n-BlocksStudio

Table of Contents

n-BlocksStudio

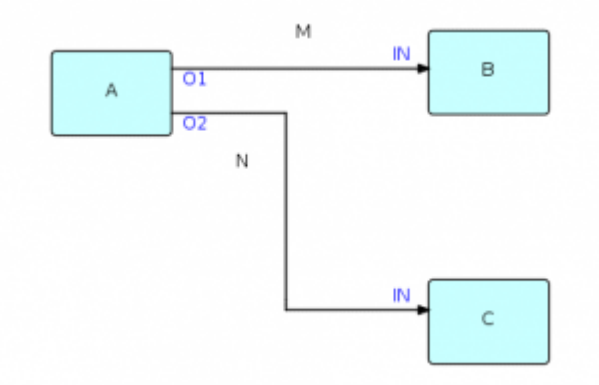
- **n-BlocksStudio** is a programming environment which facilitate an easy way to develop applications with n-Blocks based embedded systems or Internet of Things hardware.
- The main requirement in the design of n-BlocksStudio was to enable users to develop applications without having to write code (although this may not always be possible).
- The IDE uses the Flow Based Design paradigm, using interconnected nodes to generate underlying code
- The code that is created by n-BlocksStudio runs in a soft-realtime* firmware system (similar to, but not as complex as an embedded operating system)
- Inititally, we used node-RED code base for the first prototype, but soon realized that a development from scratch using Python provided a better development roadmap
- The current n-BlocksStudio runs locally on a PC, making use of OpenGL 3D rendering for attracting visualization
- The Logic and behaviour of nodes is downloaded as Libraries from a public repository
- The public server hosts libraries written by the n-Blocks team as well as by the users, who contribute and share nodes with the n-Blocks user community

*soft-realtime means tasks are guaranteed to perform in the average time, but timing for each task is not precise – as opposed to hard-realtime, in which the time for each task is strictly met and deterministic

Video

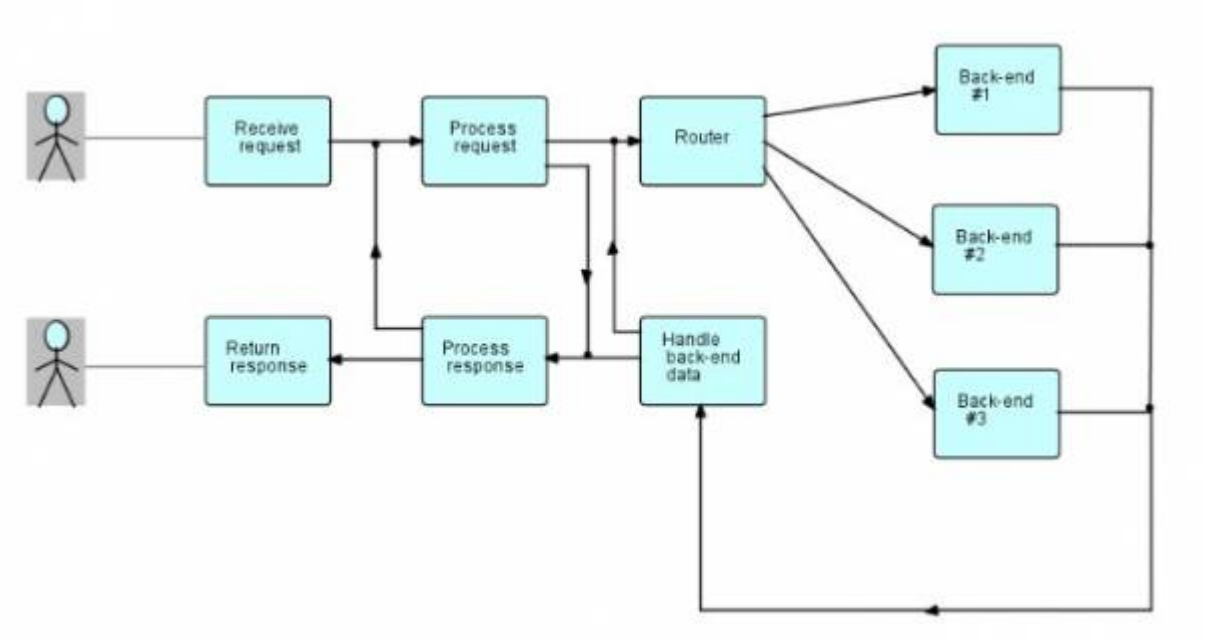
Flow based programming

In computer programming, flow-based programming (FBP) is a programming paradigm that defines applications as networks of “black box” processes, which exchange data across predefined connections by message passing, where the connections are specified externally to the processes. These black box processes can be reconnected endlessly to form different applications without having to be changed internally. FBP is thus naturally component-oriented. FBP is a particular form of dataflow programming based on bounded buffers, information packets with defined lifetimes, named ports, and separate definition of connections. https://en.wikipedia.org/wiki/Flow-based_programming

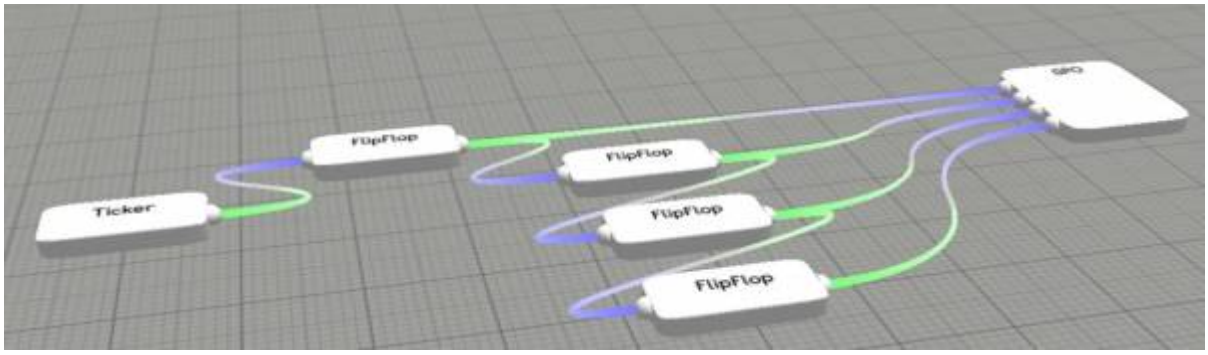


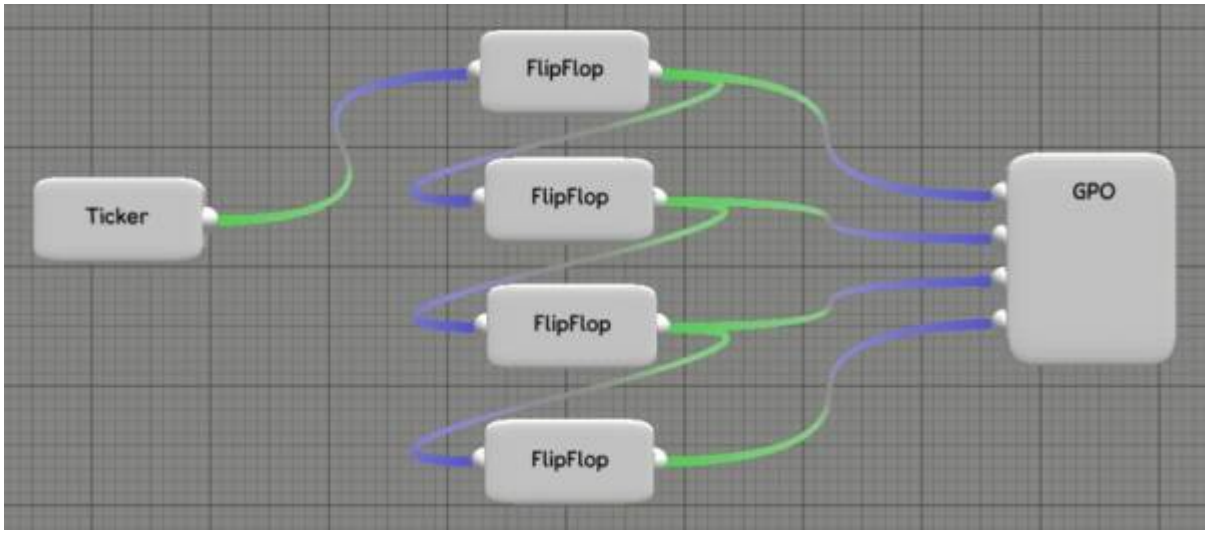
J. Morrison; Flow based programming

In computer programming, Flow-Based Programming (FBP) is a programming paradigm, discovered/invented by J. Paul Rodker Morrison in the late '60s, that uses a “data processing factory” metaphor for designing and building applications. FBP defines applications as networks of “black box” processes, which communicate via data chunks (called Information Packets) travelling across predefined connections (think “conveyor belts”), where the connections are specified externally to the processes. These black box processes can be reconnected endlessly to form different applications without having to be changed internally. FBP is thus naturally component-oriented.https://jpaulm.github.io/fbp/



Develop Application with Diagrams





- IDE to develop application with Diagrams
- No need to write code, just use nodes and connections
  - Reduced complexity
  - Modularity
  - Expandable Library of Nodes
- Friendly for
  - Makers, new firmware developers
  - Experienced embedded developers

Behind the Nodes

adc.cpp

```
#include "adc.h"

/// GPI
nBlock_ADC::nBlock_ADC(PinName pinAdc): _adc(pinAdc) {
    return;
}

void nBlock_ADC::triggerInput(uint32_t inputNumber, uint32_t value) {
    // Input 0 triggers a read regardless of value
    if (inputNumber == 0) {
        output[0] = _adc.read_u16();
        available[0] = 1;
    }
}
```

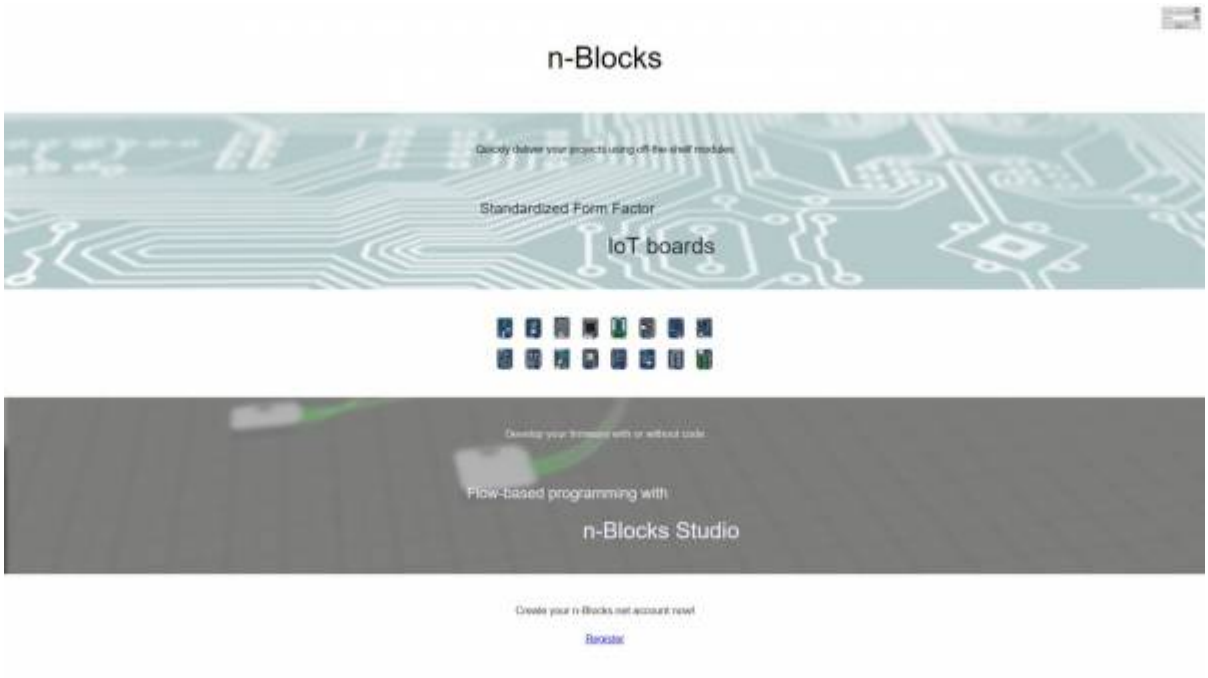
- What is behind the Node-Diagram: Embedded C++ code
- Execution of code is managed by the underlying lightweight nBlocksStudio RTkernel

nBlocksStudio-RTkernel

PERIOD 1mS

- [DATA BETWEEN NODES] Each connection object retrieves data from the source node (PREVIOUS CYCLE OR INITIAL VALUES) and sends them to the destination node
- [INSIDE NODE] The step Method of each node is called:
  - Actual operation is node specific, LIKE:
    - Shifting a FIFO
    - Perform calculations
    - Get from UART hardware buffer and send to a FIFO
    - Read Inputs

n -BlocksStudio Server

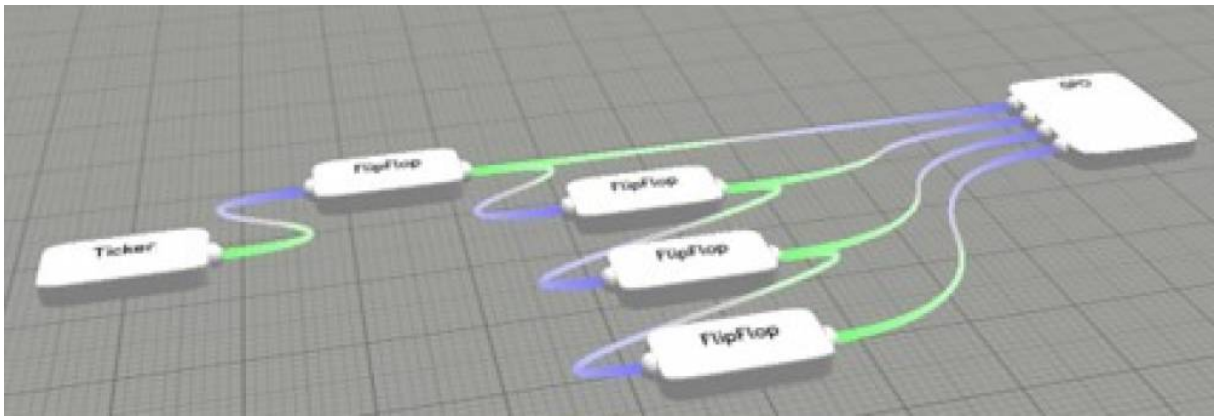


nBlocksStudio Server

- Contribution model and registered users
- Downloadable Nodes
- New Nodes are contributed by users and the n-Blocks team

n -BlocksStudio Server My_nodes

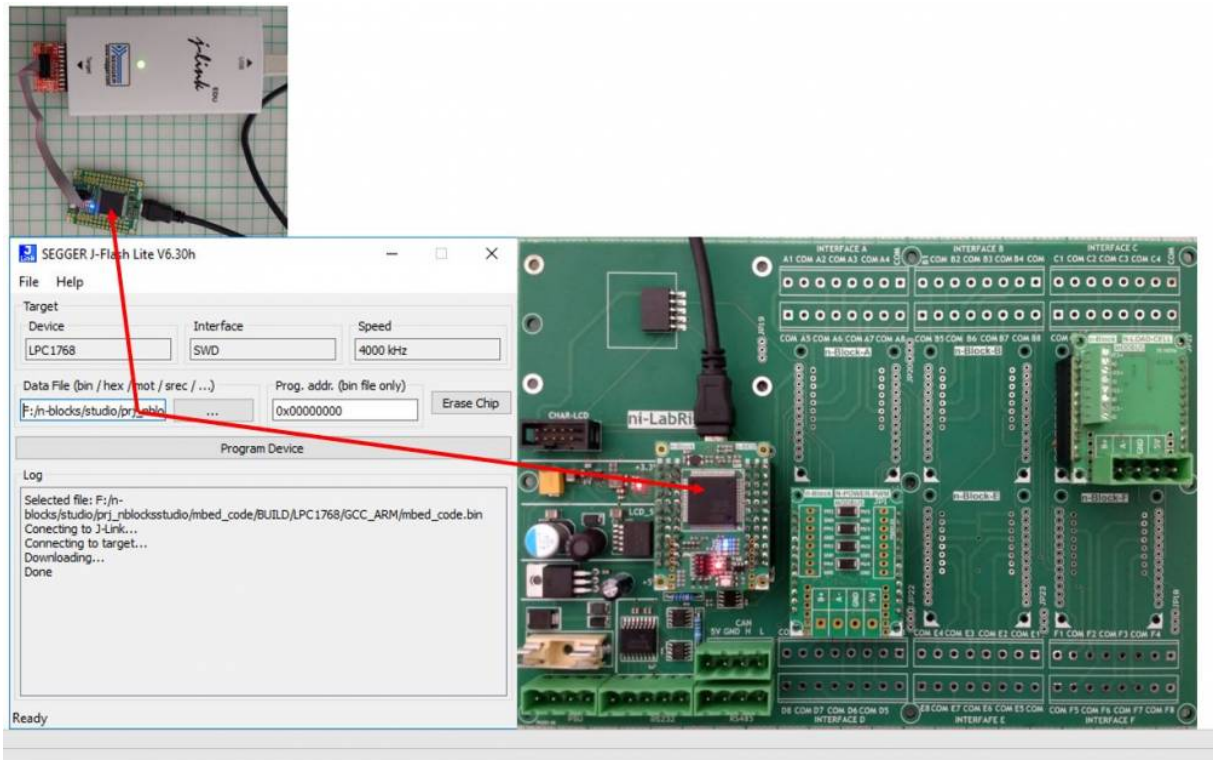
Binary Counter Example created code



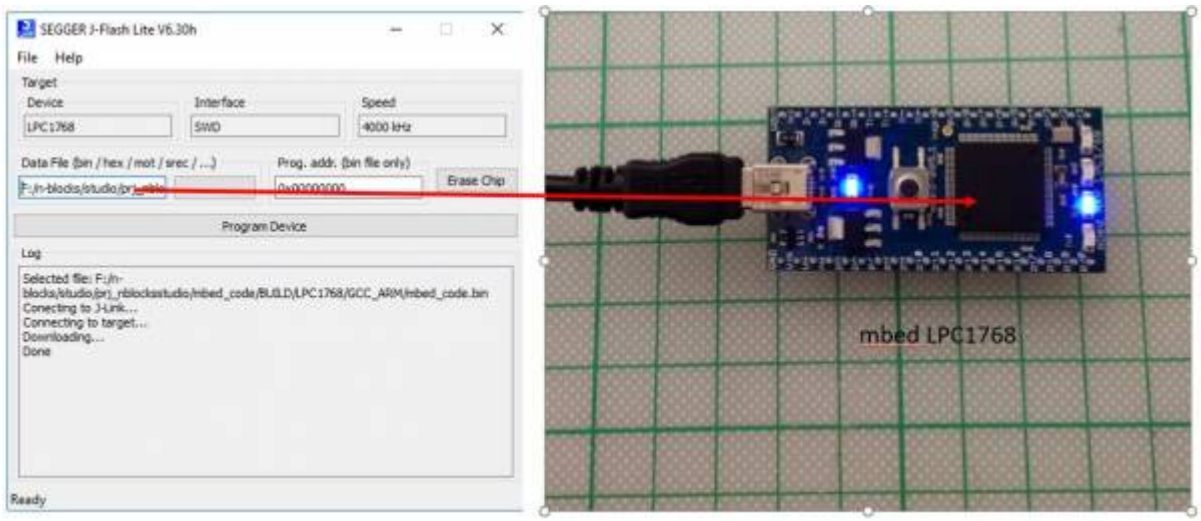
main.cpp

```
/* ================================================================ *
 *       Automatically generated by n-Blocks Studio Designer        *
 *                                                                  *
 *                        www.n-blocks.net                          *
 * ================================================================ */
#include "nblocks.h"


// -*-*- List of node objects -*-*-
nBlock_PWM                  nb_nBlockNode17_PWM;
nBlock_Ticker               nb_nBlockNode0_Ticker;
nBlock_FlipFlop             nb_nBlockNode13_FlipFlop;
nBlock_FlipFlop             nb_nBlockNode3_FlipFlop;
nBlock_GPO                  nb_nBlockNode12_GPO;
nBlock_FlipFlop             nb_nBlockNode14_FlipFlop;
nBlock_ADC                  nb_nBlockNode16_ADC;
nBlock_FlipFlop             nb_nBlockNode15_FlipFlop;

// -*-*- List of connection objects -*-*-
nBlockConnection         n_conn0(&nb_nBlockNode0_Ticker, 0, &nb_nBlockNode3_FlipFlop, 0);
nBlockConnection         n_conn1(&nb_nBlockNode3_FlipFlop, 0, &nb_nBlockNode12_GPO, 0);
nBlockConnection         n_conn2(&nb_nBlockNode3_FlipFlop, 0, &nb_nBlockNode13_FlipFlop, 0);
nBlockConnection         n_conn3(&nb_nBlockNode13_FlipFlop, 0, &nb_nBlockNode12_GPO, 1);
nBlockConnection         n_conn4(&nb_nBlockNode13_FlipFlop, 0, &nb_nBlockNode14_FlipFlop, 0);
nBlockConnection         n_conn5(&nb_nBlockNode14_FlipFlop, 0, &nb_nBlockNode12_GPO, 2);
nBlockConnection         n_conn6(&nb_nBlockNode14_FlipFlop, 0, &nb_nBlockNode15_FlipFlop, 0);
nBlockConnection         n_conn7(&nb_nBlockNode15_FlipFlop, 0, &nb_nBlockNode12_GPO, 3);
nBlockConnection         n_conn8(&nb_nBlockNode16_ADC, 0, &nb_nBlockNode17_PWM, 0);

// -*-*- Main function -*-*-
int main(void) {
    SetupWorkbench();
    while(1) {
        // Your custom code here!
    }
}
```

Binary Counter Example code is running in n-Block



JBinary Counter Example code is running in third party board



Studio [NOT Node] Class Code

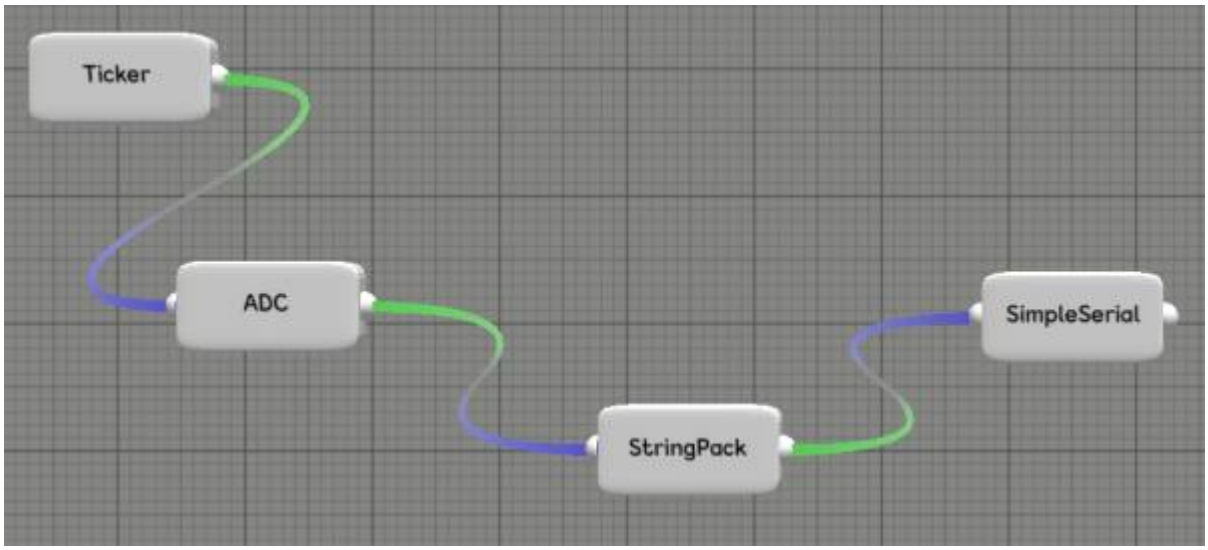
not.h

```
#ifndef __NB_NOT
#define __NB_NOT

#include "mbed.h"
#include "nworkbench.h"


class nBlock_NOT: public nBlockNode {
public:
    nBlock_NOT(void);
    uint32_t outputAvailable(uint32_t outputNumber);
    uint32_t readOutput(uint32_t outputNumber);
    void triggerInput(uint32_t inputNumber, uint32_t value);
    void step(void);
private:
    fifo internal_fifo;
};




#endif

```

not.cpp

```
#include "not.h"


// NOT GATE
nBlock_N
uint32_t nBlock_NOT::outputAvailable(uint32_t outputNumber) { // outputNumber is ignored
    return internal_fifo.available();
}
uint32_t nBlock_NOT::readOutput(uint32_t outputNumber) { // outputNumber is ignored
    uint32_t tmp;
    internal_fifo.read(&tmp);
    return tmp;
}
void nBlock_NOT::triggerInput(uint32_t inputNumber, uint32_t value) { // inputNumber is ignored
    if (value == 0) internal_fifo.put(1);
    else internal_fifo.put(0);
}
void nBlock_NOT::step(void) {
    uint32_t tmp;
    internal_fifo.get(&tmp);
    return;
}
```

ADC to serial port example



This Diagram reads an ADC every 1000ms and sends the data in readable form to UART serial channel

ADC to serial port example Compilation

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

F:\n-blocks\studio\mbed_code>mbed compile -t GCC_ARM -m LPC1768
Building project mbed_code (LPC1768, GCC_ARM)
Scan: .
Scan: env
Scan: mbed
+------------------------------------------------------------------------+-------+-------+------+
| Module                                                                 | .text | .data | .bss |
+------------------------------------------------------------------------+-------+-------+------+
| BUILD\LPC1768\GCC_ARM\adc.o                                            |   352 |     0 |    0 |
| BUILD\LPC1768\GCC_ARM\fifo.o                                           |   108 |     0 |    0 |
| BUILD\LPC1768\GCC_ARM\main.o                                           |   429 |     4 | 1340 |
| BUILD\LPC1768\GCC_ARM\nworkbench.o                                     |   468 |     4 |   84 |
| BUILD\LPC1768\GCC_ARM\simpleserial.o                                   |   120 |     0 |    0 |
| BUILD\LPC1768\GCC_ARM\stringpack.o                                     |   160 |     0 |  256 |
| BUILD\LPC1768\GCC_ARM\ticker.o                                         |   284 |     0 |    0 |
| [fill]                                                                 |   627 |     0 |   13 |
| [lib]/c.a                                                              | 21133 |  2472 |   89 |
| [lib]/gcc.a                                                            |  4132 |     0 |    0 |
| [lib]/mbed.a                                                           |  5873 |     8 |  293 |
| [lib]/misc                                                             |   208 |    12 |   28 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\CRP.o               |     4 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\analogin_api.o      |   503 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\gpio_api.o          |   289 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\mbed_board.o        |   384 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\mbed_retarget.o     |  1895 |   260 |   24 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\mbed_sdk_boot.o     |    80 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\pinmap.o            |   295 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\serial_api.o        |  2518 |     0 |  240 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\startup_LPC17xx.o   |   272 |     0 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\system_LPC17xx.o    |   164 |     4 |    0 |
| mbed\a7c7b631e539\TARGET_LPC1768\TOOLCHAIN_GCC_ARM\us_ticker.o         |   192 |     0 |    1 |
| Subtotals                                                              | 40490 |  2764 | 2368 |
+------------------------------------------------------------------------+-------+-------+------+
Total Static RAM memory (data + bss): 5132 bytes
Total Flash memory (text + data): 43254 bytes

Image: .\BUILD\LPC1768\GCC_ARM\mbed_code.bin

F:\n-blocks\studio\mbed_code>
```

ADC to serial port example main.cpp

main.cpp

```
/* ================================================================ *
 *      Automatically generated by n-Blocks Studio Designer         *
 *                                                                  *
 *                        www.n-blocks.net                          *
 * ================================================================ */
#include "nblocks.h"
#include "stringpack.h"
#include "simpleserial.h"
#include "and.h"
#include "or.h"


// -*-*- List of node objects -*-*-
nBlock_Ticker                nb_nBlockNode0_Ticker(1000);
nBlock_StringPack            nb_nBlockNode2_StringPack("Value: %d\n");
nBlock_ADC                   nb_nBlockNode1_ADC(P0_10);
nBlock_SimpleSerial          nb_nBlockNode3_SimpleSerial(P2_0,P2_1);

// -*-*- List of connection objects -*-*-
nBlockConnection        n_conn0(&nb_nBlockNode0_Ticker, 0, &nb_nBlockNode1_ADC, 0);
nBlockConnection        n_conn1(&nb_nBlockNode1_ADC, 0, &nb_nBlockNode2_StringPack, 0);
nBlockConnection        n_conn2(&nb_nBlockNode2_StringPack, 0, &nb_nBlockNode3_SimpleSerial, 0);

// -*-*- Main function -*-*-
int main(void) {
    SetupWorkbench();
    while(1) {
        // Your custom code here!
    }
}
```

[ADC-Node] C++ microprocessor code

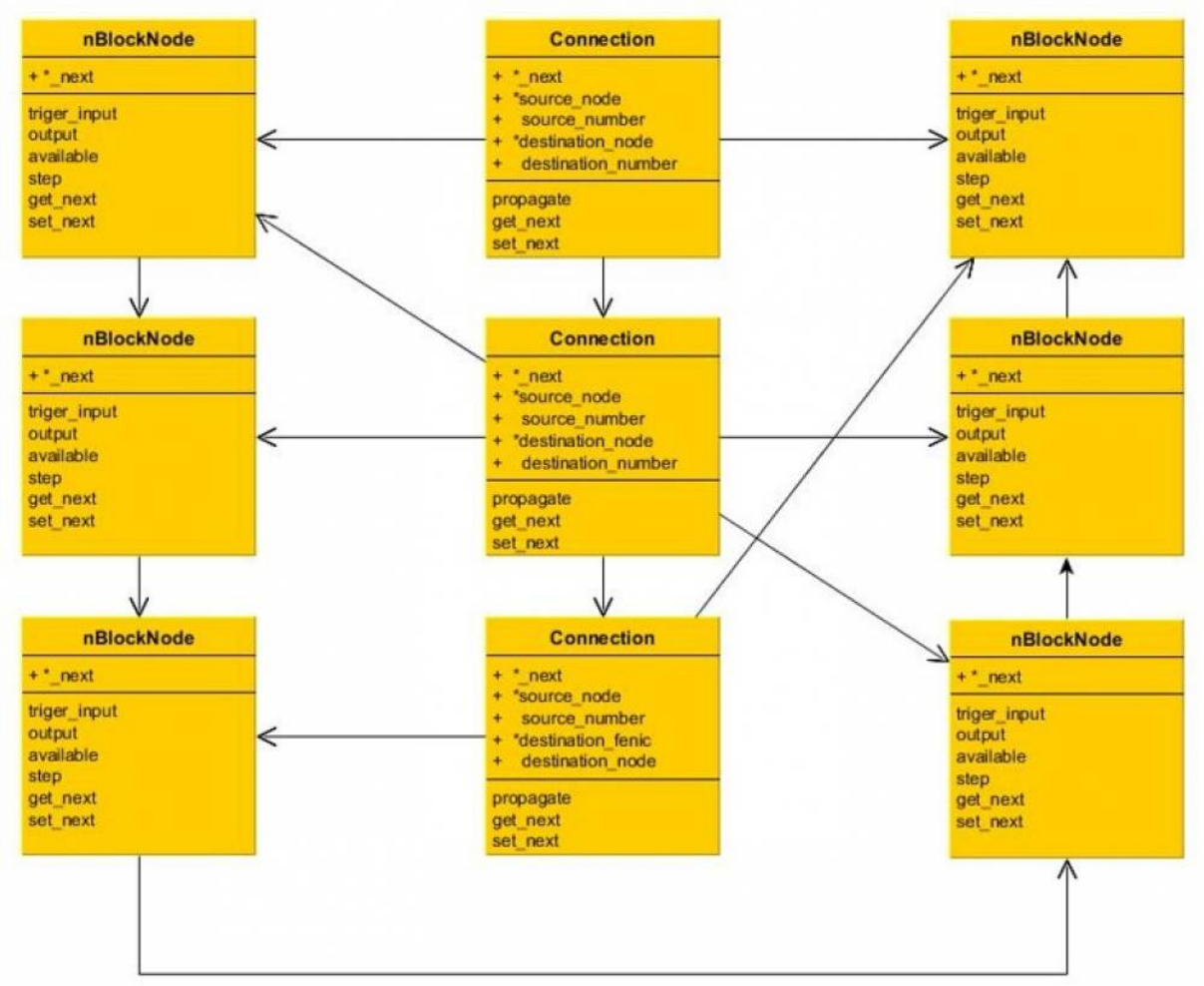
adc.h

```
#ifndef __NB_ADC
#define __NB_ADC

#include "mbed.h"
#include "nworkbench.h"

class nBlock_ADC: public nBlockSimpleNode<1> {
public:
    nBlock_ADC(PinName pinAdc);
    void triggerInput(uint32_t inputNumber, uint32_t value);
private:
    AnalogIn _adc;
};



#endif
```

adc.cpp

```
#include "adc.h"

/// GPI
nBlock_ADC::nBlock_ADC(PinName pinAdc): _adc(pinAdc) {
    return;
}

void nBlock_ADC::triggerInput(uint32_t inputNumber, uint32_t value) {
    // Input 0 triggers a read regardless of value
    if (inputNumber == 0) {
        output[0] = _adc.read_u16();
        available[0] = 1;
    }
}
```

Studio Firmware Classes



IMPORTANT NOTICE - PLEASE READ CAREFULLY

Nimbus Centre reserve the right to make changes, corrections, enhancements, modifications, and improvements to Nimbus Centre products and/or to this document at any time without notice.

All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

Address: Cork Institute of Technology Campus, Bishopstown, Cork

Phone: (021) 433 5560