EARLY DRAFT

Graph-Interpreter

a scheduler of DSP/ML nanoApps

Contents

[Use-cases 2](#_Toc170811693)

[Overview 3](#_Toc170811694)

[Implementation 4](#_Toc170811695)

[Introduction 4](#_Toc170811696)

[Manifests, introduction 4](#_Toc170811697)

[Graph Text, introduction 5](#_Toc170811698)

[Platform Manifest 7](#_Toc170811699)

[Platform manifest 7](#_Toc170811700)

[Digital manifest 7](#_Toc170811701)

[IO Manifest 9](#_Toc170811702)

[IO manifest Header, the domains 9](#_Toc170811703)

[IO configuration 9](#_Toc170811704)

[Graph description 10](#_Toc170811705)

[Control of the scheduler 10](#_Toc170811706)

[Node Manifest 12](#_Toc170811707)

[Node design 14](#_Toc170811708)

[Command RESET 14](#_Toc170811709)

[Command SET PARAMETER 14](#_Toc170811710)

[Command RUN 14](#_Toc170811711)

[Annexe 15](#_Toc170811712)

[Domain-specific IO configuration (TBD) 15](#_Toc170811713)

[Domain “general” 15](#_Toc170811714)

[audio\_in 15](#_Toc170811715)

[audio\_out 15](#_Toc170811716)

[Data types 16](#_Toc170811717)

[Filter and Detector used in the examples 17](#_Toc170811718)

# Use-cases

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

1. **Accelerate time to market**

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

2. **NanoApps repositories**

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

3. **Portability, scalability**

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

**Use-case example :**

**Tuning interfaces for AI preprocessing**

Example of a graph implementing a cascade of DSP/ML algorithms and signal feature extraction before using a classifier (NPU). The system integration task consists in tuning the signal levels and the coefficients of several filters. The system integrator tunes the nodes in charge of rescaling and triggers a GPIO based on level detection. The system integrator updates the parameters of the nodes without recompilation. The memory mapping of the node is tuned on target without recompilation. The task dispatching between processors is tuned for performance optimization, without code recompilation.

# Overview

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

A computer generated diagram of a program

Description automatically generated with medium confidence

Graph of nodes for stream-based computing

The graph to be **interpreted** is coded in a **binary graph** resulting from the **compilation** of the original graph in human-made text format. The graph compiler is a tool executed offline receives the graph and several files called **manifests** givingthe characteristics of the platform, of the nodes and graph interfaces (“**IOs**”). The node designers deliver their code with a **node manifest**. The system integrator provides a **platform manifest,** and **IO manifests** of the possible stream connexions.

**Graph Text**

**+**

**nanoApp manifests**

**Platform manifest**

**Graph interfaces manifests**

**List of installed nodes**

**List of app’s callbacks**

**Scripts**

**Compact binary format of the graph**

**Content :**

1. **Header**.

2. **IO-interfaces**.

3. **Data and stream formats**.

4. **Scripts** of byte codes.

5. **nodes and parameters**

6. **FIFO description** of arcs.

7. **Memory sections**.

**GRAPH COMPILATION**

**GRAPH EXECUTION**

**Graph interpreter of the binary graph**

1. **Parse the graph IOs**

2. **Parse the graph of nodes**

3. lock the execution

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

5. **Return** to application

Processing flow for the binary graph generation

The binary graph is a list of data structures describing : the nodes, the list of arcs (**FIFO descriptors** of the buffers used by the arcs) and their connexions between nodes.

The graph scheduler starts parsing the arcs at the boundary of the graph to fill new data or push new data out. Then it parses the list of arcs searching for the ones holding enough data to trigger the execution of a consumer node, and checking this node has enough free space in its output arc buffer. The parsing activity is configured to stop after a node execution or when no more data is available.

The graph scheduler is written in C90 and delivered with a small set of nodes implementing basic operators like : filters, mixer, router, detector, compressor, rate and data format converters.

# Implementation

## Introduction

In short, a graph is made of arcs and nodes. The arcs are implemented as circular buffers. The nodes are single-entry-point subroutines with four parameters: a **command**, a pointer to the node **instance**, a pointer to the list of input/output buffers, and a returned status.

The commands are “reset”, “set” and “read parameters”, “run” and “stop”. A minimal wrapper will make legacy codes compatible with this interface.

A node is delivered with a Manifest giving : the name, the author and the parameters (the number of arcs it is connected to, how memory is allocated, etc ..). The manifest is minimal in the following situation :

* The node has one input arc and one output arc, the processing can be managed using any frame length, there is one channel per frame, the raw data format is made of fixed-point integers in Q15 format, there is no time-stamps on frames, the processing is independent of the data rate, the amount of data consumed and produced is identical, the average data rate is identical on the two arcs.
* The node manages its memory allocation by calling the C standard library or by static allocation of its objects.

In all other situations the “node manifest” syntax helps tuning those parameters.

The graph Interpreter manages combination of situations and stream domains (image streams, audio, drifting stream, nodes assigned to a specific processor, three levels of priorities assigned to nodes, embedded scripts of byte-codes for debug and control ..).

## Manifests, introduction

The Graph Interpreter is managing the same ways a wide variety of situations. The manifest files are used define the specific situations and options on each parameters. The **parameters options** have the following syntax : a name and options within brackets.

Name\_of\_the\_parameter\_with\_options *{* index values\_or\_ranges *}*

If “index” is null it means "any value is possible", the list can be empty. When “index” is a positive integer the following data (in floating-point format) are a list of allowed values the scheduler can select. The Index tells the default value to take at reset time (starting from 1).

When “index” is a negative integer it is followed by three values to describe a range: the first possible value, the increment step and the last possible value. The absolute value of the index selects the value in this range (starting from 1).

Example, a graph interface can have a sampling rate within a predefined list of values, and the 3rd one if the default selected at reset (44.1kHz).

io\_sampling\_rate { 3 16000 32000 44100 48000 }

Example, a graph interface allows to control the gain within a range going from -12dB to +12dB by steps of 0.5dB, and the default gain value is the 25th in this range (corresponding to 0dB).

io\_digital\_gain { -25 -12 0.5 12 }

## Graph Text, introduction

The graph gives the list of stream formats, the graph boundaries and the parameter to set during the initialization of the nodes. In the example below a stream of data from an IO named “ADC” (in pratice it may be a data stream from the calling application) is filtered through a 4th order bandpass filter, the result is used for a signal detector which triggers a GPIO.

Here below the graph syntax for declaring the detector node is “ node arm\_stream\_detector 0 ”, where the second field is the name of the node declared in the platform manifest (see below) and “0” is the instance index of the node in case several detectors are used in the application.

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

A graph description includes the **presets** (preconfigured set of parameters the developer incorporates in the node) and **parameters** of the nodes to use at graph reset time. When the parameters and states need to be exchanged dynamically during the graph execution a **script** can be coupled before/after the execution of each node.

The scripts consist in a **byte-code** language (TBD). A global **script** can be used for the interactions with the application, for example to change parameters during use-case transitions and avoid creating a dependency between the application and the way the graph is designed. A graph can be reused as a **sub-graph** of a more complex graph.

The Graph scheduler / interpreter is using two subroutine interfaces : one used by the application and one to receive notifications at the end of data moves on graph boundaries.

The first interface is :

arm\_graph\_interpreter (uint32\_t command, arm\_stream\_instance\_t \*instance, uint8\_t \*data, uint32\_t size)

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

The second interface is the call-back used to notify the end data moves with IOs : arm\_graph\_interpreter\_io\_ack (uint8\_t IO\_AL\_idx, uint8\_t \*data, uint32\_t data\_size)

The parameters of this function tell the “data” pointer with an amount of “size” bytes have been exchanged on the graph boundary with the interface indexed by IO\_AL\_idx.

This subroutine can be called under interrupts. It reads the binary graph information to update the **arc circular buffer descriptor** after this data move.

# Platform Manifests

A **platform** is characterized by the list of interfaces, the list of processors and their **memory banks** pre-reserved for the execution of the graph (this is for embedded devices, when using Linux the memory allocation is dynamic).

The platform is characterized by the list of the **nodes pre-installed** before graph execution. The platform description is made of two files : a “top level description” with the list of possible interfaces of the graph and the list of nodes. And a “digital description” giving the memory mapping and characteristics of the processors.

## Platform manifest

Indexes of the file paths to read the manifest files.

List of possible interfaces to use at the boundary of the graph.

List of pre-installed nodes and their manifests.

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

; 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

3 three file paths

../../stream\_platform/ "" path index 0 is local

../../stream\_platform/windows/manifest/ "" path index 1

../../stream\_nodes/ "" path index 2

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

; 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 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 codes from platform\_computer.h

1 io\_platform\_data\_in\_0.txt 0 1 0 application processor IO\_PLATFORM\_DATA\_IN\_0

1 io\_platform\_data\_in\_1.txt 1 1 0 application processor IO\_PLATFORM\_DATA\_IN\_1

1 io\_platform\_analog\_sensor\_0.txt 2 1 0 ADC IO\_PLATFORM\_ANALOG\_SENSOR\_0

1 io\_platform\_motion\_in\_0.txt 3 1 0 accelero=gyro IO\_PLATFORM\_MOTION\_IN\_0

1 io\_platform\_audio\_in\_0.txt 4 1 0 microphone IO\_PLATFORM\_AUDIO\_IN\_0

1 io\_platform\_2d\_in\_0.txt 5 1 0 camera IO\_PLATFORM\_2D\_IN\_0

1 io\_platform\_line\_out\_0.txt 6 1 0 audio out stereo IO\_PLATFORM\_LINE\_OUT\_0

1 io\_platform\_gpio\_out\_0.txt 7 1 0 GPIO/LED IO\_PLATFORM\_GPIO\_OUT\_0

1 io\_platform\_gpio\_out\_1.txt 8 1 0 GPIO/PWM IO\_PLATFORM\_GPIO\_OUT\_1

1 io\_platform\_data\_out\_0.txt 9 1 0 application processor IO\_PLATFORM\_DATA\_OUT\_0

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

; SOFTWARE COMPONENTS MANIFESTS

20 nodes path index + file name, in the same order of p\_stream\_node node\_entry\_point\_table[NB\_NODE\_ENTRY\_POINTS]

; p\_stream\_node node\_entry\_point\_table[NB\_NODE\_ENTRY\_POINTS] =

2 swc\_manifest\_none.txt /\* 0 ID0 is reserved for by-passes \*/

2 Basic/arm/script/swc\_manifest\_script.txt /\* 1 arm\_stream\_script byte-code interpreter (arm\_stream\_script\_index = 1)\*/

2 Basic/arm/script/swc\_manifest\_graph\_control.txt /\* 2 arm\_stream\_graph\_control scheduler control : lock, bypass, loop, if-then \*/

2 Basic/arm/router/swc\_manifest\_router.txt /\* 3 arm\_stream\_router copy input arcs and subchannel and output arcs and subchannels \*/

2 Basic/arm/converter/swc\_manifest\_converter.txt /\* 4 arm\_stream\_converter raw data format converter \*/

2 Basic/arm/amplifier/swc\_manifest\_amplifier.txt /\* 5 arm\_stream\_amplifier amplifier mute and un-mute with ramp and delay control \*/

2 Basic/arm/mixer/swc\_manifest\_mixer.txt /\* 6 arm\_stream\_mixer multichannel mixer with mute/unmute and ramp control \*/

. . .

2 AI/arm/detector2D/swc\_manifest\_detector2D.txt /\* 17 arm\_stream\_detector2D image activity detection \*/

2 image/arm/filter2D/swc\_manifest\_filter2D.txt /\* 18 arm\_stream\_filter2D convolution filter of the image \*/

2 Basic/arm/analysis/swc\_manifest\_analysis.txt /\* 19 arm\_stream\_analysis energy, spectrum analysis \*/

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

## Digital manifest

System memory map with shareable, private, speed information.

List of processors and their pre-installed list of compute services.

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

; Processor and memory configuration + default memory mapping

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

;

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

; memory banks:

; - ID base offset ID reference above

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

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

; - W static0/working1/retention2,

; - P shared0/private1,

; - H DMAmemHW1

; - D Data0/Prog1/Both2

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

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

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

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

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

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

0 2 0 0 0 0 0 30000 65536 SRAM1 static

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

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

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

1 10 0 2 0 0 0 1024 524288 Retention memory

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

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

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

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

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

; 2 h2C000000 TCM Private memory of processor 1

; 3 h08000000 Internal Flash

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

; all architectures

; all processors (processor IDs >0)

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

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

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

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

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

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

A system integrator can avoid specifying the VID memory mapping and let the graph compiler manage using ID/VID=0. Tuning the performance means taking care of overlays, or arranging processors don’t have simultaneous access to the same physical memory sub-blocks, and this is where VID indexes are used.

# IO Manifests

The arcs at the boundary of the graph are called “**IOs**” (as Input / Output ports mapped to a software or hardware interface). The IOs are characterized by the physical **domain** of operation (for example a connexion with the application, an image or audio stream, a motion sensor, a GPIO).

The IO is described by the commander / servant protocol to use, the data stream format, and a function implementing the data move, initialization, settings, and stop. Those functions are located in the scheduler’s platform **AL** (platform Abstraction Layer. Each IOs function has an index named **IO\_AL\_idx**.

Example of IO manifest.

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

; Manifest of interface abstraction to ADC converter and analog sensor

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

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

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

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

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

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

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

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

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

\_end\_

### IO manifest Header, the domains

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

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

domain name description and examples

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

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

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

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

gpio\_in generic digital IO , control of relay,

gpio\_out generic digital IO , control of relay,

motion accelerometer, combined or not with pressure and gyroscope

2d\_in camera sensor

2d\_out display, led matrix,

analog\_in with aging control

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

rtc ticks sent from a programmable timer

user\_interface\_in button, slider, rotary button

user\_interface\_out LED, digits, display,

### IO configuration

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

# Graph description

A graph text several sections :

* Control of the scheduler : debug option, location of the graph in memory
* File path : 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 boundary of the graph: the IOs are a kind of node with on arc producing or consuming a stream of data
* The scripts, are a byte-code interpreted language used for simple operations like setting parameters, sharing debug information, calling “callbacks” predefined in the application.
* The list of nodes, without their connexions with other nodes. This section defines the boot parameters, the memory mapping hand optimized.
* The list of arcs, their relations with two nodes and the minimal type of debug activity on each transaction

## Control of the scheduler

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters** | **Comments** |
| set\_file\_path | Int String | A graph can “include” data from other file, this command creates an index to use instead of a computer file patch s Index and its file path, used for sub graphs and scripts |
| graph\_location | 1 | 0: destination of the binary graph is in RAM (default)  1: keep the graph in Flash and copy in RAM the portion starting from PIO (the end of node linked-list)  2: the graph is already in RAM provided by the application |
| debug\_script\_fields | 24 | LSB set means "call the debug 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 is called when starting the graph scheduling"  bit 4 (16) set means "call the debug script is called when returning of the graph scheduling"  no bit is set (default) the debug script is not called (default 0) |
| scheduler\_return | 3 | 1: return to caller after each node calls  2: return to caller once all nodes are parsed  3: return to caller when all nodes are starving (default 3) |
| allowed\_processors | 255 | bit-field of the processors allowed to execute this graph, (default = 1 main processor) |
| graph\_map\_hwblock | 0 | index of the memory block indexes where to map the graph. Default VID's is 0 internal RAM |

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters** | **Comments** |
| format | 2 | index used to start the declaration of a new format |
| format\_raw\_data | 17 | raw data of this format (17 : S16 is the default) |
| format\_frame\_length | 160 | frame length in number of bytes (default :1) |
| format\_interleaving | 0 | 0 means interleaved data, 1 means deinterleaved data by packets of "frame size" |
| format\_nbchan | 1 | number of channels in the stream (default 1) |
| format\_time\_stamp | 0 | time-stamp format 0:none, 1:absolute time-stamp, 2:relative time, 3:simple counter |
| format\_time\_stamp\_size | 0 | 0:16bits 1:32bits 2:64bits (see "STREAM\_TIME16D" for example) |
| format\_domain | 2 | domain type (0 means "any")  general (a)synchronous sensor + rescaling, .. remote data, compress  audio\_in microphone, line-in, I2S, PDM RX  audio\_out line-out, earphone / speaker, PDM TX, I2S,  gpio\_in generic digital IO , control of relay,  gpio\_out generic digital IO , control of relay,  motion accelerometer, combined or not with pressure and gyroscope  2d\_in camera sensor  2d\_out display, led matrix,  analog\_in with aging control  analog\_out D/A, position piezzo, PWM converter  rtc ticks sent from a programmable timer  user\_interface\_in button, slider, rotary button  user\_interface\_out LED, digits, display,  platform\_3  platform\_2 platform-specific #2, decoded with callbacks  platform\_1 platform-specific #1, decoded with callbacks |
| format\_sdomain |  | subdomain |
| format\_sampling\_rate |  | tbd |
| format\_audio\_mapping |  | tbd |
| format\_motion\_mapping |  | tbd |
| format\_2d\_height |  | tbd |
| format\_2d\_width | int | tbd |
| format\_2d\_border | int | tbd |

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

|  |  |  |
| --- | --- | --- |
| **Tags** | **Parameters** | **Comments** |
| node\_parameters | 0 | Start of a parameter section ending with “\_end\_”, the parameter is the “TAG” telling the index of specific parameters to update, or “0” meaning “all the parameters are reloaded with the following data”. The format of the following parameters is : number of fields, data type (u8/s8/h8, u16/s16/h16, u32/s32/h32, f32, f64), “;”, the data, and optional comments. For example :  1 u8; 2 Two biquads  5 f32; 0.284277f 0.455582f 0.284277f 0.780535f -0.340176f ellip(4, 1, 40, 3600/8000, 'low') |
| node\_preset | 1 | parameter preset used at boot time, default = #0 |
| node\_malloc\_E | 12 | "E" parameter used in "Memory Size Bytes", default = #0 |
| node\_map\_hwblock | 2 3 | index of the memory block "node\_mem" and the VID indexes from "procmap\_manifest\_xxxx.txt" where to map it. Default VID's is 0. |
| node\_map\_copy | 2 3 | copy the indexed "node\_mem" to VID 3 (faster memory) before run |
| node\_map\_swap | 2 3 | swap the indexed "node\_mem" to VID 3 (faster memory) before run, and restored after |
| node\_trace\_id | 0 | IO port used to send the trace |
| node\_map\_proc | 0 | execute this node on this processor (0: any possible, default) |
| node\_map\_arch | 0 | execute this node on this architecture (0: any possible, default) |
| node\_map\_rtos | 0 | execute this node on this thread index (0: any possible, default) |
| node\_map\_verbose | 0 | level of debug trace, default = #0 |
| node\_script | <0..127> | index of the script to call before and after execution of this node |

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

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

# Node Manifests

The interpretation process is helped by the compilation of manifests data. A node’s manifest is giving the amount of memory blocks to allocate, and their characteristics (static memory blocks, scratch, retention memory, critical fast or “best effort speed”, alignment). The node’s manifest gives the stream format used on each input/output arc connected to the node. A node can be connected up to 8 arcs, each having different stream format (raw data, frame size, sampling-rate, interleaving, time-stamp format). Those details are packed in the node binary structure interpreted by the scheduler.

Identification

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

Graph parameters

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

Node memory allocation

memory allocation size in bytes =

A : memory allocation in Bytes (default 0)

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

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

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

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

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

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

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

# Node design

The nodes have a single entry point with the format “func (int, \*,\*,\*)”. The first parameter is the command, a 32bits bit-field (the command to execute, the “PRESET” defined in the graph, the parameter “TAG” used during SET PARAMETER.

## Command RESET

The second parameter points to the list of memory banks (field “node\_mem” in the manifest), starting with the node instance. The list continues with the format of the input and output arcs (field “node\_use\_arc\_format” in the manifest). The format structure gives the frame size, number of channels, interleaving and time-stamp scheme, raw data type, and domain-dependent details (image format, audio mapping, etc..).

The second parameter is the address the function used for services (DSP / ML / math computing libraries, time library, multimedia library, interface to the standard library for nodes delivered in binary format).

## Command SET PARAMETER

The second parameter is the node instance.

The third parameter is the address of parameters to load. When the TAG is null all the parameters are updated, otherwise only the ones selected by TAG are updated after using the ones of the PRESET.

## Command RUN

The second parameter is the node instance.

The third parameter points to a list of buffers : [{data pointer size1} {data pointer size2} .. ]. For input streams the “size” if the amount of data in the buffer, for output streams the “size” is the free area available in the buffer.

When the manifest tells “node\_steady\_stream 0” the buffers have variable consumption and production size, the node updates the “size” field with the meaning “amount of input data consumed” for input streams, and “amount of data produced” on the output arcs.

When a stream data format is “deinterleaved” (for example Left and Right audio samples are in this order : LLLL..LLLLRRRR...RRRR ) the size of the first buffer (the "frame") the pointer to the following channels is computed by incrementing the base pointer address with the size of the frame.

# Annexe

## Domain-specific IO configuration (TBD)

### Domain “general”

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

### audio\_in

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

### audio\_out

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

## Data types

Raw data types

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

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

STREAM\_U1 2 one bit unsigned, boolean

STREAM\_S2 3 SX two bits per data

STREAM\_U2 4 XX

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

STREAM\_S4 6 Sxxx four bits per data

STREAM\_U4 7 xxxx

STREAM\_Q3 8 Sxxx

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

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

STREAM\_S8 11 Sxxxxxxx eight bits per data

STREAM\_U8 12 xxxxxxxx ASCII char, numbers..

STREAM\_Q7 13 Sxxxxxxx arithmetic saturation

STREAM\_CHAR 14 xxxxxxxx

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

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

STREAM\_S16 17 Sxxxxxxx.xxxxxxxx 2 bytes per data

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

STREAM\_Q15 19 Sxxxxxxx.xxxxxxxx arithmetic saturation

STREAM\_FP16 20 Seeeeemm.mmmmmmmm half-precision float

STREAM\_BF16 21 Seeeeeee.mmmmmmmm bfloat

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

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

STREAM\_S32 24 one long word

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

STREAM\_Q31 26 Sxxxxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx

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

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

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

STREAM\_S64 30 long long 8 bytes per data

STREAM\_U64 31 unsigned 64 bits

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

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

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

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

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

STREAM\_CFP64 37 fp64 fp64 (I Q)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Physical units (RFC8428 RFC8798)

STREAM\_SUBT\_ANA\_ANY 0 any

STREAM\_SUBT\_ANA\_METER 1 m meter

STREAM\_SUBT\_ANA\_KGRAM 2 kg kilogram

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

STREAM\_SUBT\_ANA\_SECOND 4 s second

STREAM\_SUBT\_ANA\_AMPERE 5 A ampere

STREAM\_SUBT\_ANA\_KELVIB 6 K kelvin

STREAM\_SUBT\_ANA\_CANDELA 7 cd candela

STREAM\_SUBT\_ANA\_MOLE 8 mol mole

STREAM\_SUBT\_ANA\_HERTZ 9 Hz hertz

STREAM\_SUBT\_ANA\_RADIAN 10 rad radian

STREAM\_SUBT\_ANA\_STERADIAN 11 sr steradian

STREAM\_SUBT\_ANA\_NEWTON 12 N newton

STREAM\_SUBT\_ANA\_PASCAL 13 Pa pascal

STREAM\_SUBT\_ANA\_JOULE 14 J joule

STREAM\_SUBT\_ANA\_WATT 15 W watt

STREAM\_SUBT\_ANA\_COULOMB 16 C coulomb

STREAM\_SUBT\_ANA\_VOLT 17 V volt

STREAM\_SUBT\_ANA\_FARAD 18 F farad

STREAM\_SUBT\_ANA\_OHM 19 Ohm ohm

STREAM\_SUBT\_ANA\_SIEMENS 20 S siemens

STREAM\_SUBT\_ANA\_WEBER 21 Wb weber

STREAM\_SUBT\_ANA\_TESLA 22 T tesla

STREAM\_SUBT\_ANA\_HENRY 23 H henry

STREAM\_SUBT\_ANA\_CELSIUSDEG 24 Cel degrees Celsius

STREAM\_SUBT\_ANA\_LUMEN 25 lm lumen

STREAM\_SUBT\_ANA\_LUX 26 lx lux

STREAM\_SUBT\_ANA\_BQ 27 Bq becquerel

STREAM\_SUBT\_ANA\_GRAY 28 Gy gray

STREAM\_SUBT\_ANA\_SIVERT 29 Sv sievert

STREAM\_SUBT\_ANA\_KATAL 30 kat katal

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Secondary Unit (rfc8798) Description SenML Unit Scale Offset

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Filter and Detector used in the examples

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

; 7 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)

; Option : either the same coefficients for all channels or list of coefficients for each channel

;

; presets:

; #1 : bypass

; #2 : LPF fc=fs/4

; #3 : DC-filter (use-case: audio, XYZ gravity compensation/estimation)

;

; parameter of filter :

; - number of biquads in cascade (1 or 2)

; - coefficients in FP32

;

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.txt" where to map the allocated memory

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 b0/b1/b2/-a1/-a2 ellip(4, 1, 40, 3600/8000, 'low')

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\_

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

; 8 arm\_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

parameters 0 ; TAG "load all parameters"

8; i8; 1 2 3 4 5 6 7 8 the 8 bytes of "struct detector\_parameters"

\_end\_

\_end\_

;