| Table of content |
| --- |
| [Graph Interpreter Memory](#Graph-Interpreter-instance) |
| [Manifest Files](#Top-Manifest) |
| [Processor Manifest](#Processor-Manifest) |
| [Interfaces Manifests](#IO-Manifest) |
| [Nodes Manifests](#Node-manifest) |
| [Design of Nodes](#Node-design) |
| [Designing a graph](#Graph-design) |
| [Formats and Domains](#Common-tables) |
| [Common Nodes](#Xb457fef2ef2f40fc3fd97fb38aa24663007d594) |

# 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.

1. **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.

1. **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.

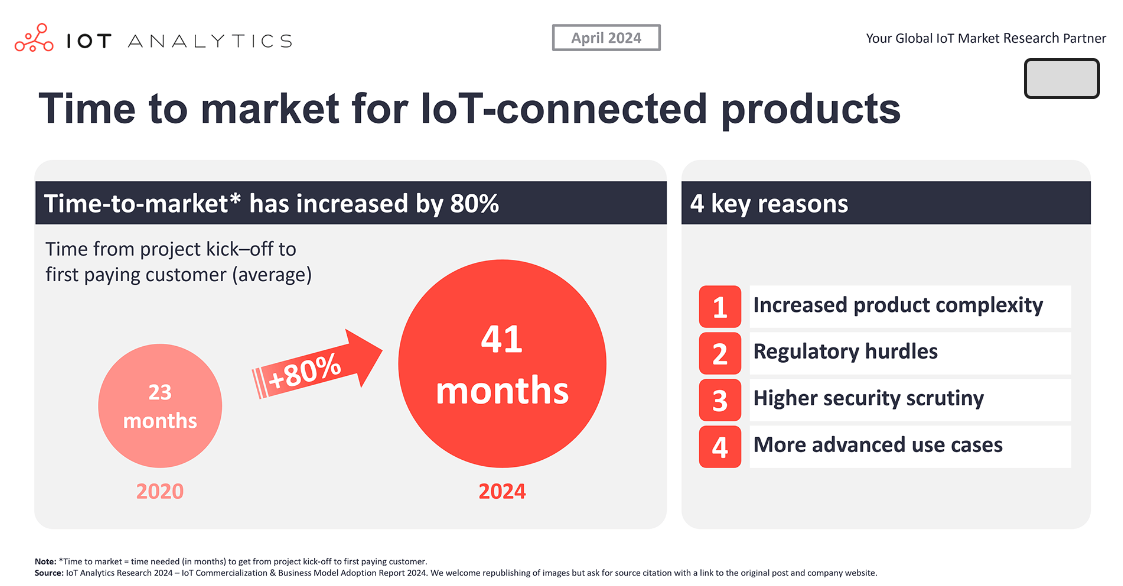
## Why

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](#Units), [RFC8428](https://datatracker.ietf.org/doc/html/rfc8428) and [RFC8798](https://datatracker.ietf.org/doc/html/rfc8798))

Computing nodes and platforms have to explain in “[Manifests](#Node-manifest)” 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](#Graph-design). 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](#Top-Manifest) and [IO manifest](#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](#Node-manifest) 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](#Node-design)).
2. The graph is either written in a text format (syntax example [here](#Example-of-graph)) or is generated from a graphical tool (proof of concept picture of the GUI [here](#GUI-design-tool)).
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:**

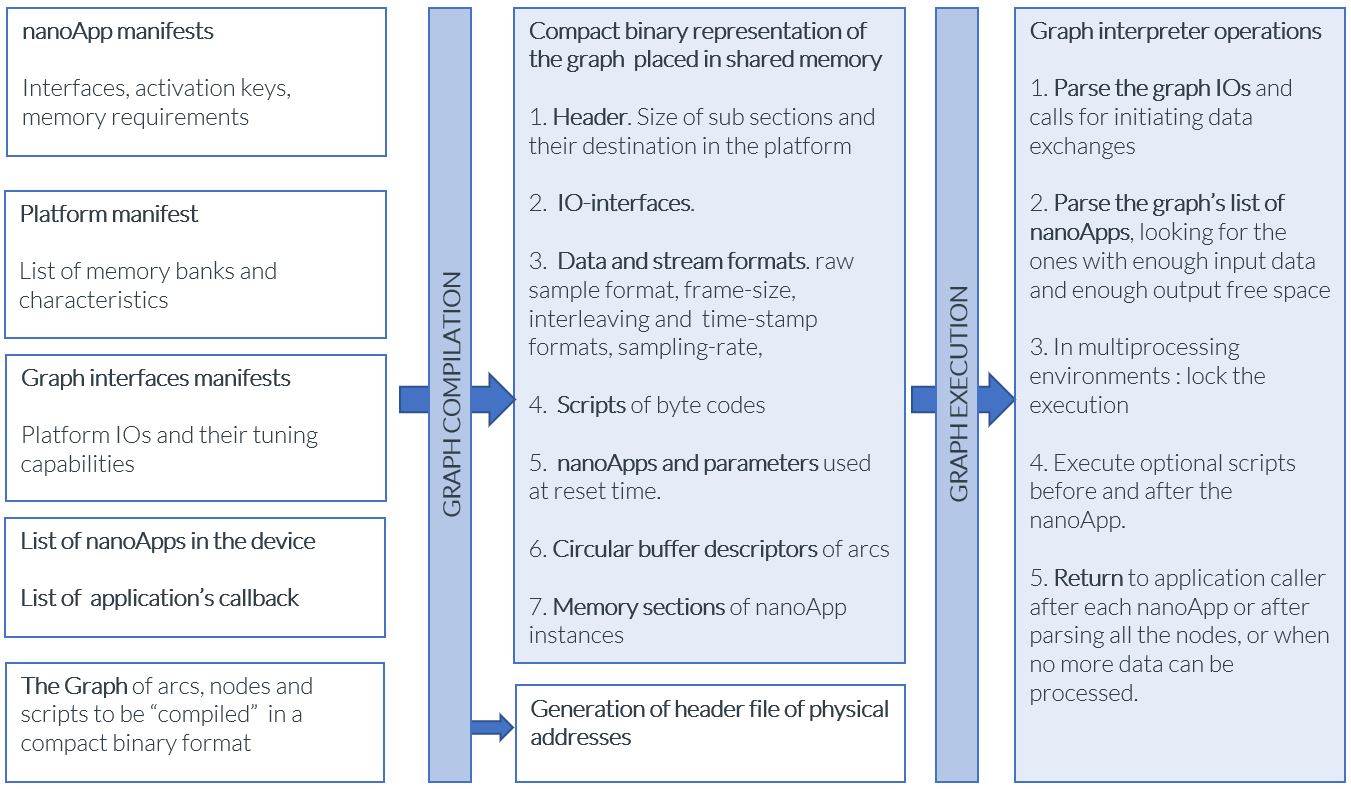
1. **Share the physical memory map base addresses**. The graph is using indexes to the base addresses of 63 different *memory banks*: for example shared external memory, fast shared internal, fast private per processor (TCM), and indexed the same way for multiprocessing without MMU. The AL shares the 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](#Graph-Scripts-byte-codes) 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\_graph\_interpreter\_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](#Node-parameters) 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.

# 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”. |
| graph | A pointer to the compiled binary version of the graph remapped in RAM. The original binary graph is usually in a flash memory area. The AL function (platform\_init\_stream\_instance()) moves it in RAM starting from the first byte of the above “long\_offset” index 0. When the device has a small internal RAM the graph can be split with a portion staying in Flash (this graph pointer is pointing to) and a portion RAM placed at long\_offset[0] address. |
| linked\_list | pointer to the linked-list of nodes of the graph |
| platform\_io | table of functions ([IO\_AL\_idx](#Top-Manifest)) 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. |
| al\_services | pointer to the function proposing services (time, script, stdlib, compute) |
| iomask | bit-field of the allowed IOs this interpreter instance can use |
| scheduler\_control | execution options of the scheduler (return to the application after each node execution, after a full graph parsing, when there is no data to process) |
| 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

# Top Manifest

The “Top” platform manifest has four sections :

* the list of file paths to ease readability
* the manifest file describing the processing architecture (see next paragraph)
* the list of available data stream ready to be connected to the graph : this list corresponds to manifest files of the data format options, and gives an index (IO\_AL\_idx) to the function to call to read/write data and update the stream configuration (when this is possible).
* 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  
; processors manifests (memory and architecture)  
; IO manifests to use for stream processing  
; list of the nodes installed in the platform and their affinities with processors  
; ----------------------------------------------------------------------------------------------  
; list of paths for the included files  
  
 6 three file paths  
 ../../stream\_platform/ "" path index 0 is local  
 ../../stream\_platform/computer/manifest/ "" path index 1 : platform IO manifests  
 ../../stream\_nodes/arm/ "" path index 2 : arm nodes  
 ../../stream\_nodes/signal-processingFR/ "" path index 3 : signal-processingFR nodes  
 ../../stream\_nodes/bitbank/ "" path index 4 : bitbank nodes  
 ../../stream\_nodes/elm-lang/ "" path index 5 : elm-lang nodes  
  
; ----------------------------------------------------------------------------------------------  
; PLATFORM DIGITAL, MIXED-SIGNAL AND IO MANIFESTS - max 32 IOs => iomask  
  
 1 procmap\_manifest\_computer.txt path index + file name  
  
; path: path ID   
; Manifest manifests file   
; FW IO 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.  
  
10 number of IO streams available aligned with struct platform\_io\_control platform\_io[] and platform\_computer.h   
  
;Path Manifest IO\_AL\_idx ProcCtrl clock-domain Comments   
1 io\_platform\_data\_in\_0.txt 0 1 0 application processor   
1 io\_platform\_data\_in\_1.txt 1 1 0 application processor   
1 io\_platform\_analog\_sensor\_0.txt 2 1 0 ADC   
1 io\_platform\_motion\_in\_0.txt 3 1 0 accelero=gyro   
1 io\_platform\_audio\_in\_0.txt 4 1 0 microphone   
1 io\_platform\_2d\_in\_0.txt 5 1 0 camera   
1 io\_platform\_line\_out\_0.txt 6 1 0 audio out stereo   
1 io\_platform\_gpio\_out\_0.txt 7 1 0 GPIO/LED   
1 io\_platform\_gpio\_out\_1.txt 8 1 0 GPIO/PWM   
1 io\_platform\_data\_out\_0.txt 9 1 0 application processor   
  
; ------------------------------------------------------------------------------------------------------------  
; PATH + SOFTWARE COMPONENTS MANIFESTS   
   
2 node\_manifest\_none.txt   
2 script/node\_manifest\_script.txt   
2 format\_converter/node\_manifest\_format\_converter.txt   
2 router/node\_manifest\_router.txt   
3 converter/node\_manifest\_converter.txt   
2 amplifier/node\_manifest\_amplifier.txt   
2 mixer/node\_manifest\_mixer.txt   
2 filter/node\_manifest\_filter.txt   
3 detector/node\_manifest\_detector.txt   
2 rescaler/node\_manifest\_rescaler.txt   
3 compressor/node\_manifest\_compressor.txt   
3 decompressor/node\_manifest\_decompressor.txt   
2 modulator/node\_manifest\_modulator.txt   
2 demodulator/node\_manifest\_demodulator.txt   
3 resampler/node\_manifest\_resampler.txt   
2 qos/node\_manifest\_qos.txt   
2 split/node\_manifest\_split.txt   
3 detector2D/node\_manifest\_detector2D.txt   
2 filter2D/node\_manifest\_filter2D.txt   
2 analysis/node\_manifest\_analysis.txt   
4 JPEGENC/node\_manifest\_bitbank\_JPEGENC.txt   
5 TJpgDec/node\_manifest\_TjpgDec.txt   
  
; ------------------------------------------------------------------------------------------------------------

# Processor Manifest

The processor manifest gives the minimum guaranteed amount of memory per memory bank to help the graph compilation process to map the working/static memory requirements of each node.

; -------------------------------------------------------------------------  
; Processor and memory configuration + default memory mapping   
; -------------------------------------------------------------------------  
  
1 1 9 4 number of architectures, processors, memory banks, offsets  
  
;memory banks:  
;- ID base offset ID reference above  
;- VID virtual ID used in the graph for manual mapping,   
;- S 0=any/1=normal/2=fast/3=critical-Fast,   
;- W static0/working1/retention2  
;- P shared0/private1,  
;- H DMAmemHW1  
;- D Data0/Prog1/Both2   
;- Size minimum sizes guaranteed per bank starting from @[ID]+offset  
;- Offset maximum offset from the base offset ID,  
;  
; the graph uses the "node\_map\_hwblock" command for manual memory mapping   
  
ID VID S W P H D Size offset from offsetID   
0 0 1 0 0 0 0 95526 10 VID0=DEFAULT flat memory bank,   
0 1 0 0 0 0 0 65526 10 SRAM0 static, hand tuned memory banks  
0 2 0 0 0 0 0 30000 65536 SRAM1 static  
0 3 0 1 0 0 0 15000 95536 SRAM1 working at application level  
0 4 0 1 0 0 0 256000 262144 DDR working at application level  
2 5 3 1 1 0 0 1024 262144 DTCM Private memory of proc 1 (>0)  
1 10 0 2 0 0 0 1024 524288 Retention memory  
3 20 0 0 0 0 0 200000 10 Data in Flash  
2 8 3 1 1 0 1 16384 0 ITCM Private memory of proc 1   
  
; offsets used by all processors and addresses seen from the main processor  
0 h20000000 image of platform\_specific\_long\_offset  
1 h28000000 in stream\_al/platform\_XXXXX.c  
2 h2C000000 TCM Private memory of processor 1   
3 h08000000 Internal Flash   
  
;---------------------------------------------------------------------  
; loop on all architectures  
 4 1 architecture code (4=armv8), 1 processor  
 ; all processors for this architecture (processor IDs >0)  
 ;------------------------------------------------------------  
 1 1 15 processor ID,   
 ; boolean "main processor" allowed to boot the graphs  
 ; Bit-field computation firmware extensions,

When memory and speed optimization is not necessary, the system integrator can declare a simplified single continuous block of memory with average speed performance, for example :

; -------------------------------------------------------------------------  
; Simplified memory plan  
; -------------------------------------------------------------------------  
1 1 1 1 number of architectures, processors, memory banks, offsets  
  
;ID VID S W P H D Size offset  
 0 0 1 0 0 0 0 95526 10   
 0 h20000000 image of platform\_specific\_long\_offset  
;---------------------------------------------------------------------  
; loop on all architectures  
 4 1 architecture code (4=armv8), 1 processor  
 ; all processors for this architecture (processor IDs >0)  
 ;------------------------------------------------------------  
 1 1 15 processor ID,   
 ; boolean "main processor" allowed to boot the graphs  
 ; Bit-field computation firmware extensions,

# 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

| Domain name | Code | Comments |
| --- | --- | --- |
| GENERAL | 0 | Any stream, for example (a)synchronous sensor data + electrical, chemical, pixels, .. data from/to the other applications of the processors, compressed streams, JSON, etc .. |
| AUDIO\_IN | 1 | audio files from the application, A/D data from a microphone, line-in, serial audio stream (I2S, PDM RX) |
| AUDIO\_OUT | 2 | line-out, earphone / speaker, PDM TX, I2S |
| GPIO\_IN | 3 | generic digital IO |
| GPIO\_OUT | 4 | generic digital IO , control of relay, |
| MOTION | 5 | accelerometer, combined or not with magnetometer and gyroscope |
| 2D\_IN | 6 | camera sensor |
| 2D\_OUT | 7 | display, led matrix |
| ANALOG\_IN | 8 | analog sensor with aging/sensitivity/THR control, example : light, pressure, proximity, humidity, color, voltage |
| ANALOG\_OUT | 9 | D/A interface, position piezo , PWM converter |
| RTC | 10 | ticks sent from a programmable timer |
| USER\_INTERFACE\_IN | 11 | button, slider, rotary button |
| USER\_INTERFACE\_OUT | 12 | LED, digits, display, |
| PLATFORM\_3 | 13 | platform-specific #3 |
| PLATFORM\_2 | 14 | platform-specific #2 |
| PLATFORM\_1 | 15 | platform-specific #1 |

## 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 | in-place processing |
| io\_direction\_rx0tx1 | 0 | data flow to the graph |
| io\_raw\_format | int16 | short 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](#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](#Data-Types) using the [Declaration of options](#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](#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](#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](#Declaration-of-options) format, in a time unit given in the first parameter. See [Standard units](#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](#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](#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](#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](#Declaration-of-options) format, in a time unit given in the first parameter. See [Standard units](#Standard-units).

io\_sampling\_period 4 {1 1 60 120 } ; sampling period, enumeration in seconds (4)

### io\_sampling\_rate\_accuracy “p”

Percentage of accuracy (or 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](#Audio-stream-format)) :

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 |
| 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](#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 | int16 | short 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)   
 + 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 fixed memory allocation in Bytes. Example :

node\_mem\_alloc 32 ; size = 32Bytes data memory, default Static, default Fast memory block

### node\_mem\_nbchan “B” “i”

Declaration of extra memory in proportion to the number of channel of a specified arc index. Example :

node\_mem\_nbchan 44 3 ; add this amount of bytes : 44 x nb of channels of arc 3

### 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 (parameter ‘1’), or assigned a fixed address at reset (default, parameter ‘0’). When the memory block is relocatable a command ‘STREAM\_UPDATE\_RELOCATABLE’ is sent to the node with address changes. 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; 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 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 processors allowed to use it, 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 their parameters to use at reset time | This section is the translation of the node manifests with 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\_location “x”

Parameter x :

* 0: (default) the destination of the binary graph is fully in RAM address corresponding to the first memory-offset
* 1: the graph stays in Flash and the static portion is copied in RAM is starting from the end of nodes linked-list
* 2: the graph is provided by the application, already in RAM, there is nothing to do

Example :

graph\_location 2 ; copy the compiled graph in RAM

### graph\_location\_offset “x”

The parameter (default 0) gives the binary graph destination offset from the memory-offset #0

Example :

graph\_location\_offset 1024 ; copy the graph at (long\_offset[0] + 1024)

### 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

### set\_file\_path “index” “path”

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

Example :

set\_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 flag

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 “n”

This command starts the declaration of a new format. example format\_raw\_data 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\_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 | 5 | accelerometer, combined or not with pressure and gyroscope audio\_in microphone, line-in, I2S, PDM RX |
| 2D\_IN | 6 | camera sensor audio\_out line-out, earphone / speaker, PDM TX, I2S, |
| 2D\_OUT | 7 | 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 |
| --- | --- |
| only acceleromete | 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 specific 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](#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 pattern placed 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 “n” “i”

A node memory allocation is described in its manifest. A node can ask for up to 6 memory banks with tunable fields :

* type (static, working, 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)  
 + 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"

For example an extra scratch area allocation can added as a function of the number of pixels in the images to process (default 0). 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 informations. 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\_map\_proc 2 ; run this node on processor 2 defined in the architecture manifest   
 node\_map\_arch 1 ; run this node on architecture 1 defined in the architecture manifest   
 node\_map\_rtos 1 ; run this node on thread index 1 (low-latency)

### node\_map\_verbose “level”

The default verbose level on the debug trace of this node is level “0”, meaning “quiet”. Other values from 1 to 3 tell to trace more information. Example:

node\_map\_verbose 3 ; highest level of verbose debug trace

### 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 “k32 k32”

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 X Y ; two 32 bits keys

### 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 12 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 12 registers of the virtual machine are “r0” .. “r11”. 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 = #float 3.14159 load PI in r3  
r4 = mul r3 12 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  
callsys 1 r1 r4 r5 system call (below)

### Graph syntax

script 1 ; script (instance) index   
 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 "callsys" 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)

#### Callsys syntax

| Callsys 1st index | register parameters |
| --- | --- |
| 1 (access to nodes) | 1: address of the node2: command (tag, reset id, cmd) set/read parameter=2/33: address of data4: number of bytes |
| 2 (access to arcs) | 1: arc’s ID2: command set/read data=8/93: address of data4: number of bytes |
| 3 (callbacks of the application) | 1: application\_callback’s ID2: parameter1 (depends on CB)3: parameter2 (depends on CB)4: parameter3 (depends on CB) |
| 4 (IO settings) | 1: IO’s graph index2: command set/read parameter=2/33: address of data4: 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) | 1: command and time format 2: parameter1 (depends on CB)3: parameter2 (depends on CB)4: 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](#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\_graph\_interpreter\_io\_ack() will copy the address of the data in the base address of the arc descriptor
* format ID (see [format “n”](#format-%22n%22)) 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”](#node-arc=%22n%22))
* format ID (see [format “n”](#format-%22n%22)) used to by the node consumer of the stream

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   
; 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

### arc flow control

Example

arc\_flow\_error 1 ; #1 do something depending on the IO domain when a flow error occurs, default #0 (no extra processing, just skip data)

### 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 |
| 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\_debug\_page 0 page of 32 words / page, 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..21 | frame size in Bytes (including the time-stamp field) |
| 0 | 22..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 |
| 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 |
| contrast | 4 | display control |

## 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 |
| 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\_TIME16 | 39 | ssssssssssssssqq q14.2 [s] 4 hours,30mn step = 0.25s |
| STREAM\_TIME16D | 40 | sqqqqqqqqqqqqqqq q1.15 [s] 2 seconds, step=30us |
| STREAM\_TIME32 | 41 | ssssss...ssssssqq q30.2 [s] (34 years step = 0.25s) |
| STREAM\_TIME32D | 42 | sss...ssssqqqqqq...qqqq q17.15 [s] (36h, step=30us) |
| STREAM\_TIME64 | 43 | \_\_\_\_sssssssssss..qqqqqqqqqq q32.26 [s] 140 Y +Q28 [s] |
| STREAM\_TIME64MS | 44 | \_\_\_\_\_\_\_\_\_\_\_mmmmmmmmmmmmmmm u42 [ms], 140Y |
| STREAM\_TIME64ISO | 45 | \_\_\_..YY .. MM..MM..DD..DD..SS..SS.....offs..MM .. ISO8601 with time offset |
| STREAM\_WGS84 | 46 | <--LAT 32B--><--LONG 32B--> |
| STREAM\_HEXBINARY | 47 | UTF-8 lower case hexadecimal byte stream |
| STREAM\_BASE64 | 48 | RFC-2045 base64 for xsd:base64Binary XML data |
| STREAM\_STRING8 | 49 | UTF-8 string of char terminated by 0 |
| STREAM\_STRING16 | 50 | 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 |
| \_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) |
| \_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 |
| —– 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 |
| \_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\_IN | 3 | generic digital IO |
| GPIO\_OUT | 4 | generic digital IO , control of relay, |
| MOTION | 5 | accelerometer, combined or not with pressure and gyroscope |
| 2D\_IN | 6 | camera sensor |
| 2D\_OUT | 7 | display, led matrix, |
| ANALOG\_IN | 8 | analog sensor with aging/sensitivity/THR control, example : light, pressure, proximity, humidity, color, voltage |
| ANALOG\_OUT | 9 | D/A, position piezzo, PWM converter |
| RTC | 10 | ticks sent from a programmable timer |
| USER\_INTERFACE\_IN | 11 | button, slider, rotary button |
| USER\_INTERFACE\_OUT | 12 | LED, digits, display, |
| 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 | sigp\_stream\_resampler | high-quality sampling-rate converter |
| 3 | arm\_stream\_router | copy input arcs and subchannel and output arcs and subchannels |
| 4 | sigp\_stream\_converter | raw data interleaving simple rate converter |
| 5 | arm\_stream\_amplifier | amplifier mute and un-mute with ramp and delay control |
| 6 | arm\_stream\_mixer | multichannel mixer with mute/unmute and ramp control |
| 7 | arm\_stream\_filter | cascade of DF1 filters |
| 8 | sigp\_stream\_detector | estimates peaks/floor of the mono input and triggers a flag on high SNR |
| 9 | arm\_stream\_rescaler | raw data values remapping using “interp1” |
| 10 | sigp\_stream\_compressor | raw data compression with adaptive prediction |
| 11 | sigp\_stream\_decompressor | raw data decompression |
| 12 | arm\_stream\_modulator | signal generator with modulation |
| 13 | arm\_stream\_demodulator | signal demodulator frequency estimator |
| 14 | sigp\_stream\_resampler | asynchronous high-quality sample-rate converter |
| 15 | arm\_stream\_qos | raw data interpolator with synchronization to one HQoS stream |
| 16 | arm\_stream\_split | let a buffer be used by several nodes |
| 17 | sigp\_stream\_detector2D | activity detection pattern detection |
| 18 | arm\_stream\_filter2D | filter rescale/zoom/extract rotate exposure compensation |
| 19 | arm\_stream\_analysis | arm\_stream\_analysis |
| 20 | bitbank\_jpg\_encoder | jpeg encoder |
| 21 | elm\_jpg\_decoder | TjpgDec |
| 22 | arm\_stream\_format\_converter | format converter |

## 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 callsys 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)  
 callsys 1 r2 r0 r3 sp0 ; callsys 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\_format\_converter (TBD)

node arm\_stream\_format\_converter 0  
  
TBD  
  
end

## arm\_stream\_router (TBD)

I/O streams have the same RAW format size, if time-stamped then only with frame size of 1 sample

table of routing : a list of 16bits words (source 8b + destination 8b) format: router arc index (3b) up to 8 arcs sub-channel index -1 (5b) up to 32 channels

example with the router\_2x2 used to merge 2 stereo input in a 4-channels output

node arm\_stream\_router 0  
 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  
 4 i16; 0 1 2 1 move arc0,chan1, to arc2,chan1  
 4 i16; 1 0 2 2 move arc1,chan0, to arc2,chan2  
 4 i16; 1 1 2 3 move arc1,chan1, to arc2,chan3  
 end  
end

## sigp\_stream\_converter (TBD)

Operation : convert input arc format to the format of the output arc. Conversion takes care of RAW data, frame size, interleaving, time-stamp format, number of channels, sampling-rate.

Operations :

* copy input data to the first internal scratch buffer with the management of the target number of channels and using deinterleave intermediate format. The start of the deinterleaved buffer is a copy of the previously processed end of buffers, for the implementation of filters using a memory.
* sampling rate converter, with a dedicated processing for the asynchronous to synchronous conversion, to second scratch buffer
* raw data conversion, interleaving conversion and copy of the result

The case of IMU format is managed like other one-dimension format.

The case of 2D format is special with operations of pixel area extraction, zoom, interpolate, rotation and pixel format conversions.

Parameters :

node arm\_stream\_converter 0  
  
TBD  
  
end

## arm\_stream\_amplifier (TBD)

Operation : 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 occured” “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

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\_mixer (TBD)

Operation : receives several mono or multichannel streams and produces one output arc. Mixer manages up to 8 multichannels input arcs, the number is given at STREAM\_RESET stage.

Parameters : input arcs names, their associated gain and time-constant in [ms], name of the output arc and an extra mixing gain

* index of the HQoS input arc to use for the frame size to use, if the other arcs have flow-errors it will be managed with their respective OVERFLRD\_ARCW2 index

Synchronization operation : arranges multiple input streams are delivered at the same time. Use-case: multi-channel audio stream, each channel is processed with independent signal processing feature. The problem is to avoid (on MP devices) one channel to be delivered to the final mixer ahead and desynchronized from the others (should be OK when the arcs have the sma buffer size / framesize)

preset : #1 (default) : mixer-4 with all arcs unmuted with gain = 0dB, slopes of ~1000 samples #2 (shut-down) : mixer-4 with all arcs with gain = -96dB, slopes of ~100 samples

2 : number of input channels

parameters of mixer (variable size):

* slopes 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]](#footnote-1))’))
* output mixer gain format FP

format to be packed in 32bits words: input arc index (4b) up to 15 arcs sub-channel index -1 (5b) up to 32 channels per arc of the input arc sub-channel index -1 (5b) up to 32 channels per arc of the output arc gain (16b) format FP muted (1b) 1 = “muted”

node arm\_stream\_mixer 0  
  
; Example: 4 input one mono, one stereo, output is stereo  
 parameters 0 ; TAG "load all parameters"  
   
  
 2 i8; 6 6 rising/falling slope on all arcs  
 1 h32; 807 -12dB output gain in FP\_8m4e format;   
  
 4 i16; 0 0 0 arcID0 inputChan0 outputChan0   
 1 h32; 805 0dB gain; mono mixed on the left output  
  
 4 i16; 0 0 1 arcID0 inputChan0 outputChan1   
 1 h32; 805 0dB gain; mono mixed on the right output  
   
 4 i16; 1 0 0 arcID1 inputChan0 outputChan0   
 1 h32; 805 0dB gain; stereo input left to the left  
  
 4 i16; 1 1 1 arcID1 inputChan0 outputChan1   
 1 h32; 803 +12dB gain; stereo input right to the right  
 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 : LPF fc=fs/4 #2 : HPF fc=fs/8 #3 : DC-filter (use-case: audio, XYZ gravity compensation/estimation) #4 : long Median filter

parameter of filter :

* number of biquads in cascade (1 or 2)

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  
 node\_map\_verbose 1 ; level #1 of debug trace  
 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

## 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

## arm\_stream\_rescaler (TBD)

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’)

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

## 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 LC3
* 6 : coder SBC
* 7 : coder mSBC
* 7 : coder OPUS Silk
* 8 : coder MP3
* 9 : coder MPEG-4 aacPlus v2
* 10: coder OPUS CELT
* 11: coder JPEG

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

​ 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 / tone sequencer
* 4 : decoder CVSD for BT speech
* 5 : decoder LC3
* 6 : decoder SBC
* 7 : decoder mSBC
* 7 : decoder OPUS Silk
* 8 : decoder MP3
* 9 : decoder MPEG-4 aacPlus v2
* 10: decoder OPUS CELT
* 11: decoder JPEG

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

## 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

Parameters types and Tags: u8 #1 wave type : 1=sine 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 and 8b DAC 10=PWM 11=ramp 12=step

parameter format for wave generation:

u16 #2 linear amplitude, format UQ16 0dB .. -96dB  
u16 #3 amplitude offset, format Q15 [-1 .. +1]  
f32 #4 wave frequency [Hz] [0.1uHz .. 250kHz], 0Hz translates to DC level.  
s16 #5 starting phase, format S15 [-pi .. +pi]/pi  
u8 #6 modulation type, 0:amplitude, 1:frequency (reserved 2:FSK, ..)  
u8 #7 modulation, 0:none 1=from arc bit stream, 2=from arc q15 words  
u16 #8 modulation index (a) and offset (b), in q15 as fraction of frequency or amplitude  
 modulation y=ax+b, x=input data.  
f32 #9 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

## sigp\_stream\_resampler (TBD)

Operation : convert multichannel input data rate to the rate of the output arcs

* synchronous rate conversion with +/- 100ppm adjustment
* conversion of time-stamp asynchronous to synchronous stream

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 to stop drift compensation TAG\_CMD = 1 to continuously apply a drift compensation of +100ppm TAG\_CMD = 2 to continuously apply a drift compensation of -100ppm

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

## arm\_stream\_qos (TBD)

Operation : align the content of arc buffers with respect to an high-priority one. (HQOS) Parameters : the list of arcs and the one taken as time-alignment reference

node arm\_stream\_qos (i)  
  
 parameters 0 ; TAG "load all parameters"  
TBD  
 end  
end

## arm\_stream\_split (TBD)

Operation : share the input arc to several output arcs, to let several nodes have access to the same buffer without data duplication. Consolidate the output read indexes before updating the input arc read index. There is no priority managed between output arcs. The base, size and write index of the output arcs are identical to the input arc

Parameters : the list of arcs , no memory for instances is needed

node   
 arm\_stream\_split (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  
 4 i16; 0 1 2 1 move arc0,chan1, to arc2,chan1  
 4 i16; 1 0 2 2 move arc1,chan0, to arc2,chan2  
 4 i16; 1 1 2 3 move arc1,chan1, to arc2,chan3  
 end  
end

## arm\_stream\_analysis (TBD)

Operation : spectrum and energy analysis Use-case : debug with script control results are used with “read\_parameter”

presets : #1 : frequency estimator, counting the number of periods Operation : input signal frequency estimator with second-order auto-regression Use-case example: measure 50Hz from power lines on the three phases (a -1% deviation means “power-down”). Parameters : none. output data estimate sampling rate is provided at reset #2 : energy estimation and DC offset #3 : energy analysis on 3 frequency points (Goertzel) #4 : full spectrum 64 points in decibels

node arm\_stream\_analysis (i)  
  
 parameters 0 ; TAG "load all parameters"  
   
 end  
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

## arm\_stream\_filter2D (TBD)

Filter, rescale/zoom/extract, rotate, exposure compensation

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

## JPG encoder

From “bitbank”

## JPG decoder

From “EML”

Use-case : images decompression, pattern generation.

1. 0:15 [↑](#footnote-ref-1)