dependency with the protocols used in the graph or the data formats used by the preceding and following nodes of the graph.

The data format translators (provided with the graph scheduler) consists in changing :

- the data frame length and the interleaving scheme (block or sample-based)
- the raw sample data format (pixel format, integer / floating point samples)
- the sampling-rate and the management of time-stamps
- the scaling of the data with respect to standard physical units (see, RFC8428 and RFC8798)

Computing nodes and platforms have to explain in "Manifests" their interfaces in a formal way.

We want to anticipate the creation of Stores of computing nodes, with a key (specific to a platform) exchange protocol, when the node is delivered in a binary format or obfuscated source code.

We want to let the graph to be modified without needing to recompile and re-flash the entire application. The graph will incorporate sections of interpreted code to manage state-machines, scripting, parameters updates and to interface with the application.

How

Graph Interpreter is a scheduler and interpreter of a binary representation of a graph. For portability reason the Graph Interpreter uses a minimal platform abstraction layer (AL) to the memory and to the input/output stream interfaces. Graph Interpreter manages the data flow of "arcs" between "nodes".

This binary graph description is a compact data structure using indexes to the physical addresses of the nodes and memory instances. This graph description is generated in three steps:

1. plaform manifest and IO manifest are prepared ahead of the graph design and describe the hardware. The manifests are giving the processing capabilities (processor architecture, minimum guaranteed amount of memory per RAM blocks and their speed, TCM sizes). The platform manifest gives references to node manifests for each of the installed processing Nodes : developer identification, input/output data formats, memory consumption, documentation of the parameters and a list of "presets", test-patterns and expected results (see also node design).
2. The graph is either written in a text format (syntax example here) or is generated from a graphical tool (proof of concept picture of the GUI here).
3. **the binary file to be used on the target is generated / compiled**. The file format is either a C source file, or a binary table to load in a specific flash memory block, to allow quick tuning cycles without full recompilation.

The platform provides and abstraction layer (AL) with the following services:

1. **Share the physical memory map base addresses**. The graph is using indexes to the base addresses of 63 different *memory banks*: for example shared external memory, fast shared internal, fast private per processor (TCM), and indexed the same way for multiprocessing without MMU. The AL shares the entry points of the nodes installed in the memory space of the processor.
2. **Interface with the graph boundary generating/consuming data streams,** declared in the platform manifest and addressed as indexes from the scheduler when the FIFOs at the boundary of the graph are full or empty.
3. **Share information for scripts**. The graph embeds byte-codes of "Scripts" used to implement state-machines, to change nodes parameters, to check the arcs data content, trigger GPIO connected to the graph, generate strings of characters to the application, etc. The Scripts provide a simple interface to the application without code recompilation.

Graph-Interpreter is delivered with a generic implementation of the above services for computers, with device drivers emulated using data files, time information emulated with counters. The Graph-Interpreter is delivered as open-source.

How (detailed)

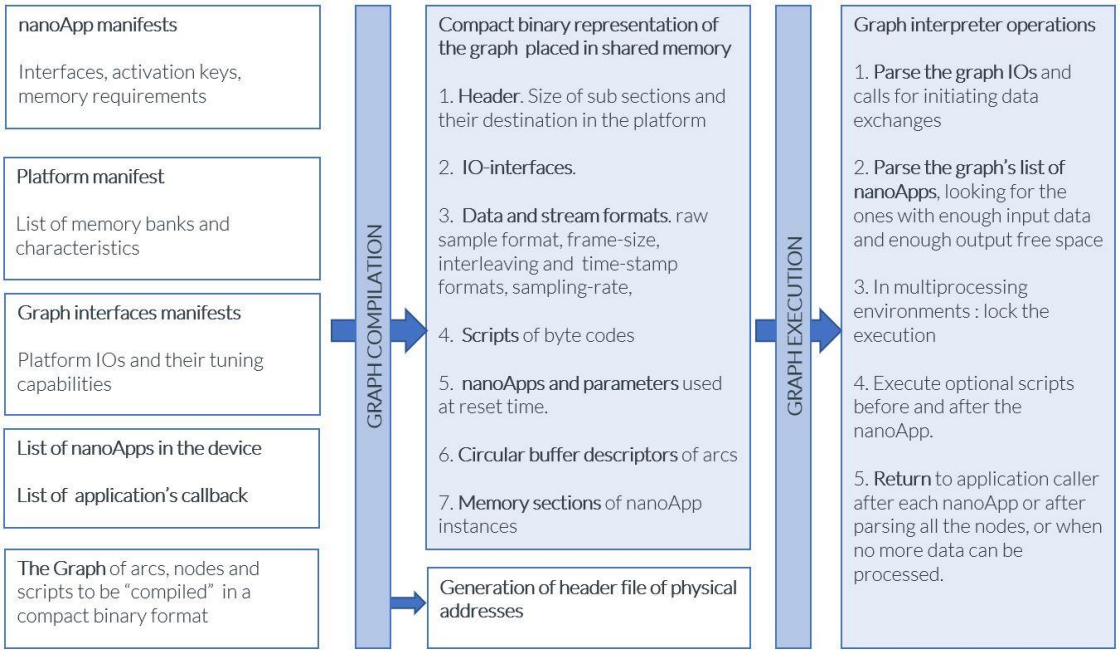
Stream-based processing is facilitated using Graph-Interpreter:

1. The Graph Interpreter has only **two functions**. One entry point for the application `void arm_graph_interpreter()` , and one entry point for the data moves: a function used to tell the data moves with the outside of the graph are done `void arm_stream_io_ack()`.
2. Nodes can be written in **any computer languages**. The scheduler is addressing the nodes from a **single entry point** using a 4-parameters API format. There is no restriction in having the nodes delivered in binary format, compiled with "**position independent execution**" option. There is no dynamic linking issue: the nodes delivered in binary can still have access to a subset of the C standard libraries through a Graph-Interpreter **service**. The nodes are offered the access to DSP/ML kernels (compiled without the position-independent option) or executed with platform-specific accelerators: the execution speed will scale with the targeted processor capabilities without recompilation.
3. **Drift management.** The streams don't need to be perfectly isochronous (the situation happens when peripherals are using different clock trees). Drift and rate conversion service is provided by nodes delivered with the interpreter. The graph defines different quality of services (QoS). When a "main" stream is processed with drifted secondary streams the time-base is adjusted to the highest-QoS streams (minimum latency and distortion), leaving the secondary streams managed with interpolators in case of flow issues.

4. Graph-Interpreter manages **TCM access**. When a Node declares, in its manifest, the need for a "critical speed memory bank" of small size (ideally less than 16kBytes), the graph compilation step will allocate it to TCM area, and can arrange data swapping if several nodes have the same need.
5. **Backup/Retention RAM**. Some applications are requiring a fast recovery in case of failures ("warm boot") or when the system restores itself after deep-sleep periods. One of the memory banks allows developers to save the state of algorithms for fast return to normal operations. The node retention memory should be limited to tens of bytes.
6. Graph-Interpreter allows memory size optimization with overlays of different nodes' scratch memory banks.
7. **Multiprocessing** SMP and AMP with 32+64bits processor architectures. The graph description is placed in a shared memory. Any processor having access to this shared memory can contribute to the processing. Buffer addresses are described with a 6-bits offset and an index, to let the same address be processed without MMU. The node execution reservation protocol is defined in the AL, with a proposed lock-free algorithm. The nodes execution can be mapped to a specific processor and architecture. The buffers associated to arcs can be allocated to a processor's private memory-banks.
8. **Scripting** are designed to avoid going back and forth with the application interfaces for simple decisions, without the need to recompile the application (Low-code/No-code strategy, for example toggling a GPIO, changing node parameter, building a JSON string...) the graph scheduler interprets a compact byte-stream of codes to execute simple scripts.
9. **Process isolation**. The nodes never read the graph description data. The arc descriptors and the memory mapping is designed for the use of hardware memory protection.
10. **Format conversions**. The developer declares, in the manifests, the input/output data formats of the node. The Graph Interpreter implements the format translation between nodes: sampling-rates conversions, changes of raw data, removal of time-stamps, channels de-interleaving. Specific conversion nodes are inserted in the graph during its binary file translation.
11. Graph-Interpreter manages the various **methods of controlling I/O** with one function per IO: parameters setting and buffer allocation, data move, stop, mixed-signal components settings.
12. Graph-Interpreter is **open-source**, and portable to 32-bits processors and computers.
13. Example of Nodes: image and voice codec, data conditioning, motion classifiers, data mixers. Graph-Interpreter comes with short list of components doing data routing, mixing, conversion and detection.
14. **From the developer point of view**, it creates opaque memory interfaces to the input/output streams of a graph, and arranges data are exchanged in the desired formats of each Node. Graph-Interpreter manages the memory mapping with speed

constraints, provided by the developer, at instance creation. This lets software run with maximum performance in rising situations of memory bounded problems.

15. **From the system integrator view**, it eases the tuning and the replacement of one Node by another one and is made to ease processing split with multiprocessors. The stream is described with a graph (a text file) designed with a graphical tool. The development of DSP/ML processing will be possible without need to write code and allow graph changes and tuning without recompilation.
16. **Graph-Interpreter design objectives**: Low RAM footprint. Graph descriptor can be placed in Flash with a small portion in RAM. Use-cases go from small Cortex-M0 with 1kBytes RAM (about 200Bytes of stack and 100Bytes of static RAM) to SMP/AMP/coprocessor and mix of 32/64bits thanks to the concept of shared RAM and indexes to memory banks provided by the local processor abstraction layer. Each arc descriptors can address buffer sizes of up to 64GBytes in each of the 64 memory banks.



The development flow is :

1) The platform provider is producing a manifest of the processor and IO interface, jointly with the AL abstraction layer and an optional list of callbacks giving specific services of the platform.
2) The node software developer is producing the code and the corresponding manifest
3) Finally, the system integrator creates a binary file representing the graph of the DSP/ML components of its application. The system integrator adds other callbacks which will be used by the scripting capability of the graph.

How to start

It depends who you are.

User	Actions
platform vendor	deliver an abstraction layer and a manifest of the platform capabilities : memory mapping, computing services, data streaming interface
software developer	deliver a program with a manifest of the node capabilities: IO port description (data format), memory consumption
system integrator	use a GUI or the graph description language to create the arcs between the nodes, and progressively, if needed, tune the performance (memory overlays, FIFO sizes, node affinity to processors)

Graph Interpreter instance

The Graph Interpreter has two functions. One entry point `void arm_graph_interpreter()` , and a function used to tell the data moves with the outside of the graph are confirmed `void arm_graph_interpreter_io_ack()`. The two functions can be called once the instance is created by the platform AL (`void platform_init_stream_instance(arm_stream_instance_t *S)`).

Interpreter instance structure:

name of the field	comments
long_offset	A pointer to the table of physical addresses of the memory banks (up to 64). The table is in the AL of each processor. The graph is not using physical memory but offsets to one of those 64 memory banks, defined in the “platform manifest”. This table allows memory translation between processors, without the need of MMU.
linked_list	pointer to the linked-list of nodes of the graph
platform_io	table of functions (IO_AL_idx) associated to each IO stream of the platform
node_entry_points	table of entry points to each node (see “TOP” manifest)
application_callbacks	the application can propose a list of functions to be called from scripts. One callback will serve as entry point for nodes generating Metadata information in natural language text, ready to be digested by specialized small language models.
al_services	pointer to the function proposing services (time, script, stdlib, compute)
iomask	bit-field of the allowed IOs this interpreter instance can trigger
scheduler_control	execution options of the scheduler (return to the application after each node execution, after a full graph parsing, when there is no

name of the field	comments
	data to process), processor identification.
script_offsets	pointer to scripts used as subroutines for the other scripts placed in the node “arm_stream_script” parameter section.
all_arcs	pointer to the list of arc descriptors (structures giving the base address, size of the associated circular buffer, read, write index, data format of the arc consumer/producer, and debug/trace information).
all_formats	pointer to the section of the graph describing the stream formats. This section is in RAM.
ongoing	pointer to a table of bytes associated to each IO ports of the graph. Each byte tells if a transfer is on-going.

Graphical view of the memory mapping

Platform Manifest

The “Top” platform manifest has four sections :

- the list of file paths to ease readability
- the name of the manifest file describing the processing architecture (see next paragraph)
- the list of available data stream ready to be connected to a graph : this list corresponds to manifest files of the data format options, and gives an index (IO_AL_idx) to the function to be called to read/write data and update the stream configuration.
  - ; path: path ID ; Manifest manifests file ; IO_AL_idx index used in the graph ; ProcCtrl processor ID affinity bit-field ; ClockDomain provision for ASRC (clock-domain) ; some IO can be alternatively clocked from the system clock (0) ; or other ones. The system integrator decides with this field to ; manage the flow errors with buffer interpolation (0) or ASRC (other clock domain index) ; The clock domain index is just helping to group and synchronize the data flow per domain.
- the list of the nodes already installed in the device. This is also a list of manifest files giving a formal way to describe how to connect the nodes each others

```
;=============================================================
; TOP MANIFEST :
;   - paths to the files
;   - shared memories
;   - list of processors and their private memories and IOs
;   - list of the nodes installed in each processor and/or architectures
;=============================================================
    6                       six file paths
```

```
    0 ../../../stream_platform/
    1 ../../../stream_platform/computer/manifest/
    2 ../../../stream_nodes/arm/
    3 ../../../stream_nodes/signal-processingFR/
    4 ../../../stream_nodes/bitbank/
    5 ../../../stream_nodes/elm-lang/



;=====================================================================
    3           number of processors
    2           number of shared memory

;   processor and architecture ID are in the range [1..7]
;
;   Access    0 data/prog R/W, 1 data R, 2 R/W, 3 Prog, 4 R/W
;   Speed     0 slow/best effort, 1 fast/internal, 2 TCM
;   Type      0 Static, 1 Retention, 2 scratch mem

;---MEMID SIZE  A S T   Comments
    0    8000   1 0 0   shared memory
    3    1000   1 0 1   simulates shared retention memory




;=====================================================================
;   Processor #1 - architecture #1 , two processors
;   proc ID, arch ID, main proc, nb mem, service mask, I/O
    1        1        1          2       15            7

;---MEMID SIZE  A S T   Comments
    1    1000   1 2 0   simulates DTCM
    2    1000   1 2 1   simulates ITCM

;-----IO AFFINITY WITH PROCESSOR 1-------------------------------
    ;Path      Manifest       IO_AL_idx    Comments
    1 io_platform_data_in_0.txt       0    application proc
    1 io_platform_data_in_1.txt       1    application proc
    1 io_platform_analog_sensor_0.txt 2    ADC
    1 io_platform_audio_in_0.txt      4    microphone
    1 io_platform_line_out_0.txt      6    audio out stereo
    1 io_platform_gpio_out_0.txt      7    GPIO/LED
    1 io_platform_data_out_0.txt      9    application proc
```

```
;=================================================================
;   Processor #2
;   proc ID, arch ID, main proc, nb mem, service mask, I/O
    2        1        0          2       15            2

;---MEMID SIZE  A S T   Comments
    1    1000   1 2 0   index 1/2 point to different physical addresses
    2    1000   1 2 1   index 1/2 point to different physical addresses

;------IO AFFINITY WITH PROCESSOR 2-------------------------------
    ;Path   IO Manifest        IO_AL_idx    Comments
    1 io_platform_data_in_0.txt        0    shared with Proc 1
    1 io_platform_motion_in_0.txt      3    accelero=gyro




;=================================================================
;   Processor #3 - new architecture, one processor
;   proc ID, arch ID, main proc, nb mem, service mask, I/O
    1        2        0          0       15            2

;---MEMID SIZE  A S T   Comments

;------IO AFFINITY WITH PROCESSOR 2-------------------------------
    ;Path   IO Manifest        IO_AL_idx    Comments
    1 io_platform_2d_in_0.txt          5    camera
    1 io_platform_gpio_out_1.txt       8    GPIO/PWM




;======== ALL NODES ============================================
; scheduler algorithm :
; if the node archID > 0 then check compatibility with processor archID and
exit
; if the node procID > 0 then check compatibility with processor procID and
exit
;
;   Path                  Node Manifest                PROC ARCH |  ID
    2       script/node_manifest_script.txt              0   0   |   1 runs
everywhere
    2       router/node_manifest_router.txt              0   1   |   2 SMP on
archID-1
    2    amplifier/node_manifest_amplifier.txt           0   1   |   3
    2       filter/node_manifest_filter.txt              0   1   |   4
    2    modulator/node_manifest_modulator.txt           0   1   |   5
    2  demodulator/node_manifest_demodulator.txt         0   1   |   6
```

```
    3     detector/node_manifest_detector.txt           0   1   |   7
    3    resampler/node_manifest_resampler.txt          0   1   |   8
    3   compressor/node_manifest_compressor.txt         0   1   |   9
    3 decompressor/node_manifest_decompressor.txt       0   1   |  10
    4      JPEGENC/node_manifest_bitbank_JPEGENC.txt    0   1   |  11
    5      TJpgDec/node_manifest_TjpgDec.txt            0   1   |  12
;-------------------------------------------------------------  |     ----
    2      filter2D/node_manifest_filter2D.txt          2   2   |  13 only
archID-2
    3    detector2D/node_manifest_detector2D.txt        0   2   |  14 single
processor
;
;
end ; the platform manifest ends here
;================================================================
```

IO Manifest

The graph interface manifests ("IO manifests") is a text file of commands used by the graph compiler to understand the type of data flowing through the IOs of the graph. A manifest gives a precise list of information about the data format (frame length, data type, sampling rate, …). This paragraph gives the syntax used in an IO manifest.

Am IO manifest starts with a *Header* of general information, followed by the list of command used for the description of the stream. The IO manifest reader is assuming default values and associated command are useless and inserted by the programmer for information and readability.

Because of the variety of stream data types and setting options, the graph interpreter introduces the concept of physical "*domains*" (audio, motion, 2D, …). The document gives starts with the list of general information for the stream digital format, followed by the specification of domain-related information.

IO manifest header

The "IO manifest" starts with the name which will be used in a GUI design tool, followed by the "domain" (list below).

Example of a simple IO manifest :

```
    io_name     io_platform_sensor_in_0  ; the IO name
    io_domain   analog_in                ; the domain of operation
```

List of IO Domains

See Stream format "domains"

Declaration of options

The manifest gives the list of **options** possible described as a **list**, or as a **range** of values. The syntax is : an index and the list of numbers within brackets "{" and "}". The index gives the default value to consider in the list. Index "1" corresponds to the first element of the list. Index value "0" means "any value". The list can be empty in that case.

Example of a list of options between five values, the index is 2 meaning the default value is the second in the list (value = 6).

```
{ 2   5 6 7 8 9 }
```

When the index is negative the list is decoded as a "range". A Range is a set of three fields :

- the first option
- the step to the next possible option
- the last (included) option

The absolute index value selects the default value in this range.

Example of an option list of values (1, 1.2, 1.4, 1.6, 1.8, .. , 4.2), the index is -3 meaning the default value is the third in the list (value = 1.4).

```
{ -3    1 0.2 4.2 }
```

Default values of an IO manifest

When not mentioned in the manifest the following assumptions are :

io manifest command	default value	comments
io_commander0_servant1	1	servant
io_set0copy1	0	buffer address is set by the scheduler
io_direction_rx0tx1	0	data flow to the graph
io_raw_format	float32	float is the default data format
io_interleaving	0	raw data interleaving
io_nb_channels	1	mono
io_frame_length	1	one sample (mono or multichannel)

Common information of all digital stream

IO manifests describe the stream data format and how to copy/use the data. This is common to the digital streams of any IO. The next section ("IO Controls Bit-fields per domain") is specific to each domain of operation.

io_commander0_servant1 "0/1"

The IO is "commander" when it initiates the data exchanges with the graph without the control from the scheduler (for example an audio Codec). It is "servant" when the scheduler

needs to pull or push asynchronously the data by calling the AL IO functions (for example an interface to the main application). IO stream are managed from the graph scheduler with the help of one subroutine per IO (IO_AL_idx function of the AL) using the template (see also next section):

```
typedef void (*p_io_function_ctrl) (uint32_t command, void *data, uint32_t
length);
```

The "command" parameter can be : STREAM_SET_PARAMETER (set the domain-specific IO parameters), STREAM_DATA_START (initiate a data exchange), STREAM_STOP, STREAM_SET_BUFFER (tell the scheduler the default IO interface buffer location).

Once the move is done the external IO driver calls `arm_graph_interpreter_io_ack()` to tell the scheduler to update the corresponding arc.

```
void arm_graph_interpreter_io_ack (uint8_t graph_io_idx, void *data, uint32_t
size);
```

Example :

```
io_commander0_servant1  1  ; default is servant (1)
```

Graph interpreter implementation details

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.

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_RUN, ..) to ask for data move. Once the move is done the IO driver calls arm_graph_interpreter_io_ack()

io_set0copy1 "0/1"

Declares if the IO stream is using a pointer provided by the scheduler (the pointer is **set** value 0), or if the data needs to be copied from the IO internal buffer to the graph arc buffer (**copy** value 1).

```
io_set0copy1      1     ; data will be copied from/to the IO pointer to/from
the arc buffer (rx/tx IO)
```

io_direction_rx0tx1 "0/1"

Declaration of the direction of the stream from the graph scheduler point of view.

```
io_direction_rx0tx1   1  ; direction of the stream  0:input 1:output
```

io_raw_format "option"

Declaration of the size and type of the raw Data Types using the Declaration of options format.

```
io_raw_format {1 17}  ; raw arithmetic's computation format is STREAM_S16
```

io_interleaving “0/1”

```
io_interleaving    1  ; multichannel interleaved (0), deinterleaved by frame-
size (1)
```

io_nb_channels “option”

Declaration of the possible number of channels using the Declaration of options format.

```
io_nb_channels {2  1 2 3 4}  ; options for the number of channels, stereo
default
```

io_frame_length “option”

Declaration of the possible frame length using the Declaration of options format, in samples. A sample can be multichannel but is still counted as one sample.

```
io_frame_length  {1 1 2 16 }   ; options of possible frame_length in samples
```

io_frame_duration “U option”

Declaration of the possible frame duration using the Declaration of options format, in a time unit given in the first parameter. See Standard units.

```
io_frame_duration  69 {1 1 2 16 }   ; options of possible frame_length in
minutes
```

io_setup_time “x”

Information of the time it takes before valid / calibrated samples are ready for processing after reset. This is “for information” and given for documentation purpose.

```
io_setup_time  12.5 ; wait 12.5ms before receiving valid data
```

io_units_rescale “u a b max”

Syntax : “physical unit name” “coefficient a” “coefficient b” “maximum value”. Rescaling information between normalized (1.0 or 0x7FFF) digital (“D”) and physical (“P”) units with the formulae P = a x (D - b).

See file “Table.md” for the list of available Units from RFC8798 and RFC8428.

```
io_units_rescale  VRMS  0.0135  -10.1  0.15

; V_physical = a x (X_sample - b) with the default hardware settings
; V [VRMS] <=> 0.0135 x ( X - (-10.1))
; 0.15 VRMS <=> 0.0135 x (1.0 - (-10.1))  0.15 Vrms corresponds to digital
full-scale
```

io_subtype_multiple "..."

Multiple units interleaved streams with rescaling factors of above "io_units_rescale". Used for example with motion sensors delivering acceleration, speed, magnetic field, temperature, etc ..

```
io_subtype_multiple DPS a b max GAUSS a b max
```

io_position "U 3D"

Declaration of the position in a unit given in the first parameter. See Standard units.

```
io_position 98  1.1 -2.2 0.01 ; centimeter=98 and relative XYZ position with
the platform reference point
```

io_euler_angles "U 3A"

Relative angles of the IO in the platform reference space. See Standard units.

```
io_euler_angles 66 10 20 88.5  ; degree=66  Euler angles with respect to the
platform reference orientation, in degrees
```

io_sampling_rate "U option"

IO stream sampling rate in a frequency unit given in the first parameter. See Standard units.

```
io_sampling_rate 9 {2 16e3 44.1e3 48000} ; sampling rate options in Hz=9
```

io_sampling_period "U option"

Declaration of the sampling period using the Declaration of options format, in a time unit given in the first parameter. See Standard units.

```
io_sampling_period  4 {1 1 60 120 }  ; sampling period, enumeration in
seconds (4)
```

io_sampling_rate_accuracy "p"

Percentage of random inaccuracy of the given sampling rate.

```
io_sampling_rate_accuracy  0.01   ; in percentage, or 100ppm
```

io_time_stamp_format "option"

See file "Table.md" for the definition of time-stamp format inserted before each frame :

- 0: no time stamp
- 1: simple counter
- 39: STREAM_TIME16 format q14.2 in seconds, maximum range 4 hours 30mn, 0.25s steps
- 40: STREAM_TIME16D format q1.15 2 seconds, for time differences, step=30us

- 41: STREAM_TIME32 format q30.2 seconds, maximum 34 years , 0.25s steps
- 42: STREAM_TIME32D format q17.15 seconds, maximum 36hours steps 30us for time differences
- 43: STREAM_TIME64 format q32.26 seconds, maximum 140 years +/- 4ns
- 44: STREAM_TIME64MS format u42 in milliseconds, maximum 140 years
- 45: STREAM_TIME64ISO ISO8601 with signed offset, example 2024-05-04T21:12:02+07:00

```
io_time_stamp_format {1 39 41 } ; time-stamp format options
```

IO Controls Bit-fields per domain

The graph starts with a table of 4 words per IO. The first word is used to connect the IO with the graph (arc index, direction, index of the AL function associated to). Three 32bits words are reserved for specific tuning items of their domains, they are named W1, W2 and W3 below.

Domain audio_in and audio_out

Domain audio_in setting word 1

Channel mapping with a bit-field (20 channels description see audio channels) :

```
io_channel_mapping  0x0B       ; Front Left + Right + LFE

Name                          bit position
Front Left            FL          0
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
```

Graph syntax example :

```
stream_io_setting 15 ; selection of the four first channels
```

Domain audio_in setting word 2

Control of the gains and filters

```
io_audio_analog_gain     {1  0 12 24       }   ; analog gain (PGA)
io_audio_digital_gain    {-1 -12 1 12      }   ; digital gain range
io_audio_hp_filter       {1 1 20 50 300    }   ; high-pass filter (DC
blocker) ON(1)/OFF(0)
io_audio_agc              0                    ; agc automatic gain control,
ON(1)/OFF(0)
io_audio_router          {1  0 1 2 3       }   ; router  from AMIC0 DMIC1 HS2
LINE3 BT/FM4
io_audio_gbass_filter    {1  1  1  0 -3 3 6}   ; ON(1)/OFF(0) options for
gains in dB
io_audio_fbass_filter    {1  20 100 200    }   ; options for frequencies
io_audio_gmid_filter     {1  1  1  0 -3 3 6}   ; ON(1)/OFF(0) options for
gains in dB
io_audio_fmid_filter     {1  500 1000      }   ; options for frequencies
io_audio_ghigh_filter    {1  1  0 -3 3 6   }   ; ON(1)/OFF(0) options for
gains in dB
io_audio_fhigh_filter    {1  4000 8000     }   ; options for frequencies
```

field name	nb bits	comments
io_analog_gain	3	analog gain (PGA)
io_digital_gain	4	digital gain range
io_hp_filter	2	high-pass filter (DC blocker) ON(1)/OFF(0)
io_agc	1	agc automatic gain control, ON(1)/OFF(0)
io_router	3	router from AMIC0 DMIC1 HS2 LINE3 BT/FM4
io_gbass_filter	3	bass gain in dB
io_fbass_filter	2	filter frequencies
io_gmid_filter	3	mid frequency gains in dB
io_fmid_filter	2	filter frequencies
io_ghigh_filter	3	high frequency gain in dB
io_fhigh_filter	2	filter frequencies

Domain audio_in setting word 3

Not used

Domain gpio_in and gpio_out

Domain gpio_in setting word 1

Field name	nb bits	comments
State	3	High-Z, low, high
type	3	PWM, motor control, GPIO
control	3	PWM duty, duration, frequency (buzzer)

Domain gpio_in/out setting word 2 and word 3 are not used.

Domain motion

Domain motion setting word 1

Selection of the multichannel interleaving :

```
aXg0m0 1 only accelerometer
a0gXm0 2 only gyroscope
a0g0mX 3 only magnetometer
aXgXm0 4 A + G
aXg0mX 5 A + M
a0gXmX 6 G + M
aXgXmX 7 A + G + M

offset removal on A,M,G
Metadata pattern detection activation and sensitivity
```

Domain motion setting word 2 and word 3 are not used.

Domain 2d_in and 2d_out

Domain 2d setting word 1

Domain 2d setting word 2

Domain 2d setting word 3

Feature name	bits	Description
io_raw_format_2d	3	(U16 + RGB16) (U8 + Grey) (U8 + YUV422) https://gstreamer.freedesktop.org/documentation/additional/design/mediatype-video-raw.html?gi-language=c YCbCr 4:2:2 (16b/pixel), RGB 8:8:8 (24b/pixel)
io_trigger flash	4	activate the flash when polling a new image
io_synchronize_IR	2	sync with IR transmitter

Feature name	bits	Description
io_frame rate per second	1	
io_exposure time	3	The amount of time the photosensor is capturing light, in seconds.
io_image size	3	
io_modes	2	portrait, landscape, barcode, night modes
io_Gain	3	Amplification factor applied to the captured light. >1.0 is brighter <1.0 is darker.
io_WhiteBalanceColor	2	Temperature parameter when using the regular HDRP color balancing.
io_MosaicPattern	3	Color Filter Array pattern for the colors
io_WhiteBalanceRGBCoef	2	RGB scaling values for white balance, used only if EnableWhiteBalanceRGBCoefficients is selected.
io_WhiteBalanceRGBCoef	3	Enable using custom RGB scaling values for white balance instead of temperature and tint.
io_Auto White Balance	4	Assumes the camera is looking at a white reference, and calibrates the WhiteBalanceRGBCoefficients
io_wdr	2	wide dynamic range
io_watermark	1	watermark insertion
io_flip	3	image format
io_night_mode	3	
motion detection	2	sensitivity (low, medium, high)
io_detection_zones	3	+ {center pixel (in %) radius}, {}, {}
io_focus_area	2	
io_auto exposure	3	on focus area
io_focus_distance	2	forced focus to infinity or xxx meter
io_jpeg_quality	2	compression level
io_backlight brightness control	2	2D rendering forced focus to infinity or xxx meter

Domain analog_in and analog_out

```
aging coefficient
```

Comments section for IOs

Information examples :

- jumpers to set on the board
- manufacturer references for components and internet URLs

- any other system integration warning and recommendations

Node manifest

A node manifest file gives the name of the software component, the author, the targeted architecture, the description of input and output streams connected to it.

The graph compiler allocates a predefined amount of memory and this file explains the way to compute the memory allocation. See Declaration of options for the option syntax.

Example of node manifest

```
; ------------------------------------------------------------------------
-----------
; SOFTWARE COMPONENT MANIFEST - "stream_filter"
; ------------------------------------------------------------------------
-----------
;
node_developer_name    ARM          ; developer name
node_name             stream_filter ; node name

node_mask_library      64           ; dependency with DSP services

;-------------------------------------------------------------------------
------------
;   MEMORY ALLOCATIONS

node_mem                0           ; first memory bank (node instance)
node_mem_alloc         76           ; amount of bytes

node_mem                1           ; second memory bank (node fast working
area)
node_mem_alloc         52           ;
node_mem_type           1           ; working memory
node_mem_speed          2           ; critical fast
;-------------------------------------------------------------------------
-----------
;    ARCS CONFIGURATION
node_arc                0
node_arc_nb_channels    {1 1 2}     ; arc intleaved,  options for the number
of channels
node_arc_raw_format     {1 17 27}   ; options for the raw arithmetics
STREAM_S16, STREAM_FP32

node_arc                1
node_arc_nb_channels    {1 1 2}     ; options for the number of channels
node_arc_raw_format     {1 17 27}   ; options for the raw arithmetics
STREAM_S16, STREAM_FP32
```

```
end
```

The nodes have the same interface :

```
void (node name) (uint32_t command, void *instance, void *data, uint32_t
*state);
```

Node are called with parameter “data” being a table of arc data structures of two fields :

- a pointers the arc buffer
- the amount of data in byte placed after the above address (input arcs) and free space available (output arcs)

The nodes returns after updating the second field of the structures :

- The amount of data consumed, for RX arcs
- The amount of data produced in the TX arcs

Default values of a node manifest

When not mentioned in the manifest the following assumptions are :

io manifest command	default value	comments
io_commander0_servant1	1	servant
io_set0copy1	0	in-place processing
io_direction_rx0tx1	0	data flow to the graph
io_raw_format	float32	float is the default data format
io_interleaving	0	raw data interleaving
io_nb_channels	1	mono
io_frame_length	1	one sample (mono or multichannel)

Manifest header

The manifest starts with the identification of the node.

node_developer_name “name”

Name of the developer/company having the legal owner of this node. Example:

```
node_developer_name CompanyA & Sons Ltd
```

node_name “name”

Name of the node when using a graphical design environment. Example:

```
node_name arm_stream_filter
```

node_logo “file name”

Name of the graphical logo file (file path of the manifest) of the node when using a graphical design environment. Example:

```
node_logo arm_stream_filter.gif
```

node_nb_arcs “in out”

Number of input and output arcs of the node used for data streaming. Example

```
node_nb_arcs 1 1 ; nb arc input, output, default values "1 1"
```

node_mask_library “n”

The graph interpreter offers a short list of optimized DSP/ML and Math functions optimized for the platform using dedicated vector insttructions and coprocessors. Some platform may not incorporate all the libraries, the “node_mask_library” is a bit-field associate to one of the librarry service. This list of service is specially useful when the node is delivered in binary format.

bit 4 for the STDLIB library (string.h, malloc) bit 5 for MATH (trigonometry, random generator, time processing) bit 6 for DSP_ML (filtering, FFT, 2D convolution-8bits) bit 7 for audio codecs bit 8 for image and video codec

Example :

```
node_mask_library  64  ; the node has a dependency to DSP/ML computing
services
```

node_architecture “name”

Create a dependency of the execution of the node to a specific processor architecture. For example when the code is incorporating in-line assembly of a specific architecture.

Example:

```
   node_architecture    armv6m  ; a node made for Cortex-M0
```

node_fpu_used “option” (TBD)

The command creates a dependency on the FPU capabilities Example :

```
   node_fpu_used   0  ; fpu option used (default 0: none, no FPU assembly or
intrinsic)
```

node_version “n”

For information, the version of the node Example :

```
 node_version    101            ; version of the computing node
```

node_complexity_index “n”

For information and debug to set a watchdog timer: the parameters give the maximum amount of cycles for the initialization of the node and the processing of a frame. Example :

```
 node_complexity_index 1e5 1e6 ; maximum number of cycles at initialization
time and execution of a frame
```

node_not_reentrant

node_not_reentrant “n”

Information of reentrancy : the function cannot be called again before it completes its previous execution. Default is “0”, nodes are reentrant. Example :

```
 node_not_reentrant 1 ; one single instance of the node can be scheduled in
the graph
```

node_stream_version “n”

Version of the stream scheduler it is compatible with. Example :

```
  node_stream_version     101
```

node_logo “file name”

File name of the node logo picture (JPG/GIF format) to use in the GUI.

Node memory allocation

A node can ask for **up to 6 memory banks** with tunable fields :

- type (static, working/scratch, static with periodic backup)
- speed (normal, fast, critical fast)
- relocatable (the location can change after the node was notified)
- program / data
- size in bytes

The size can be a simple number of bytes or a computed number coupled to a function of stream format parameters (number of channels, sampling rate, frame size) and a flexible parameter defined in the graph, here. The total memory allocation size in bytes =

```
   A +                              fixed memory allocation in Bytes (default
0)
   size of raw samples of arc(i) x
   ( B x nb_channels of arc(i)      number of channels in arc index i (default
0)
   + C x sampling_rate of arc(j)    sampling rate of arc index j (default 0)
   )
   + D x frame_size of arc(k)       frame size used for the arc index k
```

```
(default 0)
   + parameter from the graph      optional field "node_malloc_add"
```

The first memory block is the node instance, followed by other blocks. This block has the index #0.

node_mem “index”

The command is used to start a memory block declaration with the index in the parameter. Example :

```
` node_mem 0    ; starts the declaration section of memory block #0 `
```

node_mem_alloc “A”

The parameter gives the “A” value of additional memory allocation in Bytes. Example :

```
node_mem_alloc  32    ; add 32 bytes to the current node_mem
```

node_mem_nbchan “B” “i”

Declaration of extra memory in proportion to the number of channel of input arcs (i=1), output arcs (i=2), all arcs (i=3) Example :

```
node_mem_nbchan 44 1  ; add this amount of bytes : 44 x nb of channels of
input arcs
```

node_mem_sampling_rate “C” “j’”

Declaration of extra memory in proportion with the sampling rate of a given arc index. Example :

```
node_mem_sampling_rate 44.0 3  ; add this amount of bytes : 44.0 x
sampling_rate in Hertz of arc 3
```

node_mem_frame_size “D” “k”

Declaration of extra memory in proportion with the frame size of the stream flowing through a specified arc index. Example :

```
node_mem_frame_size 44 3  ; add this amount of bytes : 44 x frame size of arc
3
```

node_mem_alignment “n”

Declaration of the memory Byte alignment Example :

```
node_mem_alignement     4          ; 4 bytes to (default) `
```

node_mem_type “n”

Definition of the dynamic characteristics of the memory block :

0 STATIC : memory content is preserved (default)

1 WORKING : scratch memory content is not preserved between two calls

2 PERIODIC_BACKUP static parameters to reload during a warm reboot

3 PSEUDO_WORKING static only during the uncompleted execution state of the NODE

Example :

```
node_mem_type 3   ; memory block put in a backup memory area when possible
```

node_mem_speed "n"

Declaration of the memory desired for the memory block.

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

Example :

```
node_mem_speed 0   ; relax speed constraint for this block
```

node_mem_relocatable "0/1"

Declares if the pointer to this memory block is relocatable, or assigned a fixed address at reset (default, parameter = '0'). When the memory block is relocatable a command 'STREAM_UPDATE_RELOCATABLE' is used with address changes: void (node) (command, ..); This is done with the associated script of the node (TBD).

Example :

```
node_mem_relocatable    1   ; the address of the block can change
```

node_mem_data0prog1 "0/1"

This command tells if the memory will be used for data or program accesses. Default is '0' for data access. Example : `node_mem_data0prog1  1 ; program memory block`

Configuration of the arcs attached to the node

The arc configuration gives the list of compatible options possible for the node processing. Some options are described as a list, or as a range of values. The syntax is : an index and the list of numbers within brackets "{" and "}". The index gives the default value to consider in the list. Index "1" corresponds to the first element of the list. Index value "0" means "any value". The list can be empty in that case.

Example of an option list between five values, the index is 2 meaning the default value is the second in the list (value = 6). `{ 2  5 6 7 8 9 }` When the index is negative the list is decoded as a "range". A Range is a set of three numbers :

- the first option
- the step to the next possible option
- the last (included) option

The absolute index value selects the default value in this range.

Example of is an option list of values (1, 1.2, 1.4, 1.6, 1.8, .. , 4.2), the index is -3 meaning the default value is the third in the list (value = 1.4).

```
{ -3  1 0.2 4.2 }
```

node_arc "n"

The command starts the declaration of a new arc, followed by its index used when connecting two nodes. Example :

```
node_arc 2    ; start the declaration of a new arc with index 2
```

Implementation comment : all the nodes have at least one arc on the transmit side used to manage the node's locking field.

node_arc_name "name"

Name of the arc used in the GUI. Example:

```
node_arc_name filter_output  ; "filter_output" is the name of the arc
```

node_arc_rx0tx1 "0/1"

Declares the direction of the arc from the node point of view : "0" means a stream is received through this arc, "1" means the arc is used to push a stream of procesed data.

```
node_arc_rx0tx1             0               ; followed by 0:input
1:output, default = 0 and 1
```

node_arc_interleaving "0/1"

Arc data stream interleaving scheme: "0" for no interleaving (independent data frames per channel), "1" for data interleaving at raw-samples levels. Example :

```
node_arc_interleaving 0     data is deinterleaved on this arc
```

node_arc_nb_channels "n"

Number of the channels possible for this arc (default is 1). Example :

```
node_arc_nb_channels {1 1 2}  ; options for the number of channels is
mono or stereo
```

node_arc_units_scale "unit" "scale"

Command used when the node needs the streams to be rescaled to absolute scaled units (See paragraph "Units" of Tables.md).

```
    node_arc_units_scale VRMS  0.15  ; full-scale is equivalent to 0.15 VRMS
```

node_arc_units_scale_multiple “unit” “scale”

Command used when the node needs the streams to be rescaled to absolute scaled units and there are multiple units in sequence (See paragraph “Units” of Tables.md).

```
    node_arc_units_scale_multiple DPS 360 GAUSS 0.002
    ; interleaved format with maximum 360 dps and 0.002 Gauss
```

node_arc_raw_format “f”

Raw samples data format for read/write and arithmetic’s operations. The stream in the “2D domain” are defining other sub-format Example :

```
    node_arc_raw_format {1 17 27} raw format options: STREAM_S16,
STREAM_FP32, default values S16
```

node_arc_frame_length “n”

Frame size options in Bytes. node_arc_frame_length {1 1 2 16} ; options of possible frame_size in number of sample (can mono or multi-channel) Example :

```
    node_arc_frame_length 2    ; start the declaration of a new arc with
index 2
```

node_arc_frame_duration “t”

Duration of the frame in milliseconds. The translation to frame length in Bytes is made during the compilation of the graph from the sampling-rate and the number of channels. A value “0” means “any duration” which is the default. Example :

```
    node_arc_frame_duration {1 10 22.5}  frame of 10ms (default) or 22.5ms
```

node_arc_sampling_rate “fs”

Declaration of the allowed options for the node_arc_sampling_rate in Hertz. Example :

```
    node_arc_sampling_rate {1 16000 44100} ; sampling rate options, 16kHz is
the default value if not specified
```

node_arc_sampling_period_s “T”

Duration of the frame in seconds. The translation to frame length in Bytes is made during the compilation of the graph from the sampling-rate and the number of channels. A value “0” means “any duration” which is the default. Example :

```
    node_arc_sampling_period_s {-2 0.1 0.1 1}  frame sampling going from
100ms to 1000ms, with default 200ms
```

node_arc_sampling_period_day “D”

Duration of the frame in days. The translation to frame length in Bytes is made during the compilation of the graph from the sampling-rate and the number of channels. A value “0” means “any duration” which is the default. Example :

```
    node_arc_sampling_period_day {-2 1 1 30}  frame sampling going from 1 day
to 1 month with steps of 1 day.
```

node_arc_sampling_accuracy “p”

When a node does not need the input data to be rate-accurate, this command allows some rate flexibility without the need for the insertion of a synchronous rate converter. The command parameter is in percent. Example :

```
    node_arc_sampling_accuracy  0.1  ; sampling rate accuracy is 0.1%
```

node_arc_inPlaceProcessing “in out”

Memory optimization with arc buffer overlay. This command tells the “in” arc index is overlaid with the “out” arc index. The default configuration is to allocate different memory for input and output arcs. The arc descriptors are different but the base address of the buffers are identical. Example :

```
    node_arc_inPlaceProcessing  1 2   ; in-place processing can be made
between arc 1 and 2
```

Node design

All the programs can be used with the Graph-scheduler (also called computing “nodes”) as soon as a minimal amount of description is given in a “manifest” and the program can be used through a single entry point with a wrapper using the prototype :

```
void (node) (uint32_t command, stream_handle_t instance, stream_xdmbuffer_t
*data, uint32_t *state);
```

Where “command” tells to reset, run, set parameters, .. “instance” is an opaque access to the static area of the node, “data” is a table of pointers+size pairs of all the arcs used by the node, and “state” returns information of computing completion of the subroutine for the data being shared through the arcs.

During “reset” sequence of the graph the node are initialized. Nothing prevents a node to call the standard library for memory allocations or math computing. But the context of the graph interpreter is Embedded IoT, with extreme optimization for costs and power consumption.

General recommendations

General programming guidelines of Node :

- Nodes must be C callable, or respecting the EABI.
- Nodes are reentrant, or this must be mentioned in the manifest.
- Data are treated as little endian by default.
- Data references are relocatable, there is no “hard-coded” data memory locations.
- All Node code must be fully relocatable: there cannot be hard coded program memory locations.
- Nodes are independent of any particular I/O peripheral, there is no hard coded address.
- Nodes are characterized by their memory, and MIPS requirements when possible (with respect to buffer length to process).
- Nodes must characterize their ROM-ability; i.e., state whether or not they are ROM-able (no self-modifying code unless documented so).
- Run-time object creation should be avoid : memory reservation should be done once during the initialization step.
- Nodes are managing buffers with pointer using physical addresses and shared in the parameters.
- Processors have no MMU : there is no mean of mapping physically non-contiguous segments into a contiguous block.
- Cache coherency is managed in Graph-Interpreter at transitions from one node to the next one.
- Nodes should not use stack allocated buffers as the source or destination of any graph services for memory transfer.
- Static/persistent data, retention data used for warm-boot, scratch data, stack and heap usage will be documented.
- Manifest is detailing the memory allocation section with respect to latency/speed requirements.

Node parameters

A node is using the following prototype

```
void (node) (uint32_t command, void *instance, void *data, uint32_t *state);
```

With following parameters:

Parameter name	Details	Types
command	input parameter	uint32_t
instance	instance	void * casted to the node type
data	input data	casted pointer to struct stream_xdmbuffer { int address;

Parameter name	Details	Types
		int size; }
state	returned state	uint32_t *

Command parameter

Command bit-fields :

Bit-fields	Name	Details
31-24	reserved	
16-23	node tag	different roles depending on the command. With “set parameter” it gives the index of the parameter to update from the data* address (if 0 then all the parameters are prepared in data*)
15-12	preset	the node can define 16 “presets”, preconfigured sets of parameters
11-5	reserved	
4	extended	set to 1 for a reset command with warm boot : static areas need to be initialized except the memory segments assigned to a retention memory in the manifest. When the processor has no retention memory those static areas area cleared by the scheduler.
3-0 (LSB)	command	1: reset 2: set parameter 3: read parameter4: run5: stop6: update the physical address of a relocatable memory segment

Instance

Instance is an opaque memory pointer (void *) to the main static area of the node. The memory alignment requirement is specified in the node manifest.

Data

The multichannel data field is a pointer of arcs’ data. This is pointer to list of structures of two “INTPTR_T” (32bits or 64bits wide depending on the processor architecture). The first INTPTR_T is a pointer to the data, the second tells the number of bytes to process (for an input arc) or the number of bytes available in the buffer (for output arcs).

A node can have **16 arcs**. Each of them can have individual format (number of channels, frame length, interleaving scheme, raw sample type, sampling rate, time-stamps). Arcs can be used for other purpose than data stream, like parameter storage.

Status

Nodes return state is “0” unless the data processing is not finished, then the returned status of “1”.

Node calling sequence

The nodes are first called with the command *reset* followed by *set parameters*, several *run* commands and finally *stop* to release memory. This paragraph details the content of the parameters of the node during "reset", "set parameter" and "run" :

```
void (node) (uint32_t command, void *instance, void *data, uint32_t *state);
```

Reset command

Each Node is define by an index on 10 bits and a "synchronization Byte" with 3-bits defining the architecture it made for (from 1 to 7, "0" means any architecture), 3-Bits defining the processor index within this architecture (from 1 to 7, "0" means any processor), and 2-bits for thread instance ("0" means any thread, "1, 2, 3" respectively for low-latency, normal latency, and background tasks. At reset time processor 1, of architecture 1 is allowed to copy the graph from Flash to RAM and unlock the others.

Then each processor parses the graph looking nodes associated to him, resets it and updates the parameters from graph data. When all the nodes have been set the application is notified and the graph switches to "run" mode. Each graph scheduler instance takes care input and output streams are not blocked : each IOs is associated to a processor. Most of the time a single processor is in charge of all.

The multiprocessor synchronization mechanisms are abstracted outside of the graph interpreter (in the platform abstraction layer), a software-based lock is proposed by default.

The second parameter "instance" is a pointer to the list of memory banks reserved by the scheduler for the node, in the same sequence order of the declarations made in the node manifest. The first element of the list is the instance of the node, followed by the pointers to the data (or program) memory reservations.

The third parameter "data" is used to share the address of function providing computing services.

Set Parameter command @@@

The bit-field "Node Tag" tells which (or all) parameter will be updated.

The third parameter "data" is a pointer to the new parameters.

Run command

The bit-field "Node Tag" tells which (or all) parameter will be updated.

The third parameter "data" is a pointer to the list buffer ("struct stream_xdmbuffer { int address; int size; }") associated to each arc connected to the node.

Test-bench and non-regression test-patterns

Nodes are delivered with a test-bench (code and non-regression database).

Node example @@@

```
typedef struct
{   q15_t coefs[MAX_NB_BIQUAD_Q15*6];
    q15_t state[MAX_NB_BIQUAD_Q15*4];
 } arm_filter_memory;

typedef struct
{   arm_filter_memory *TCM;
} arm_filter_instance;

void arm_stream_filter (int32_t command, void *instance, void *data, uint32_t
*status)
{
    *status = NODE_TASKS_COMPLETED;    /* default return status, unless
processing is not finished */

    switch (RD(command,COMMAND_CMD))
    {
    /* func(command = (STREAM_RESET, COLD, PRESET, TRACEID tag, NB ARCS
IN/OUT)
            instance = memory_results and all memory banks following
            data = address of Stream function

            memresults are followed by 4 words of STREAM_FORMAT_SIZE_W32 of
all the arcs
            memory pointers are in the same order as described in the NODE
manifest

            memresult[0] : instance of the component
            memresult[1] : pointer to the allocated memory (biquad states and
coefs)

            memresult[2] : input arc Word 0 SIZSFTRAW_FMT0 (frame size..)
            memresult[ ] : input arc Word 1 SAMPINGNCHANM1_FMT1
            ..
            memresult[ ] : output arc Word 0 SIZSFTRAW_FMT0
            memresult[ ] : output arc Word 1 SAMPINGNCHANM1_FMT1

            preset (8bits) : number of biquads in cascade, max = 4, from NODE
manifest
            tag (8bits)  : unused
    */
    case STREAM_RESET:
    {
        uint8_t *pt8b, i, n;
        intPtr_t *memreq;
        arm_filter_instance *pinstance;
        uint8_t preset = RD(command, PRESET_CMD);
        uint16_t *pt16dst;
```

```
        /* read memory banks */
        memreq = (intPtr_t *)instance;
        pinstance = (arm_filter_instance *) (*memreq++);        /* main
instance */
        pinstance->TCM = (arm_filter_memory *) (*memreq);       /* second
bank = fast memory */

        /* here reset */
        pt8b = (uint8_t *) (pinstance->TCM->state);
        n = sizeof(pinstance->TCM->state);
        for (i = 0; i < n; i++) { pt8b[i] = 0; }

        /* load presets */
        pt16dst = (uint16_t *)(&(pinstance->TCM->coefs[0]));
        switch (preset)
        {   default:
            case 0:     /* by-pass*/
                pt16dst[0] = 0x7FFF;
                break;
        }
        break;
    }

    /* func(command = bitfield (STREAM_SET_PARAMETER, PRESET, TAG, NB ARCS
IN/OUT)
                TAG of a parameter to set, NODE_ALL_PARAM means "set all the
parameters" in a raw
            *instance,
            data = (one or all)
    */
    case STREAM_SET_PARAMETER:
    {   uint8_t *pt8bsrc, i, numStages;
        uint16_t *pt16src, *pt16dst;
        int8_t postShift;
        arm_filter_instance *pinstance = (arm_filter_instance *) instance;

        pt8bsrc = (uint8_t *) data;
        numStages = (*pt8bsrc++);
        postShift = (*pt8bsrc++);

        pt16src = (uint16_t *)pt8bsrc;
        pt16dst = (uint16_t *)(&(pinstance->TCM->coefs[0]));
        for (i = 0; i < numStages; i++)
        {   /* format:  {b10, 0, b11, b12, a11, a12, b20, 0, b21, b22, a21,
a22, ...} */
            *pt16dst++ = *pt16src++;    // b10
            *pt16dst++ = 0;             // 0
            *pt16dst++ = *pt16src++;    // b11
```

```
            *pt16dst++ = *pt16src++;    // b12
            *pt16dst++ = *pt16src++;    // a11
            *pt16dst++ = *pt16src++;    // a12
        }

        stream_filter_arm_biquad_cascade_df1_init_q15(
            &(pinstance->TCM->biquad_casd_df1_inst_q15),
            numStages,
            (const q15_t *)&(pinstance->TCM->coefs[0]),
            (q15_t *)&(pinstance->TCM->state),
            postShift);
        break;
    }


    /* func(command = STREAM_RUN, PRESET, TAG, NB ARCS IN/OUT)
           instance,
           data = array of [{*input size} {*output size}]

           data format is given in the node's manifest used during the YML-
>graph translation
           this format can be FMT_INTERLEAVED or FMT_DEINTERLEAVED_1PTR
    */
    case STREAM_RUN:
    {
        arm_filter_instance *pinstance = (arm_filter_instance *) instance;
        intPtr_t nb_data, stream_xdmbuffer_size;
        stream_xdmbuffer_t *pt_pt;
        int16_t *inBuf, *outBuf;


        pt_pt = data;   inBuf = (int16_t *)pt_pt->address;
        stream_xdmbuffer_size = pt_pt->size;  /* data amount in the input
buffer */
        pt_pt++;        outBuf = (int16_t *)(pt_pt->address);
        nb_data = stream_xdmbuffer_size / sizeof(int16_t);

        /* data processing here
            ..
        */

        /* update the data consumption/production */
        pt_pt = data;
        *(&(pt_pt->size)) = nb_data * sizeof(SAMP_IN); /* amount of data
consumed */
        pt_pt ++;
        *(&(pt_pt->size)) = 1 * sizeof(SAMP_OUT);   /* amount of data
produced */
        break;
```

```
    }

    case STREAM_STOP:
    case STREAM_READ_PARAMETER:
    case STREAM_UPDATE_RELOCATABLE:
    default : break;
}
```

Conformance checks

Purpose: create an automatic process to incorporate new NODE in a large repository and have a scalable mean to check conformance:

- verification of the conformance to the APIs
- injection of typical and non-typical data aligned with NODE description
- check of outbound parameter behavior
- check of stack consumption and memory leakage.

Services provided to the nodes

The "service" function has the following prototype

```
typedef void    (services) (uint32_t service_command, uint8_t *ptr1, uint8_t *ptr2, uint8_t *ptr3, uint32_t n);
```

Service command bit-fields :

Bit-fields	Name	Details
31-28	control	set/init/run w/wo wait completion, in case of coprocessor usag
27-24	options	compute accuracy, in-place processing, frame size
23-4	function	Operation/function within the Group
3-0 (LSB)	Group	index to the groups of services : SERV_INTERNAL 1 SERV_SCRIPT 2 SERV_CONVERSION 3 : raw data format conversion (fp32 to int16, etc..)SERV_STDLIB 4 : extract of string and stdlib.h (atof, memset, strstr, malloc..)SERV_MATH 5 : extract of math.h (srand, sin, tan, sqrt, log..)SERV_DSP_ML 6 : filtering, spectrum fixed point and integerSERV_DEEPL 7 : fully-connected and convolutional network SERV_MM_AUDIO 8 : audio codecs (TBD)SERV_MM_IMAGE 9 : image processing (TBD)

TODO : application_callbacks (or scripts)

Graph design

The Graph-Interpreter is scheduling a linked-list of computing nodes interconnected with arcs. Nodes descriptors tell which processor can execute the code, which arc it is connected to. The arcs descriptors tell the base address of the buffers, read/write indexes, debug/trace information to log and a flag to tell the consumer node to wrap data to the base addresses. The buffers base address are portable using 6-bits "offset" and 22-bits "index". The offset is translated by each graph interpreter instance of each processor in a physical address given in the Platform-Manifest.

The graph is placed in a shared memory for all processors, there is no message passing scheme, the Graph-Interpreter scheduler's instances and doing the same estimations in parallel, deciding which node needs to be executed in priority.

A graph text has several sections :

- **Control of the scheduler** : debug option, location of the graph in memory
- **File paths** : to easily incorporate sections of data "included" with files
- **Formats** : most of the arcs are using the same frame length and sampling rate, to avoid repeating the same information the formats are grouped in a table and referenced by indexes
- The **IOs or boundaries of the graph** : the IOs are a kind of arcs producing or consuming a stream of data
- The **scripts** are byte-code interpreted programs used for simple operations like setting parameters, sharing debug information, calling "callbacks" predefined in the application.
- The list of **nodes** ("linked-list" of nodes), without their connexions with other nodes. This section defines also the boot parameters, the memory mapping
- The list of **arcs**, their relations with two nodes and the minimal type of debug activity on data moves

Binary format of the graph

Graph section name	Description
—— next sections can either be in RAM or Flash	
Header (7 words)	The header tells where the graph will be in RAM the size of the following sections, the percentage of memory consumed in the memory banks
IO description and links to the device-driver abstraction (4 words/IO)	Each IO descriptor tell if data will be copied in the arc buffers or if the arc descriptor will be set to point directly to the data.
Scripts byte code and parameters	Scripts are made to update parameters, interface with the application's callbacks, implement simple state-machines, interface with the IOs of the graph
List of Nodes instance and	This section is the translation of the node manifests with

Graph section name	Description
their parameters to use at reset time	additional information from the graph : memory mapping of the node data banks and parameters (preset and specific paremeters)
—— graph sections in RAM area starts here	
List of flags telling if data requests are on-going on the IOs (1 byte/IO)	The flags “on-going” are set by the scheduler and reset upon data transfer completion
List of debug/trace registers used by arcs (2 words/debug register)	Basic programmable data stream analysis (time of last access, average values, estimated data rate)
List of Formats, max 256, 4 words/format	Frame length, number of channels, interleaving scheme, specific data of the domain
List of arc descriptors (5 words/arc)	Base address in the portable format (6bits offset 22bits index in words), read/write indexes with Byte accuracy. The descriptor has an “extension” factor to scale all parameter up to 64GB addressing space.

Example of graph

The graph in text format :

```
;--------------------------------------------------------------------------
;   Stream-based processing using a graph interpreter :
;
;       - The ADC detection is used to toggle a GPIO
;
;   +----------+     +--------+      +--------+     +--------+
;   | ADC      +-----> filter +------> detect +-----> GPIO   |
;   +----------+     +--------+      +--------+     +--------+
;
;-------------------------------------------------------------------
format_index            0
format_frame_length     8
format_index            1
format_frame_length     16
;-------------------------------------------------------------------
stream_io               0                   ; IO0
stream_io_hwid          1                   ; io_platform_data_in_1.txt
stream_io               1                   ; IO1
stream_io_hwid          9                   ; io_platform_data_out_0.txt
;-------------------------------------------------------------------
node arm_stream_filter  0                   ; first node
    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;  1                           ; postShift
        5 s16; 681   422   681 23853 -15161 ; band-pass 1450..1900/16kHz
        5 s16; 681 -1342   681 26261 -15331 ;
    end
;--------------------------------------------------------------------
node sigp_stream_detector 0                    ; second node
    node_preset         3                      ; detector preset
;--------------------------------------------------------------------
;  arc connexions between IOs and node and between nodes

arc_input   0 1 0 arm_stream_filter     0 0 0
arc_output  1 1 1 sigp_stream_detector  0 1 1

; arc going from the filter to the detector
arc arm_stream_filter 0 1 0 sigp_stream_detector 0 0 1
arc_jitter_ctrl  1.5                    ; increase the buffer size

end
```

The compiled result which will be the input file of the interpreter:

```
//--------------------------------------
//  DATE Thu Sep 19 19:49:37 2024
//  AUTOMATICALLY GENERATED CODES
//  DO NOT MODIFY !
//--------------------------------------
0x0000003C, // ------- Graph size = Flash=36[W]+RAM24[W]  +Buffers=48[B]
12[W]
0x00000000, // 000 000 [0] Destination in RAM 0, and RAM split 0
0x00000042, // 004 001 [1] Number of IOs 2, Formats 2, Scripts 0
0x00000015, // 008 002 LinkedList size = 21, ongoing IO bytes, Arc debug
table size 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 ...
0x00083000, // 01C 007 IO(graph0) 1 arc 0 set0copy1=1 rx0tx1=0 servant1 1
shared 0 domain 0
0x00000000, // 020 008 IO(settings 0, fmtProd 0 (L=8) fmtCons 0 (L=8)
0x00000000, // 024 009
0x00000000, // 028 00A
...
0x00000000, // 0A0 028           domain-dependent
0x00000010, // 0A4 029 Format  1 frameSize 16
0x00004400, // 0A8 02A           nchan 1 raw 17
0x00000000, // 0AC 02B           domain-dependent
0x00000000, // 0B0 02C           domain-dependent
0x0000003C, // 0B4 02D IO-ARC descriptor(0) Base 3Ch (Fh words) fmtProd_0
frameL 8.0
```

```
0x00000008, // 0B8 02E     Size 8h[B] fmtCons_0 FrameL 8.0 jitterScaling 1.0
0x00000000, // 0BC 02F
0x00000000, // 0C0 030
0x00000000, // 0C4 031     fmtCons 0 fmtProd 0 dbgreg 0 dbgcmd 0
0x0000003E, // 0C8 032 IO-ARC descriptor(1) Base 3Eh (Fh words) fmtProd_1
frameL 16.0
0x00000010, // 0CC 033     Size 10h[B] fmtCons_1 FrameL 16.0 jitterScaling
1.0
0x00000000, // 0D0 034
0x00000000, // 0D4 035
0x00000101, // 0D8 036     fmtCons 1 fmtProd 1 dbgreg 0 dbgcmd 0
0x00000042, // 0DC 037 ARC descriptor(2) Base 42h (10h words) fmtProd_0
frameL 8.0
0x00000018, // 0E0 038     Size 18h[B] fmtCons_1 FrameL 16.0 jitterScaling
1.5
0x00000000, // 0E4 039
0x00000000, // 0E8 03A
0x00000100, // 0EC 03B     fmtCons 1 fmtProd 0 dbgreg 0 dbgcmd 0
```

Graph: control of the scheduler

The first words of the binary graph give the portion of the graph to move to RAM. To have addresses portability of addresses between processors, the graph interpreter is managing a list of "memory-offsets". Every physical address is computed from a 28 bit-field structure made of : 6 bits used to select maximum 64 memory-offsets (or memory bank). And a 22bits field used as an index in this memory bank. The function "platform_init_stream_instance()" initializes the interpreter memory-offset table.

graph_locations "x"

There are seven parameters corresponding to the seven graph memory sections (below). Each is associated with a code "x" : -1 means "this section of the graph stays and is used from here, as-is", otherwise it means "memory bank ID 'x' will be used for this information segment of the graph and it will be moved to before execution".

```
GRAPH_PIO_HW        0
GRAPH_PIO_GRAPH     1
GRAPH_SCRIPTS       2
GRAPH_LINKED_LIST   3
GRAPH_ONGOING       4
GRAPH_FORMATS       5
GRAPH_ARCS          6
```

Example :

```
graph_locations -1 -1 -1 -1 0 0 0 ; 4 segments stay in Flash, others go in
RAMID 0
```

debug_script_fields "x"

The parameter is a bit-field of flags controlling the scheduler loop :

- bit 0 (lsb 1) set means “call the debug/trace script before each node is called”
- bit 1 (2) set means “call the debug script after each node 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 when starting the graph scheduling”
- bit 4 (16) set means “call the debug script when returning from the graph scheduling”
- no bit is set (default) the debug script is not called

Example :

```
debug_script_fields 0   ; no debug script activated
```

scheduler_return “x”

- 1: return to application caller subroutines after each node execution calls
- 2: return to caller once all node of the graph are parsed
- 3: return to caller when all nodes are starving (default 3)

Example :

```
debug_script_fields 0   ; no debug script activated
```

allowed_processors “x”

bit-field of the processors allowed to execute this graph, (default = 1 main processor)

Example :

```
allowed_processors 0x81 ; (10000001) processor ID 1 and 8 can read the graph
```

graph_file_path “index” “path”

Index and its file path, used when including files (sub graphs, parameter files and scripts).

Example :

```
graph_file_path 2  ../nodes/   ; file path index 2 to the folder ../nodes
```

graph_memory_bank “x”

Command used in the context of memory mapping tuning. “x” : index of the memory bank indexes where to map the graph (default 0).

Example :

```
graph_memory_bank 1   ; select of memory bank 1 of the Platform manifest
```

Graph: IO control and stream data formats

There are three data declared in the graph scheduler instance (*arm_stream_instance_t*): A - a pointer to a RAM area giving

- on-going transfer flags
- @@ debug area of arcs

B - a pointer to the list of IOs bit-fields controlling the setting of the IO, the content of which depends on the *Domain*:

- the index of the arc creating the interface between the IO and a node of the graph ("arcID")
- the Rx/Tx direction of the stream, from the point of view of the graph
- the dynamic behavior : data polling initiated by the scheduler or transfer initiated outside of the graph
- flag telling if the data are copied in the arc's buffer or if the arc's descriptor is modified to point directly to the data
- flag telling if the buffer used by the IO interface must be reserved by the graph compiler
- physical Domain of the data (see command "format_domain")
- index to the Abstraction Layer in charge of operating the transfers
- for audio (mixed-signal setting, gains, sampling-rate, ..)

C - a pointer to the "Formats" which are structures of four words giving :

- word 0 : frame size 4MB (Byte accurate)
- word 1 : number of channels (1..32), interleaving scheme, time-stamp, raw format, domain, sub-type, frame size extension (up to 64GB +/-16kB)
- word 2 : sampling rate in [Hz], truncated IEEE FP32 on 24bits : S_E8_M15
- word 3 : specific to each domain (audio and motion channel mapping, image format and border)

format_index "n"

This command starts the declaration of a new format. example `format_index 2 ; all further details are for format index 2` index used to start the declaration of a new format

format_raw_data "n"

The parameter is the raw data code of the table below. Example `format_raw_data 17 ; raw data is "signed integers of 16bits"` The default index is 17 : STREAM_16 (see Annexe "Data Types").

format_frame_length “n”

Frame length in number of bytes of the current format declaration (default :1) Example `format_frame_length 160`

format_nbchan “n”

Number of channels in the stream (default 1) Example `format_nbchan 2 ; stereo format`

format_sampling_rate “f”

Sampling rate in Hertz Example `format_sampling_rate 1e-3 ; 1mHz

format_interleaving “n”

Example `format_interleaving 0` 0 means interleaved raw data, 1 means deinterleaved data by packets of “frame size”

format_time_stamp “n”

Example `format_time_stamp 40 ; time-stamp format TIME16D` time-stamp format :

- 0: no time stamp
- 1: simple counter
- 39: STREAM_TIME16 format q14.2 in seconds, maximum range 4 hours 30mn, 0.25s steps
- 40: STREAM_TIME16D format q1.15 2 seconds, for time differences, step=30us
- 41: STREAM_TIME32 format q30.2 seconds, maximum 34 years , 0.25s steps
- 42: STREAM_TIME32D format q17.15 seconds, maximum 36hours steps 30us for time differences
- 43: STREAM_TIME64 format q32.26 seconds, maximum 140 years +/- 4ns
- 44: STREAM_TIME64MS format u42 in milliseconds, maximum 140 years
- 45: STREAM_TIME64ISO ISO8601 with signed offset, example 2024-05-04T21:12:02+07:00

format_domain “n”

Usage context of this command is for the section “B” of above chapter “IO control and stream data formats”. Example `format_domain 2  ; this format uses specific details of audio out domain`

DOMAIN	CODE	COMMENTS
GENERAL	0	(a)synchronous sensor, electrical, chemical, color, remote data, compressed streams, JSON, SensorThings, application processor
AUDIO_IN	1	microphone, line-in, I2S, PDM RX
AUDIO_OUT	2	line-out, earphone / speaker, PDM TX, I2S,
MOTION	4	accelerometer, combined or not with pressure and gyroscope

DOMAIN	CODE	COMMENTS
		audio_in microphone, line-in, I2S, PDM RX
2D_IN	5	camera sensor audio_out line-out, earphone / speaker, PDM TX, I2S,
2D_OUT	6	display, led matrix, gpio_in generic digital IO

Information specific of domains

Word 3 of "Formats" holds specific information of each domain.

Audio

Audio channel mapping is encoded on 20 bits. For example a stereo channel holding "Back Left" and "Back Right" will be encoded as 0x0030.

Channel name	Name	Bit
Front Left	FL	0
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
Channel 19	C19	18
Channel 20	C20	19

Motion

Motion sensor channel mapping (w/wo the temperature)

Motion sensor data	Code

Motion sensor data	Code
only accelerometer	1
only gyroscope	2
only magnetometer	3
A + G	4
A + M	5
G + M	6
A + G + M	7

2D

Format of the images in pixels: height, width, border.

Graph: interfaces of the graph

stream_io "n"

This command starts a section for the declaration of IO "n". The parameter is the interface index used in the graph. This declaration starts the definition of a new IO Example

```
stream_io 2
```

stream_io_hwid "ID"

The stream_io is using the ID of the physical interface given in platform manifests (default #0) Example

```
stream_io_hwid 2
```

stream_io_format "n"

Parameter: index to the table of formats (default #0) Example

```
stream_io_format 0
```

stream_io_setting "W1 W2 W3"

"IO settings" is a bit-field structure, specific to the IO domain, placed at the beginning of the binary graph, and used during the initialization sequence of the graph. Up to three control words in hexadecimal can be used.

See also IO Controls Bit-fields per domain

Example

```
stream_io_setting 7812440 0 0
```

stream_io_setting_callback “cb” “X”

The function “platform_init_stream_instance()” initializes the interpreter pointers to the callbacks proposed by the platform. Example

```
    stream_io_setting_callback 6 7812440 ; Use callback 6 for the setting of
the
                                         ; current stream_io using parameter
7812440
```

Graph: memory mapping

Split the memory mapping to ease memory overlays between nodes and arcs by defining new memory-offset index (“ID”). Format : ID, new ID to use in the node/arc declarations, byte offset within the original ID, length of the new memory offset.

```
;               original_id  new_id    start   length
memory_mapping      2         100      1024    32700
```

Memory fill

Filling of a word32 pattern after the arc descriptors

```
    mem_fill_pattern 5 3355AAFF   memory fill 5 word32 value 0x3355AAFF
(total 20 Bytes)
```

Graph: subgraphs

(To be defined)

A subgraph is equivalent to program subroutines for graphs. A subgraph can be reused in several places in the graph or in other subgraph. The graph compiler creates references by name mangling from the call hierarchy. A subgraph receives indexes of IO streams and memory bank indexes for tuning the memory map. The caller gives its indexes of the arcs to use in the subgraph, and the memory mapping offset indexes. Example :

```
    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 for fast/slow/working
(identical here)
```

Graph: nodes declarations

Nodes are declared with their name and respective instance index in the graph (or subgraph). The system integrator can set a "preset" (pre-tuned list of parameters described on node's documentation) and node-specific parameters to load at boot-time. The address offset of the nodes is provided as a result of the graph compilation step. Declaration syntax example :

```
    node arm_stream_filter  0  ; first instance of the nore
"arm_stream_filter"
```

node_preset "n"

The system intgrator can select 16 "presets" when using a node, each corresponding to a configuration of the node (see its documentation). The Preset value is with RESET and SET_PARAMETER commands, the default value is 0. Example :

```
    node_preset              1      ; parameter preset used at boot time
```

node_malloc_add "A s"

Adds and extra number of bytes "A" to the "node_mem" segment index "s". Example :

```
    node_malloc_add 12 0  ; add 12 bytes to segment 0
```

node_map_hwblock "m" "o"

This command is used to tune the memory mapping and bypass the speed requirement of the node manifest. It tells to force the memory segment index given in the first parameter to be mapped to the memory offset index of the second parameter. Example :

```
    node_map_hwblock 0 2 ; memory segment 0 is mapped to bank offset 2
```

node_map_swap "m" "o"

This command is used to optimize the memory mapping of small and fast memory segment by swapping, a memory segment content from and other memory offset (usually a slower one). Usage :

```
; forced swap of the node memory segment 1 to hardware memory offset 0
    node_map_swap 1 0
```

In the above both cases the memory segment 1 is copied (next is swapped) from offset memory segment 0 (a dummy arc descriptor is created to access this temporary area) before code execution.

node_trace_id "io"

Selection of the graph IO interface used for sending the debug and trace information. Example :

```
    node_trace_id  0      ; IO port 0 is used to send the trace
```

node_map_proc, node_map_arch, node_map_rtos

The graph can be executed in a multiprocessor and multi tasks platform. Those commands allow the graph interpreter scheduler to skip the nodes not associated to the current processor / architecture and task. The platform can define 7 architectures and 7 processors. When the parameter is not defined (or with value 0) the scheduler interprets it as “any processor” or “any architecture” can execute this node. Several OS threads can interpret the graph at the same time. A parameter “0” means any thread can execute this node, and the value “1” is associated to low-latency tasks, “3” to background tasks.
Examples :

node_memory_isolation “0/1”

Activate (parameter “1”) the processor memory protection unit (on code, private memory allocated segments, and stack) during the execution of this node. Example :

```
   node_memory_isolation 1 ; activation of the memory protection unit (MPU), default 0
```

node_memory_clear “m”

Debug and security feature: Clear the memory bank “m” before and after the execution of the node.
Example :

```
   node_memory_clear 2 ; clear the memory bank 2 as seen in the manifest before and after execution
```

node_script “index”

The indexed script is executed before and after the node execution. The conditional is set on the first call and cleared on the second call. Example :

```
  node_script 12 ; call script #12 associated to this node
```

node_user_key “k64”

The 64bits key is sent to the node during the reset sequence, It is placed after the memory allocation pointers.

The node receives the “node_key” from the scheduler and this “user_key” to decide the features to activate.

Example :

```
  node_user_key 101447945804525706  64 bits key
```

node_parameters “tag”

This command declares the parameters to share with the node during the RESET sequence. If the “tag” parameter is null is tells the following parameters is a full set. Otherwise it is an

index of a subset defined in the node documentation. The following declaration is a list of data terminated with the “end”. Example of a packed structure of 22 bytes of parameters:

```
     node_parameters     0                    TAG = "all parameters"
         1  u8;  2                           Two biquads
         1  u8;  1                           postShift
         5 s16; 681   422   681 23853 -15161 elliptic band-pass
1450..1900/16kHz
         5 s16; 681 -1342   681 26261 -15331
     end
```

Graph Scripts byte codes

Scripts are interpreted byte-codes designed for control and calls to the graph scheduler for node control and parameter settings. Scripts are declared as standard nodes with extra parameters to declare memory size and allowing it to be reused for several scripts. The script-nodes have the transmit arc used to hold the instance memory (registers, stack and heap memory).

The virtual engine has 20 instructions and up to 10 registers.

There are two formats of instructions:

- test and load : [test field] [register to test or to load] [ALU operation] [ALU operands]
- jump and special operations : calls, scatter/gather load, bit-field operations

Test instructions

The result of the test is evaluated by adding a conditional field to any instruction :

```
List of test instructions :

test_equ         test if equal
test_leq         test if less or equal
test_lt          test if lower
test_neq         test if non equal
test_geq         test if great or equal
test_gt          test if greater

test the result :
if_yes ...
if_no ...
```

Arithmetic operations

```
add         addition of two operands
sub         substraction
mul         multiplication
```

```
div         division
or          logical OR,  FP32 operands are pre-converted to "int"
nor         logical NOR
and         logical AND
xor         logical XOR
shr         shift right, sign extension applied on "signed" registers
shl         shift left
set         set a bit
clr         clear a bit
max         compute the maximum of two operands
min         minimum
amax        maximum of absolute values
amin        minmum of absolute values
norm        normalize to MSB and return the amount of shifts
addmod      addition with modulo defined by "base" and "size"
submod      subtraction with modulo
```

The 10 registers of the virtual machine are “r0” .. “r9”. Using “sp0” (or simply “sp”) means an access to the data located at the stack pointer position, “sp1” tells to increment the stack pointer **SP** after a write to the stack and to decrement it after a read. In case several stack accesses are made in the same instruction the update of the stack pointer are made when reading the instructions from right to left.

For example : `sp1 = add sp1 #float 3.14` : the literal constant “3.14” is added to the data on top of the stack and SP is post-decremented after the read (“pop” operation), the result of the addition is saved on the stack (“push”) with SP post-incremented. `sp1 = add sp0 sp1` pops the stack adds the next stack value (without SP decrement) and the result is pushed.

```
r6 = 3                  r6 = 3  (the default litterals type is int32)
r11 = sp                read data from the top of the stack
r6 = add r5 3           r6 = ( r5 + 3 )
sp1 = r6                push the result on stack (SP incremented)
test_eq r6 sub r5 r4    test if r6 == ( r5 - r4 )
if_yes r6 = add r5 3    conditional addition of r5 with 3 saved in r6
```

Literal constants are signed integers by default, if other data types are needed the constant is preceded by “#float” or “uint8”, for example :

```
r3 = 3.14159            load PI in r3
r4 = mul r3 12.0        floating-point multiplication saved in r4
```

Other instructions examples :

```
swap r2 sp1             swap r2 with the top of the stack, pop it
label L1                label declaration
if_not call L1          conditional call
banz L1 r2              decrement r2 and branch if not zero
jump L1 r1              jump to label and push up to 2 registers
call L1 r2 r3           call a subroutine and push 2 registers
```

```
set r4 #uint32            cast r4 as an unsigned integer
set r4 #heap L2           load r4 with an address in the heap RAM
set r4 base L2            set the base address of a circular buffer
set r4 size 12            set the size of a circular buffer
set r0 graph node_name_2  set r0 with the graph's node instance #2
save r4 r5 r0 r2 r11      push 5 registers on the stack
restore r4 r5             pop 2 registers from the stack
delete 4                  remove the last 4 registers from the stack
[ r4  12 ]+ =  r5         scatter load with pre-increment
r3 = [ r4 ]  r0           gather load
r3 | 8 15 | = r2          bit-field load of r2 to the 2nd byte of r3
r3 = r2 | 0 7 |           bit-field extract the LSB of r2 to r3
return                    return from subroutine or script
Syscall 1 r1 r4 r5        system call (below)
```

Graph syntax

```
script 1                  ; script (instance) index
    script_name    TEST1  ; for reference in the GUI
    script_stack      12  ; size of the stack in word64
    script_registers   4  ; only r0..r3 will be used to save memory
    script_mem_shared  1  ; default is private memory (0) or shared (1)

    script_code
    ...
    return                ; return to the graph scheduler
    script_parameters 0   ;
        1  u8 ;  34       ; data section following the code
        label BBB         ; label to the the second byte
        2  u32; 0x33333333 0x444444444 ;
        label CCC
        1  u8 ;  0x55     ; second label address
        1  u32;  0x66666666 ;

    script_heap           ; heap RAM section (arc buffer)
        1  u8 ;  0        ; RAM
        label DDD
        4  u32;  0 0 0 0  ; label DDD points to a byte address
        label EEE
        1  u8 ;  0        ; heap is initialized at node reset
    end
```

System calls

The "Syscall" instruction gives access to nodes (set/read parameters) and arc (read/write data). It allows the access to other system information:

- FIFO content (read/write), filling status and access to the arc debug information (last time-stamp access, average of samples, etc ..)
- Node parameters read and update, with / without a reset of the node
- Basic compute and data move functions

- The call-backs provided by the application (use-case, change the graph IO parameters, debug and trace)

Syscall syntax

Syscall instructions have five parameters

```
syscall index command param1 param2 param3
```

Syscall index	register parameters
1 (access to nodes)	R1: command (set/read parameter)R2: address of the nodeR3: address of dataR4: number of bytes
2 (access to arcs)	R1: command set/read data=8/9R2: arc's IDR3: address of dataR4: number of bytes
3 (callbacks of the application)	R1: application_callback's IDR2: parameter1 (depends on CB)R3: parameter2 (depends on CB)R4: parameter3 (depends on CB)
4 (IO settings)	R1: command set/read parameter=2/3R2: IO's graph indexR3: address of dataR4: number of bytes
5 (debug and trace)	TBD
6 (computation)	TBD
7 (low-level functions)	TBD, peek/poke directly to memory, direct access to IOs (I2C driver, GPIO setting, interrupts generation and settings)
8 (idle controls)	TBD, Share to the application the recommended Idle strategy to apply (small or deep-sleep).
9 (time)	R1: command and time format R2: parameter1 (depends on CB)R3: parameter2 (depends on CB)R4: parameter3 (depends on CB)

GUI design tool

The compiled binary graph can be generated with graphical tool (prototyped in "stream_tools"). The tool creates the compiled binary format and the intermediate text file for later manual tuning.

c

c

Arcs of the graph

The syntax is different for arcs connected to the boundary of the graph, and arcs placed between two nodes. Depending on real-time behaviors (CPU load and jitter, task priorities, speed of data streams) the data can be processed in-place (large input images for examples) or it can be mandatory to copy the data in temporary FIFO before being processed in the graph. The parameter "set0copy1" is set to 0 (default value) for a

processing made “in-place” : the base address the arc FIFO descriptor is modified during the transfer acknowledgment subroutine `arm_graph_interpreter_io_ack()` to point directly to the IO data (no data move). When the parameter is 1 the data is copied in the arc FIFO. The graph compiler will allocate an amount of memory corresponding to a frame length.

Example :

```
; Syntax :
; arc_input   { io / set0copy1 / fmtProd } + { node / inst / arc / fmtCons }
; arc_output  { io / set0copy1 / fmtCons } + { node / inst / arc / fmtProd }
; arc  { node1 / inst / arc / fmtProd } + { node2 / inst / arc / fmtCons }

 arc_input 4 1 0    xxfilter 6 0 8   ; IO-4 sends data to the node xxfilter

; output arc from node xxdetector instance 5 output #1 using format #2
;            to graph IO 7 using set0copy1=0 and format #9
 arc_output 5 1 2   xxdetector 5 1 2

;  arc between nodeAAA instance 1 output #2 using format #0
;      and nodeBBB instance 3 output #4 using format #1
 arc nodeAAA 1 2 0   nodeBBB 3 4 1
```

arc_input

A declaration of a graph input gives the name of the index of the stream which is the “producer” and the node it is connected to (the “consumer”).

- index of the IO (see stream_io)
- 0 or 1 to indicate the data is consumed “in-place” (parameter =0), or will be copied in the buffer associated to the arc (parameter =1). When the data is processed in-place the graph declares an arc descriptor without buffer, the function `void arm_stream_io_ack()` will copy the address of the data in the base address of the arc descriptor
- format ID (see format “n”) used to produce the stream
- name of the consumer node
- Instance index of the node, starting from 0
- arc index of the node (see node_arc “n”)
- format ID (see format “n”) used to by the node consumer of the stream
- optional information to tell this arc is managed with “high quality of service” (HQoS) : the node consuming the stream will treat the corresponding processing with the highest priority whatever the content of the other arc connected to this node. This consumer node will arrange with data interpolations to let the HQoS stream be processed first with the lowest latency.

Example

```
arc_input   1 0 3 arm_stream_filter   4 0 6
; 1 input stream from io 1
; 0 set the pointer to IO buffer without copy
; 3 third format used
; arm_stream_filter receives the data
; 4 fifth instance of the node in the graph
; 0 arc index of the node connected to the stream (node input)
; 6 stream is consumed using the seventh format
```

arc_output

A declaration of a graph output gives the name of the index of the stream which is the "consumer" and the node it is connected to (the "producer").

Example

```
arc_output   1 1 3 arm_stream_filter   4 1 0
; 1 output stream from io 1
; 1 copy the data from the arc buffer
; 3 third format used
; arm_stream_filter produces the data
; 4 fifth instance of the node in the graph
; 1 arc index of the node connected to the stream (node output)
; 0 stream is generated using the first format
```

arc node1 - node2

Declaration of an arc between two nodes.

Example

```
arc arm_stream_filter 4 1 0 sigp_stream_detector 0 0 1   H
; arm_stream_filter produces the data to the sigp_stream_detector
; 4 fifth instance of the node in the graph
; 1 arc index of the node connected to the stream (node output)
; 0 stream is generated using the first format
; sigp_stream_detector consumes the data
; 0 first instance of the node in the graph
; 0 arc index of the node connected to the stream (node input)
; 1 stream is consumed using the second format
; 'H' tells to process the stream with priority
```

arc flow control RD WR

Flow error management with arc descriptor bits bits FLOW_RD_ARCW2 / FLOW_WR_ARCW2, to let an arc stay with 25% .. 75% of data. Process done in "router" node when using HQOS arc and IO master interfaces

The arc is initialized with 50% of null data. The processing is frame-based, there are minimum 3 frames in the buffer.

When a IO-master writes in an arc with FLOW_WR_ARCW2=1 and the arc is full at +75%, the new data is extrapolated and the arc stays at 75% full

```
  buffer full after NewData was push by the IO-master
  Buff  xxxxxxxx|xxxx|xxxx|xxxx|bbbb|aaaa|
          R_ptr                    W_ptr
The previous frame (bbb) is filled (bbb x win_rampDown) + (newData_aaa x
win_rampUp) W_ptr steps back
        xxxxxxxx|xxxx|xxxx|xxxx|bbaa|----|  buffer full
          R_ptr                    W_ptr
```

When a IO-master read from an arc with FLOW_RD_ARCW2=1 and the arc is empty at -25%, the new data is extrapolated and the arc stays at 25% empty

```
  Buff  |bbbb| is read by the IO-master
  buffer hold only ONE frame |aaaa| after the previous read
        |bbbb|aaaa|----|
              R_ptr W_ptr
  The previous frame (bbb) is filled (aaa x win_rampDown) + (bbb x
win_rampUp) and R_ptr steps back
        |aabb|aaaa|----|  bbbb
         R_ptr      W_ptr
```

Activation of the error flow management bits on read and write access:

```
arc_flow_error  1 1 ; read write
```

arc debug

Each arc descriptor can be configured to have an operation (in a list of 32) implemented with result returned in a dedicated memory section of the graph.

CODE	DEBUG OPERATION
0	no operation
1	increment DEBUG_REG_ARCW1 with the number of RAW samples
2	set a 0 in to *DEBUG_REG_ARCW1, 5 MSB gives the bit to clear
3	set a 1 in to *DEBUG_REG_ARCW1, 5 MSB gives the bit to set
4	increment *DEBUG_REG_ARCW1
5	
6	call-back in the application side, data rate estimate in DEBUG_REG_ARCW1
7	second call-back : wake-up processor from DEBUG_REG_ARCW1=[ProcID, command]
8	
9	time_stamp_last_access
10	peak with forgeting factor 1/256 in DEBUG_REG_ARCW1
11	mean with forgeting factor 1/256 in DEBUG_REG_ARCW1

CODE	DEBUG OPERATION
12	min with forgeting factor 1/256 in DEBUG_REG_ARCW1
13	absmin with forgeting factor 1/256 in DEBUG_REG_ARCW1
14	when data is changing the new data is push to another arc DEBUG_REG_ARCW1=[ArcID]
15	automatic rewind read/write

Example :

```
    arc_debug_cmd  1  debug action "ARC_INCREMENT_REG"
    arc_debug_reg  3  index of the 64bits result, default = #0
```

arc_flush

```
    arc_flush  0 ; forced flush of data in MProcessing and shared tasks
```

arc_map_hwblock

```
     arc_map_hwblock   0  map the buffer to a memory offset, default = #0
(VID0)
```

arc_jitter_ctrl

Command used during the compilation step for the FIFO buffer memory allocation with some margin.

```
    arc_jitter_ctrl  1.5  ; factor to apply to the minimum size between the
producer and the consumer, default = 1.0 (no jitter)
```

arc_parameters

Arcs are used to node parameters when the inlined way (with the node declaration) is limited to 256kBytes. The node manifest declares the number of arcs used for large amount of parameters (NN model, video file, etc ..).

```
    arc_parameters  0       ; (parameter arcs) buffer preloading, or arc
descriptor set with script
       7  i8; 2 3 4 5 6 7 8 ; parameters
    include 1 filter_parameters.txt ; path + text file-name using parameter
syntax
    end
```

Common tables

Stream format Words 0,1,2

Words 0, 1 and 2 are common to all domains :

Word	Bits	Comments
0	0..24	frame size in Bytes (including the time-stamp field) + extension
0	25..31	reserved
1	0..4	nb channels-1 [1..32 channels]
1	5	0 for raw data interleaving (for example L/R audio or IMU stream), 1 for a pointer to the first channel, next channel address is computed by adding the frame size divided by the number of channels
1	6..7	time-stamp format of the stream applied to each frame :0: no time-stamp 1: absolute time reference 2: relative time from previous frame 3: simple counter
1	8..9	time-stamp size on 16bits 32/64/64-ISO format
1	10..15	raw data format
1	16..19	domain of operations (see list below)
1	20..21	extension of the size and arc descriptor indexes by a factor 1/64/1024/16k
1	22..26	sub-type (see below) for pixel type and analog formats
2	0..7	reserved
2	8..31	IEEE-754 FP32 truncated to 24bits (S-E8-M15), 0 means “asynchronous”

Stream format Word 3

Word 3 of “Formats” holds specific information of each domain.

Audio stream format

Audio channel mapping is encoded on 20 bits. For example a stereo channel holding “Back Left” and “Back Right” will be encoded as 0x0030.

Channel name	Name	Bit
Front Left	FL	0
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

Channel name	Name	Bit
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
Channel 19	C19	18
Channel 20	C20	19

Motion

Motion sensor channel mapping (w/wo the temperature)

Motion sensor data	Code
only accelerometer	1
only gyroscope	2
only magnetometer	3
A + G	4
A + M	5
G + M	6
A + G + M	7

2D

Format of the images in pixels: height, width, border. The “extension” bit-field of the word -1 allow managing larger images.

2D data	bits range	comments
smallest dimension	0 - 11	the largest dimension is computed with (frame_size - time_stamp_size)/smallest_dimension
image ratio	12 - 14	TBD =0, 1/1 =1, 4/3 =2, 16/9 =3, 3/2=4
image format	15	0 for horizontal, 1 for vertical
image sensor border	17 - 18	0 .. 3 pixels border
interlace mode	2	progressive, interleaved, mixed, alternate
chroma	2	jpeg, mpeg2, dv, none
color space	2	ITU-BT.601, ITU-BT.709, SMPTE 240M
invert pixels	1	for test/debug
brightness	4	display control

2D data	bits range	comments
contrast	4	display control

Stream format Word 4

Domain-specific, TBD

Stream format Word 5

Domain-specific, TBD

Data Types

Raw data types

TYPE	CODE	COMMENTS
STREAM_DATA_ARRAY	0	stream_array : { `0NNN TT 00` } number, type
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	`uu`
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

TYPE	CODE	COMMENTS
STREAM_Q23	22	Sxxxxxxx.xxxxxxxx.xxxxxxxx 24bits 3 bytes per data
STREAM_Q23_	32	SSSSSSSS.Sxxxxxxx.xxxxxxxx.xxxxxxx 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.. FP32
STREAM_CQ15	28	Sxxxxxxx.xxxxxxxx+Sxxxxxxx.xxxxxxxx (I Q)
STREAM_CFP16	29	Seeeeemm.mmmmmmmm+Seeeeemm.. (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.m..+Seee.. (I Q)
STREAM_FP128	36	Seeeeeee.eeeeeeee.mmmmmmm ... quadruple precision
STREAM_CFP64	37	fp64 + fp64 (I Q)
STREAM_FP256	38	Seeeeeee.eeeeeeee.eeeeemm ... octuple precision
STREAM_WGS84	39	<--LAT 32B--><--LONG 32B-->
STREAM_HEXBINARY	40	UTF-8 lower case hexadecimal byte stream
STREAM_BASE64	41	RFC-2045 base64 for xsd:base64Binary XML data
STREAM_STRING8	42	UTF-8 string of char terminated by 0
STREAM_STRING16	43	UTF-16 string of char terminated by 0

Units

NAME	CODE	UNIT	COMMENT
_ANY	0		any
_METER	1	m	meter
_KGRAM	2	kg	kilogram
_GRAM	3	g	gram
_SECOND	4	s	second
_AMPERE	5	A	ampere
_KELVIB	6	K	kelvin
_CANDELA	7	cd	candela
_MOLE	8	mol	mole
_HERTZ	9	Hz	hertz

NAME	CODE	UNIT	COMMENT
_RADIAN	10	rad	radian
_STERADIAN	11	sr	steradian
_NEWTON	12	N	newton
_PASCAL	13	Pa	pascal
_JOULE	14	J	joule
_WATT	15	W	watt
_COULOMB	16	C	coulomb
_VOLT	17	V	volt
_FARAD	18	F	farad
_OHM	19	Ohm	ohm
_SIEMENS	20	S	siemens
_WEBER	21	Wb	weber
_TESLA	22	T	tesla
_HENRY	23	H	henry
_CELSIUSDEG	24	Cel	degrees Celsius
_LUMEN	25	lm	lumen
_LUX	26	lx	lux
_BQ	27	Bq	becquerel
_GRAY	28	Gy	gray
_SIVERT	29	Sv	sievert
_KATAL	30	kat	katal
_SQUAREMETER	31	m2	square meter (area)
_CUBICMETER	32	m3	cubic meter (volume)
_LITER	33	l	liter (volume)
_M_PER_S	34	m/s	meter per second (velocity)
_M_PER_S2	35	m/s2	meter per square second (acceleration)
_M3_PER_S	36	m3/s	cubic meter per second (flow rate)
_L_PER_S	37	l/s	liter per second (flow rate)
_W_PER_M2	38	W/m2	watt per square meter (irradiance)
_CD_PER_M2	39	cd/m2	candela per square meter (luminance)
_BIT	40	bit	bit (information content)
_BIT_PER_S	41	bit/s	bit per second (data rate)

NAME	CODE	UNIT	COMMENT
_LATITUDE	42	lat	degrees latitude[1]
_LONGITUDE	43	lon	degrees longitude[1]
_PH	44	pH	pH value (acidity; logarithmic quantity)
_DB	45	dB	decibel (logarithmic quantity)
_DBW	46	dBW	decibel relative to 1 W (power level)
_BSPL	47	Bspl	bel (sound pressure level; log quantity)
_COUNT	48	count	1 (counter value)
_PER	49	/	1 (ratio e.g., value of a switch;)
_PERCENT	50	%	1 (ratio e.g., value of a switch;)
_PERCENTRH	51	%RH	Percentage (Relative Humidity)
_PERCENTEL	52	%EL	Percentage (remaining battery energy level)
_ENERGYLEVEL	53	EL	seconds (remaining battery energy level)
_1_PER_S	54	1/s	1 per second (event rate)
_1_PER_MIN	55	1/min	1 per minute (event rate, “rpm”)
_BEAT_PER_MIN	56	beat/min	1 per minute (heart rate in beats per minute)
_BEATS	57	beats	1 (Cumulative number of heart beats)
_SIEMPERMETER	58	S/m	Siemens per meter (conductivity)
_BYTE	59	B	Byte (information content)
_VOLTAMPERE	60	VA	volt-ampere (Apparent Power)
_VOLTAMPERESEC	61	VAs	volt-ampere second (Apparent Energy)
_VAREACTIVE	62	var	volt-ampere reactive (Reactive Power)
_VAREACTIVESEC	63	vars	volt-ampere-reactive second (Reactive Energy)
_JOULE_PER_M	64	J/m	joule per meter (Energy per distance)
_KG_PER_M3	65	kg/m3	kg/m3 (mass density, mass concentration)
_DEGREE	66	deg	degree (angle)
_NTU	67	NTU	Nephelometric Turbidity Unit

NAME	CODE	UNIT	COMMENT
—– rfc8798 —–		Secondary Unit (SenML Unit)	Scale and Offset
_MS	68	s millisecond	scale = 1/1000 1ms = 1s x [1/1000]
_MIN	69	s minute	scale = 60
_H	70	s hour	scale = 3600
_MHZ	71	Hz megahertz	scale = 1000000
_KW	72	W kilowatt	scale = 1000
_KVA	73	VA kilovolt-ampere	scale = 1000
_KVAR	74	var kilovar	scale = 1000
_AH	75	C ampere-hour	scale = 3600
_WH	76	J watt-hour	scale = 3600
_KWH	77	J kilowatt-hour	scale = 3600000
_VARH	78	vars var-hour	scale = 3600
_KVARH	79	vars kilovar-hour	scale = 3600000
_KVAH	80	VAs kilovolt-ampere-hour	scale = 3600000
_WH_PER_KM	81	J/m watt-hour per kilometer	scale = 3.6
_KIB	82	B kibibyte	scale = 1024
_GB	83	B gigabyte	scale = 1e9
_MBIT_PER_S	84	bit/s megabit per second	scale = 1000000
_B_PER_S	85	bit/s byteper second	scale = 8
_MB_PER_S	86	bit/s megabyte per second	scale = 8000000
_MV	87	V millivolt	scale = 1/1000
_MA	88	A milliampere	scale = 1/1000
_DBM	89	dBW decibel rel. to 1 milliwatt	scale = 1 Offset = -30 0 dBm = -30 dBW
_UG_PER_M3	90	kg/m3 microgram per cubic meter	scale = 1e-9
_MM_PER_H	91	m/s millimeter per hour	scale = 1/3600000
_M_PER_H	92	m/s meterper hour	scale = 1/3600
_PPM	93	/ partsper million	scale = 1e-6
_PER_100	94	/ percent	scale = 1/100
_PER_1000	95	/ permille	scale = 1/1000
_HPA	96	Pa hectopascal	scale = 100

NAME	CODE	UNIT	COMMENT
_MM	97	m millimeter	scale = 1/1000
_CM	98	m centimeter	scale = 1/100
_KM	99	m kilometer	scale = 1000
_KM_PER_H	100	m/s kilometer per hour	scale = 1/3.6
_GRAVITY	101	m/s2 earth gravity	scale = 9.81 1g = m/s2 x 9.81
_DPS	102	1/s degrees per second	scale = 360 1dps = 1/s x 1/360
_GAUSS	103	Tesla Gauss	scale = 10-4 1G = Tesla x 1/10000
_VRMS	104	Volt Volt rms	scale = 0.707 1Vrms = 1Volt (peak) x 0.707
_MVPGAUSS	105	millivolt Hall effect, mV/Gauss	scale = 1 1mV/Gauss
_DBSPL	106	Bspl versus dB SPL(A)	scale = 1/10

Stream format “domains”

Domain name	Code	Comments
GENERAL	0	(a)synchronous sensor + rescaling, electrical, chemical, color, .. remote data, compressed streams, JSON, SensorThings
AUDIO_IN	1	microphone, line-in, I2S, PDM RX
AUDIO_OUT	2	line-out, earphone / speaker, PDM TX, I2S,
GPIO	3	generic digital IO, programmable timer ticks, control of relay
MOTION	4	accelerometer, combined or not with pressure and gyroscope
2D_IN	5	camera sensor
2D_OUT	6	display, led matrix,
ANALOG_IN	7	analog sensor with aging/sensitivity/THR control, example : light, pressure, proximity, humidity, color, voltage
ANALOG_OUT	8	D/A, position piezzo, PWM converter
USER_INTERFACE_IO	9	button, slider, rotary button, LED, digits, display,
PLATFORM_6	10	platform-specific #6
PLATFORM_5	11	platform-specific #5
PLATFORM_4	12	platform-specific #4
PLATFORM_3	13	platform-specific #3
PLATFORM_2	14	platform-specific #2
PLATFORM_1	15	platform-specific #1

Architectures codes of platform manifest

Architecture codes (https://sourceware.org/binutils/docs/as/ARM-Options.html) armv1, armv2, armv2a, armv2s, armv3, armv3m, armv4, armv4xm, armv4t, armv4txm, armv5, armv5t, armv5txm, armv5te, armv5texp, armv6, armv6j, armv6k, armv6z, armv6kz, armv6-m, armv6s-m, armv7, armv7-a, armv7ve, armv7-r, armv7-m, armv7e-m, armv8-a, armv8.1-a, armv8.2-a, armv8.3-a, armv8-r, armv8.4-a, armv8.5-a, armv8-m.base, armv8-m.main, armv8.1-m.main, armv8.6-a, armv8.7-a, armv8.8-a, armv8.9-a, armv9-a, armv9.1-a, armv9.2-a, armv9.3-a, armv9.4-a, armv9.5-a

List of pre-installed nodes (development)

ID	Name	Comments
1	arm_stream_script	byte-code interpreter index “arm_stream_script_INDEX”
2	arm_stream_router	router, mixer, rate and format converter
3	arm_stream_amplifier	amplifier mute and un-mute with ramp and delay control
4	arm_stream_filter	cascade of filters
5	arm_stream_modulator	signal generator with modulation
6	arm_stream_demodulator	signal demodulator frequency estimator
7	arm_stream_filter2D	filter / rescale / zoom / extract / merge / rotate
8	sigp_stream_detector	signal detection in noise
9	sigp_stream_detector2D	image activity detection
10	sigp_stream_resampler	asynchronous high-quality sample-rate converter
11	sigp_stream_compressor	raw data compression with adaptive prediction
12	sigp_stream_decompressor	raw data decompression
13	bitbank_jpg_encoder	jpeg encoder
14	elm_jpg_decoder	TjpgDec

arm_stream_script

Scripts are nodes interpreted from byte codes declared in the indexed SCRIPTS section of the graph, or inlined in the parameter section of the node “arm_stream_script”. The first one are simple code sequences used as subroutines or called in the “node_script”index”.

The nodes can manage the data RAM location in a shared arc for all script (instance registers+stack parameters) constants are placed after the byte-codes.

The default memory configuration is “shared” meaning the buffers associated with the script are sharing the same memory buffer.

To have individual static memory associated to a script the “script_mem_shared” must be 0.

Special functions activated with Syscall and conditional instructions: - lock : a block of nodes to a processor to have good cache performance, - if-then: a block of nodes based on script decision (FIFO content/debug registers, ..)

- loop : repeat a list of node several time for cache efficiency and small frame size
- Checks if the data it needs is available and returns to the scheduler

```
node arm_stream_script 1  ; script (instance) index
    script_stack      12  ; size of the stack in word64
    script_register    6  ; number of registers in word64
    script_parameter  30  ; size of the parameter/heap in word32
    script_mem_shared  1  ; private memory (0) or shared(1)
    script_mem_map     0  ; mapping to VID #0 (default)

    script_code
        r1 = add r2 3       ; r1 = add r2 3
      label AAA
        set r2 graph sigp_stream_detector_0
        r0 = 0x412              ; r0 = STREAM_SET_PARAMETER(2)
        set r3 param BBB        ; set r3 param BBB
        sp0 = 1                 ; push 1 Byte (threshold size in BBB)
        Syscall 1 r2 r0 r3 sp0  ; Syscall NODE(1) r2(cmd=set_param) r0(set)
r3(data)
        return              ; return
    end

    script_parameters      0
        1  u8 ;  34
        2  u32; 0x33333333 0x444444444
        label BBB
        1  u8 ;  0x55
        1  u32;  0x66666666
    end
```

---

arm_stream_router

Operation

This node receives up to 4 streams (arcs) and generate up to 4 stream, each can be multichannel. The Format of the streams is known with the “reset and”set param” commands to the node.

Input streams are moved, routed and mixed to generate the output streams in a desired stream format. The output stream are isochronous to the other graph streams (they have a known sampling rate), but the input can be asynchronous (each sample have a time-stamp).

The first parameters give the number of arcs, the input arc to use with HQoS (High Quality of Service), or -1. Followed by a list of routing and mixing information. When there is an HQoS arc the amount of data moves is aligned with it, in the time-domain, to all the other arcs (in case of flow issue data is zeroed or interpolated). Otherwise the node checks all the input and output arcs and finds the minimum amount of data, in the time domain, possible for all arcs.

Use-cases

The following use-cases can be combined to create a new use-case:

1. Router, deinterleaving, interleaving, channels recombination: the input arc data is processed deinterleaved, and the output arc is the result of recombination of any input arc. Audio example with two stereo input arcs using 5ms and 10ms frame lengths, recombined to create a stereo stream interleaved output using the left channel from the first arc and the left channel of the second arc.

2. Router and mixer with smoothed gain control: the output arc data can result from the weighted mix of input arcs. The applied gain can be changed on the fly. The slope of the time taken to the desired gain is controlled. Audio example: a mono output arc is computed from the combination of two stereo input arc, by mixing the four input channels with a factor 0.25 applied in the mixer.

3. Router and raw data conversion. The raw formats can be converted to any other format in this list : int16, int32, int64, float16, float32, float64.

4. Router and sampling-rate conversion of isochronous streams (input streams have a determined and independent sampling-rate). Audio example: input streams sampled at 44100Hz is converter to 48000Hz. The sampling-rate information, and all the details of the arc's data format, is shared by the graph scheduler during the reset phase of the nodes.

5. Router and conversion of asynchronous streams using time-stamps to an isochronous stream with a determined sampling-rate. Motion sensor example: an accelerometer is sampled at 200Hz (5ms period) with +/- 1ms jitter sampling time uncertainty. The samples are provided with an accurate time-stamp in float32 format for time differences between samples (or float64 for absolute time reference to Jan 1st 2025). The output samples are delivered resampled at 410Hz with no jitter.

6. Router of data needing a time synchronization at sample or frame level. In this use-case the node waits the input samples are arriving within a time window before delivering an output frame. Example with motor control and the capture of current and voltage on two input arcs: it is important to send a time-synchronized pairs of data. The command node_script "index" is used to call a script checking the arrival of current and voltage with their respective time-stamps (logged in the arc descriptors), the scripts check the arrival of data within a time and release execution of the router when conditions are met.

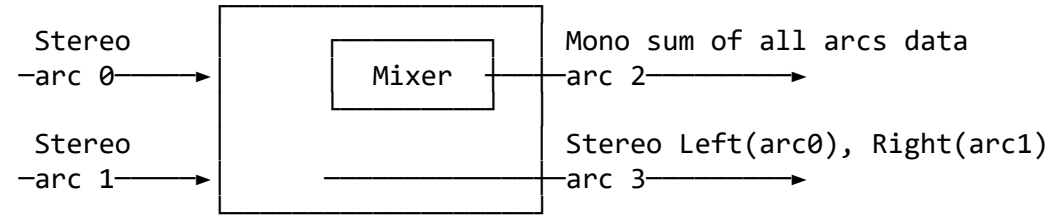
7. Router of streams generated from different threads. The problem is to avoid on multiprocessing devices one channel to be delivered to the final mixer ahead and desynchronized from the others. This problem is solved with an external script like in the use-case 6.

Parameters

The list of routing and mixing information is :

- index of the input arc (<= 4)
- index of the channels (1 Byte to 31 Bytes)
- index of the output destination arc (<=4)
- index of the channels (1 Byte to 31 Bytes)
- mixer gain to apply (fp32) and convergence speed (fp32)

Example with the router with two stereo input arcs and two output arcs. The first output arc is mono and the sum of all the input channels, the second arc is stereo combining the two left channels of the input arcs.




```
; parameters arranged to be accessed with 32bits data
  2  i8; 2 2          nb input/output arcs
  2  i8; -1 -1        no HQoS arc on input and output
  ;
  ;     arcin ichan arcout ichan
  4  i8; 0     0     2     0     ; move arc0-left  to arc2 mono x0.25
  2 f32: 0.25  0.1               ;    gain and convergence speed
  4  i8; 0     1     2     0     ; move arc0-right to arc2 mono x0.25
  2 f32: 0.25  0.1
  4  i8; 1     0     2     0     ; move arc1-left  to arc2 mono x0.25
  2 f32: 0.25  0.1
  4  i8; 1     1     2     0     ; move arc1-right to arc2 mono x0.25
  2 f32: 0.25  0.1
  4  i8; 0     0     3     0     ; move arc0-left  to arc3 left no mixing
  2 f32: 0.25  0.1
  4  i8; 1     1     3     1     ; move arc1-right to arc3 right no mixing
  2 f32: 0.25  0.1
```

Operations :

- when receiving the reset command: compute the time granularity for the processing, check if bypass are possible (identical sampling rate on input and output arcs).

- check all input and output arcs to know which is the amount of data (in the time domain) which can me routed and split in “time granularity” chunks. Clear the mixer buffers.

Loop with “time granularity” increments :

- copy the input arcs data in internal FIFO in fp32 format, deinterleaved, with time-stamps attached to each samples.
- use Lagrange polynomial interpolation to resample the FIFO to the output rate. The interpolator is preceded by a an adaptive low-pass filter removing high-frequency content when the estimated input sampling rate higher than the output rate.

---

arm_stream_amplifier (TBD)

Operation : rescale and control of the amplitude of the input stream with controlled time of ramp-up/ramp-down. The gain control “mute” is used to store the current gain setting, being reloaded with the command “unmute” Option : either the same gain/controls for all channels or list of parameters for each channel

Parameters : new gain/mute/unmute, ramp-up/down slope, delay before starting the slope. Use-cases : Features : adaptive gain control (compressor, expander, AGC) under a script control with energy polling Metadata features : “saturation occurred” “energy” Mixed-Signal glitches : remove the first seconds of an IR sensor until it was self-calibrated (same for audio Class-D)

parameters of amplifier (variable size): TAG_CMD = 1, uint8_t, 1st-order shifter slope time (as stream_mixer, 0..75k samples) TAG_CMD = 2, uint16_t, desired gain FP_8m4e, 0dB=0x0805 TAG_CMD = 3, uint8_t, set/reset mute state TAG_CMD = 4, uint16_t, delay before applying unmute, in samples TAG_CMD = 5, uint16_t, delay before applying mute, in samples

lopes of rising and falling gains, identical to all channels slope coefficient = 0..15 (iir_coef = 1-1/2^coef = 0 .. 0.99) Convergence time to 90% of the target in samples: slope nb of samples to converge 0 0 1 3 2 8 3 17 4 36 5 73 6 146 7 294 8 588 9 1178 10 2357 11 4715 12 9430 13 18862 14 37724 15 75450 convergence in samples = abs(round(1./abs(log10(1-1./2.[1])’))

Operation : applies vq = interp1(x,v,xq) Following https://fr.mathworks.com/help/matlab/ref/interp1.html linear of polynomial interpolation (implementation) Parameters : X,V vectors, size max = 32 points

no preset (‘0’)

---

[1] 0:15

Or used as compressor / expander using long-term estimators instead of sample-based estimator above.

```
node arm_stream_rescaler 0

    parameters     0             ; TAG   "load all parameters"

;               input   output
        2; f32; -1      1
        2; f32;  0      0       ; this table creates the abs(x) conversion
        2; f32;  1      1
    end
end
```

```
node  arm_stream_amplifier 0


    parameters     0             ; TAG   "load all parameters"
        1  i8;  1           load only rising/falling coefficient slope
        1 h16;  805         gain -100dB .. +36dB (+/- 1%)
        1  i8;  0           muted state
        2 i16;  0 0         delay-up/down
    end
end
```

---

arm_stream_filter

Operation : receives one multichannel stream and produces one filtered multichannel stream. Parameters : biquad filters coefficients used in cascade. Implementation is 2 Biquads max. (see www.w3.org/TR/audio-eq-cookbook) Presets: #0 : bypass #1 : offset removal filter #2 : Median filter, 5 points #3 : Low pass filter #4 : High pass filter #5 : Peaking filter #6 : Bandpass filter #7 : Notch filter #8 : Low shelf filter #9 : High shelf filter #10: All pass filter #11: Dithering filter

parameter of filter :

Normalized frequency f0/FS default = 0.25,

Q factor default = 1.414

Default gain = 4 (12dB)

```
node arm_stream_filter 0         node subroutine name + instance ID
    node_preset         1        ; parameter preset used at boot time,
default = #0
```

```
    node_map_hwblock    0  0     ; list of "nb_mem_block" VID indexes of
                                  ; "procmap_manifest_xxxx.txt" where to map
                                  ; the allocated memory
                                  ;  default = #0

    parameters     0              ; TAG   "load all parameters"
        1  u8;  2       Two biquads
        1  i8;  0       postShift
        5 f32; 0.284277f 0.455582f 0.284277f 0.780535f -0.340176f
        5 f32; 0.284277f 0.175059f 0.284277f 0.284669f -0.811514f
        ; or  _include    1   arm_stream_filter_parameters_x.txt      (path +
file-name)
    end
end
```

---

arm_stream_modulator (TBD)

Operation : sine, noise, square, saw tooth with amplitude or frequency modulation use-case : ring modulator, sweep generation with a cascade of a ramp generator and a frequency modulator

see https://www.pjrc.com/teensy/gui/index.html?info=AudioSynthWaveform

```
u8  wave type 1=cosine 2=square 3=white noise 4=pink noise
    5=sawtooth 6=triangle 7=pulse
    8=prerecorded pattern playback from arc
    9=sigma-delta with OSR control for audio on PWM ports or 8b DAC
    10=PWM 11=ramp 12=step
u8  modulation type, 0:amplitude, 1:frequency, 2:FSK
u8  modulation, 0:none 1=from arc bit stream

f32 modulation amplitude
f32 offset
f32 wave frequency [Hz]
f32 starting phase,[-pi .. +pi]
f32 modulation y=ax+b, x=input data, index (a) and offset (b)
f32 modulation frequency [Hz] separating two data bits/samples from the arc

node arm_stream_modulator (i)

    parameters     0              ; TAG   "load all parameters"

        1  u8;  1       sinewave
        2 h16;  FFFF 0  full-scale, no offset
        1 f32;  1200    1200Hz
        1 s16;  0       initial phase
        2  u8;  1 1     frequency modulation from bit-stream
        2 h16;  8000 0  full amplitude modulation with sign inversion of the
```

```
bit-stream
        1 f32;  300     300Hz modulation => (900Hz .. 1500Hz modulation)
    end
end
```

---

arm_stream_demodulator (TBD)

Operation : decode a bit-stream from analog data. Use-case: IR decoder, CAN/UART on SPI/I2S audio. Parameters : clock and parity setting or let the algorithm discover the frame setting after some time. https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

presets control : #1 .. 10: provision for demodulators

Metadata information can be extracted with the command “parameter-read”: TAG_CMD = 1 read the signal amplitude TAG_CMD = 2 read the signal to noise ratio

```
node
    arm_stream_demodulator (i)
    parameters     0             ; TAG   "load all parameters"

        2  i8; 2 2          nb input/output arcs
        4 i16; 0 0 2 0      move arc0,chan0, to arc2,chan0
    end
end
```

---

arm_stream_filter2D (TBD)

Filter, rescale/zoom/extract, rotate, exposure compensation. Channel mixer : insert a portion of the processed image in a larger frame buffer.

Operation : 2D filters Parameters : spatial and temporal filtering, decimation, distortion, color mapping/log-effect

presets: #1 : bypass

parameter of filter :

```
node arm_stream_filter2D   (i)

    TBD
end
```

---

sigp_stream_detector

Operation : provides a boolean output stream from the detection of a rising edge above a tunable signal to noise ratio. A tunable delay allows to maintain the boolean value for a minimum amount of time Use-case example 1: debouncing analog input and LED / user-interface. Use-case example 2: IMU and voice activity detection (VAD) Parameters : time-constant to gate the output, sensitivity of the use-case

presets control #1 : no HPF pre-filtering, fast and high sensitivity detection (button debouncing) #2 : VAD with HPF pre-filtering, time constants tuned for ~10kHz #3 : VAD with HPF pre-filtering, time constants tuned for ~44.1kHz #4 : IMU detector : HPF, slow reaction time constants #5 : IMU detector : HPF, fast reaction time constants

Metadata information can be extracted with the command "TAG_CMD" from parameter-read: 0 read the floor noise level 1 read the current signal peak 2 read the signal to noise ratio

```
node arm_stream_detector 0                node name  + instance ID
    preset              1               parameter preset used at boot time,
default = #0
end
```

---

sigp_stream_detector2D (TBD)

Motion and pattern detector (lines)

Operation : detection of movement(s) and computation of the movement map Parameters : sensitivity, floor-noise smoothing factors Metadata : decimated map of movement detection

```
node arm_stream_detector2D (i)

TBD

end
```

---

sigp_stream_resampler (TBD)

Operation : high quality conversion of multichannel input data rate to the rate of the output arcs

- asynchronous rate conversion within +/- 1% adjustment

SSRC synchronous rate converter, FS in/out are exchanged during STREAM_RESET ASRC asynchronous rate converter using time-stamps (in) to synchronous FS (out) pre-LP-filtering tuned from Fout/Fin ratio + Lagrange polynomial interpolator

drift compensation managed with STREAM_SET_PARAMETER command: TAG_CMD = 0 bypass TAG_CMD = 1 rate conversion

The script associated to the node is used to read the in/out arcs filling state to tune the drift control

```
node arm_stream_resampler (i)

    parameters     0             ; TAG   "load all parameters"

        2  i8; 2 2          nb input/output arcs
        4 i16; 0 0 2 0      move arc0,chan0, to arc2,chan0
    end
end
```

---

sigp_stream_compressor (TBD)

Operation : wave compression using IMADPCM(4bits/sample) Parameters : coding scheme

presets (provision codes):

- 1 : coder IMADPCM
- 2 : coder LPC
- 3 :
- 4 : coder CVSD for BT speech
- 5 : coder SBC
- 6 : coder MP3

```
node
    arm_stream_compressor 0

    parameters     0             ; TAG   "load all parameters"
        4; i32; 0 0 0 0     provision for extra parameters in other codecs
    end
end
```

---

sigp_stream_decompressor (TBD)

Operation : decompression of encoded data Parameters : coding scheme and a block of 16 parameter bytes for codecs, VAD threshold and silence frame format (w/wo time-stamps)

dynamic parameters : pause, stop, fast-forward x2 and x4.

```
WARNING : if the output format can change (mono/stereo, sampling-rate, ..)
    the variation is detected by the node and reported to the scheduler with
    "STREAM_SERVICE_INTERNAL_FORMAT_UPDATE", the "uint32_t *all_formats" must
be
```

```
    mapped in a RAM for dynamic updates with
"COPY_CONF_GRAPH0_COPY_ALL_IN_RAM"

Example of data to share with the application
    outputFormat: AndroidOutputFormat.MPEG_4,
    audioEncoder: AndroidAudioEncoder.AAC,
    sampleRate: 44100,
    numberOfChannels: 2,
    bitRate: 128000,
```

presets provision

- 1 : decoder IMADPCM
- 2 : decoder LPC
- 3 : MIDI player
- 4 : decoder CVSD for BT speech
- 5 : decoder SBC
- 6 : decoder MP3

```
node arm_stream_decompressor 0

    parameters     0             ; TAG   "load all parameters"
        4; i32; 0 0 0 0     provision for extra parameters in other codecs
    end
end
```

---

bitbank_jpg_encoder

From “bitbank”

https://github.com/google/jpegli/tree/main

https://opensource.googleblog.com/2024/04/introducing-jpegli-new-jpeg-coding-library.html

eml_tjpg_decoder

From “EML”

Use-case : images decompression, pattern generation.