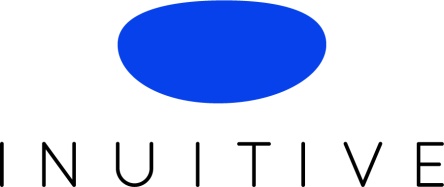
NU4000 FDK Read Frame on ARM core

User Guide

Revision 1.0



March, 2022

**Table of changes**

|  |  |  |
| --- | --- | --- |
| **version** | **Date** | **description** |
| 1.0 | 30/03/2022 | Initial creation |

Contents

[1 Introduction 1](#_Toc99628559)

[2 ARM core application 2](#_Toc99628560)

[3 Windows host application 2](#_Toc99628561)

[4 SW graph 3](#_Toc99628562)

[4.1 sw\_graph\_boot60\_FDK.xml 3](#_Toc99628563)

[4.2 sw\_graph\_boot60\_FDK-without-separation.xml 4](#_Toc99628564)

[4.3 Notes 4](#_Toc99628565)

[4.3.1 Data node 4](#_Toc99628566)

[4.3.2 Function node 5](#_Toc99628567)

[5 Demo activation 6](#_Toc99628568)

[6 Output 6](#_Toc99628569)

[6.1 sw\_graph\_boot60\_FDK.xml 6](#_Toc99628570)

[6.2 sw\_graph\_boot60\_FDK-without-separation.xml 7](#_Toc99628571)

[7 Known issues 7](#_Toc99628572)

# Introduction

This is a sample application based on basic demo FDK.

It’s location is under:

nu4000c0\src\client\_demo\read\_frame

The purpose of this application is to demonstrate a simple access to a frame on ARM core. It is intentionally simplified so that only the important functions are activated.

All the processing is done on RGB channel of M4.3VN module.

Via this example, developer can understand:

* How to send messages from Host to FDK node on ARM
* How to send messages from FDK node on ARM towards Host
* How to access the frame data on ARM core, inside FDK node
* How to separate between streams before/after FDK processing
* How to use SDK commands on Host side to activate the FDK node

This example has 2 applications:

* “Target” application that runs on ARM core and implements the FDK node
* “Host” application that runs on Windows machine and implements control – activation and display of the FDK streams.

# ARM core application

ARM code implementation is in the basic\_demo\_gp.c file.

The node is activated by Host sending message, that contains parameters for processing and enable bit.

After initial activation, no host commands are sent to ARM core, it acts indepenetly.

FDK node operate parameters inputs array includes 2 fileds:

[DATA\_RGB\_INPUT][DATA\_RAW\_OUTPUT]

* DATA\_RGB\_INPUT is the frame that arrives from RGB camera.
* DATA\_RAW\_OUTPUT is data object which is created, updated and sent to host application.

When a new frame arrives from RGB camera, BASIC\_DEMO\_operate(inu\_fdkH meH, inu\_function\_\_operateParamsT \*paramsP) function is called.

This function creates new data object, from type ‘RAW\_DATA’, that will be sent to Host after some processing is done on the frame (DATA\_RAW\_OUTPUT).

After the data object is created, we access DATA\_RGB\_INPUT, do a simple processing on it and call the BASIC\_DEMO\_finish\_operate(meH, paramsP) function.

BASIC\_DEMO\_finish\_operate(meH, paramsP) function assigns calculated values to the message that is sent in DATA\_RAW\_OUTPUT and releases the data.

# Windows host application

Windows host application uses “User Defined Stream” API on Host side, that enables:

* To start FDK streams
* To send pre-defined data structures to ARM core, in this case the ROI that will be needed for processing.
* To receive the data sent from ARM core as callbacks.
* To send message to Target with different parameters, modify the structure values inside BasicDemoHost.sln Execute() function and rebuild the solution:

BASIC\_DEMO\_hostGpMsgStructT hostGPmsg;

memset(&hostGPmsg,0,sizeof(BASIC\_DEMO\_hostGpMsgStructT));

hostGPmsg.algInfo.enable = 1;

hostGPmsg.algInfo.startX = 10;

hostGPmsg.algInfo.startY = 10;

hostGPmsg.algInfo.picWidth = 200;

hostGPmsg.algInfo.picHeight = 200;

# SW graph

SW graph represents the data flow in the system. It defines connectivity between the function and data nodes.

Function node is any algorithm/functionality defined by Inuitive.

FDK is function node defined by the customer. It can implement any functionality on any of the NU4000 cores.

In this example, we provide 2 graphs, to show how it can be deployed.

## sw\_graph\_boot60\_FDK.xml

This is the default xml that will be loaded by host application. It enables to see the “clean” frame before processing and the processed frame.

The frame that is injected to FDK\_1 node is an instance of IMAGE\_9 – we called it IMAGE\_9Pre Process. It is the buffer on which the processing is done, while IMAGE\_9 is streamed out to the host.

This way there is a “clean” frame that comes out of Stream\_RGB and “processed” frame that comes out of Stream\_out\_FDK\_image. From camera point of view, this is of course, same frame.

FDK\_RAW\_data is the other data FDK provides, the counter and clock.

Diagram

Description automatically generated

## sw\_graph\_boot60\_FDK-without-separation.xml

To change the application to use this xml, please either rename it to sw\_graph\_boot60\_FDK.xml or modify this line in the host application:

inuSensorParams.GraphXmlPath = build\_file\_name(2, current\_directory, "sw\_graph\_boot60\_FDK.xml");

to:

inuSensorParams.GraphXmlPath = build\_file\_name(2, current\_directory, "sw\_graph\_boot60\_FDK-without-separation.xml");

In this option, there is no separation, the input to FDK is IMAGE\_9. This is the buffer that gets processed, therefore there will be no “clean” frame, in both steamers we will see processed frames.

Diagram

Description automatically generated

(for more information about FW graph architecture please refer to FW FDK doc.pdf)

## Notes

In this section, there are some notes to explain some special points of the xml.

### Data node

* <source> parameter

The <source> parameter is the source of the data. In most cases, it will be similar to the <input>. However, in some cases, the data may be source for another node. Source acts like input, but not always direct input from the previous node.

When we define source to node, the graph engine have the ability to use same data (frame) along the whole flow of nodes of graph.

For example, if channel is connected to streamer, once streamer function is done, the frame will be released. But, if besides the channel there is another function that needs it – CNN for example, then we would like this frame to be kept for CNN processing even though the streamer already done with it.

* Source of the data defines data type, for example, data that is defined like below,

<data>

<name>FDK\_image\_data</name>

<type>IMAGE\_DATA</type>

<input>FDK\_1</input>

<source>**INU\_SOC\_CHANNEL\_9**</source>

<params></params>

</data>

Will be created by graph engine as a clone of the frame data.

* When <input> = <source>, user needs to define the sizes and parameters of the data. When <input> is different from <source>, data header will be taken from the <source> data.
* In this specific example, we create RAW\_DATA and add it to inputs of the operate parameters:

ret = BASIC\_DEMO\_createRawData(meH, &newRawDataH);

However, the IMAGE\_DATA does not need to be created by FDK, because the <source> of it is the hw channel.

* INU\_SOC\_CHANNEL\_9

This is the virtual representation of hw function that creates the IMAGE\_9.

### Function node

* <numBuffers> and <sizeBuffer> parameters
* Each function uses pre-defined memory size, taken from mempool. When the function is created, the size and number of buffers are allocated for its use, by the <numBuffers> and <sizeBuffer> parameters.

The size should be big enough to hold the size of output of the function. For example, if the output is frame, then the <sizeBuffer> is frame size. Number of buffers depend on how many outputs will need to be stored.

# Demo activation

To run the demo:

* Go toINSTALL\_DIR\InuDev\bin\NU4000c0\boot0\nu4000c0.zip\_ and move the directory to a temporary location
* Rename nu4000c0.zip\_ to nu4000c0.zip
* Copy inu\_target.out (that was compiled or provided by Inuitive) to nu4000c0.zip
* Copy read\_frame directory to:

INSTALL\_DIR\InuDev\FDK\nu4000c0\src\client\_demo

* Open host project from VS Solution (Windows):

INSTALL\_DIR\InuDev\FDK\nu4000c0\src\client\_demo\basic\_demo\host\BasicDemoHost.sln

* Make sure the Solution Configurations in the Toolbar is set on Release (and not Debug) and build the project.
* Run the project

# Output

## sw\_graph\_boot60\_FDK.xml

* Two windows will be opened, both display the same frame:
  + “Original RGB image” – clean frame
  + “RGB Image after FDK” – processed frame
* In console window output of RAW FDK data will be printed

A picture containing text, indoor, computer, electronics

Description automatically generated

## sw\_graph\_boot60\_FDK-without-separation.xml

* Two windows will be opened, both display the same frame. In this case, both Original RGB image” and “RGB Image after FDK” look the same, because the processing is done on IMAGE\_9, which is common for first streamer and the FDK output.
* In console window output of RAW FDK data will be printed

A picture containing text, indoor, computer

Description automatically generated

# Known issues

* The processed area does not change accordingly to the requested ROI, it has ashift in coordinates (the original function was written for YUV). But it is not the point of this example so it was left as is.
* There is an issue with ARM compilation (missing library), for compiling the ARM please take all nu4000c0 folder that was sent to you to the ARM toolchain environment (instead of using the one from installation package).