

Tiny_Hack Guide: XIP Weights on Nicla Vision (STM32H747) with Zig + Arduino

Team tiny_hack

October 1, 2025

Abstract

This guide explains how to (1) build a static library `.a` from a Zig model (e.g. `mobilenet_v2`), (2) package it as an Arduino library, (3) enable QSPI XIP (memory-mapped) on Arduino Nicla Vision (STM32H747), (4) place model weights in a dedicated `.flash_weights` section mapped at `0x9000_0000`, and (5) split and flash internal firmware and external weights via `objcopy` and `dfu-util`. A minimal troubleshooting section is provided at the end.

1 Prerequisites

- Arduino CLI and the Nicla Vision core: `arduino:mbed_nicla` (tested 4.4.1).
- Toolchain from Arduino packages (`arm-none-eabi`).
- `dfu-util` (upload over DFU).
- Zig toolchain (for building the `.a`).

2 Build the static library with Zig

From your Zig project, produce a Cortex-M7 static archive with XIP support:

```
1 zig build lib-gen \  
2   -Dmodel="my_model" \  
3   -Dxip=true \  
4   -Ddynamic \  
5   -Ddo_export \  
6   -Denable_user_tests  
7  
8 zig build lib-test  
9   -Dmodel="my_model" \  
10  -Dxip=true \  
11  -Ddynamic \  
12  -Ddo_export \  
13  -Denable_user_tests  
14  
15 zig build lib \  
16   -Dmodel="my_model" \  
17   -Dtarget=thumb-freestanding \  
18   -Dcpu=cortex_m7 \  
19   -Dxip=true
```

This should generate a `libzant.a` (or similar) under your Zig build output.

3 Package as an Arduino library

Create an Arduino library folder layout under your `Arduino libraries/` directory:

```
1 # Example path: ~/Arduino/libraries/ZantLib
2 mkdir -p ~/Arduino/libraries/ZantLib/src/cortex-m7
```

Copy the produced `.a` into `src/cortex-m7/`. Add a `library.properties` file (you can find it at `examples/tiny_hack/ZantLib/library.properties`) at the library root:

```
1 # library.properties (example)
2 name=Zant
3 version=1.0.0
4 author=AnonymousZanter <AnonymousZanter@zant.com>
5 maintainer=AnonymousZanter <AnonymousZanter@zant.com>
6 sentence= Static Cortex-M7 library.
7 paragraph= Library description
8 category=Uncategorized
9 url=https://zantfoundation.github.io/Website/
10 architectures=mbed_nicla
11 precompiled=full
12 includes=lib_zant.h
13 precompiled=true
14 ldflags=-lzant
```

Your Arduino libraries directory should look like this:

```
1 ~/Arduino/libraries/Zantlib
2     |...library.properties
3     |...src
4     |...cortex-m7/libzant.a
5     |...lib_zant.h
```

Note. If your Arduino header is `lib_zant.h`, make sure it declares the prediction entry point (e.g. `int predict(float*, uint32_t*, uint32_t, float**);`). You can find an example at `examples/tiny_hack/ZantLib/src/lib_zant.h`.

4 Arduino sketch: enable QSPI XIP and call `predict`

Create an Arduino sketch (e.g. `path/tiny_hack/tiny_hack.ino`) that:

1. Exports the symbol `flash_weights_base` at `0x90000000` (required by your Zig lib).
2. Initializes the QUADSPI via HAL.
3. Enables Quad mode (QE in SR2).
4. Enters memory-mapped (XIP) using Fast Read Quad Output (`0x6B / 1-1-4 / 8 dummy`).
5. Prepares a `[1,3,32,32]` NCHW input and calls `predict()`.

The following code can be also found at `Zant/examples/tiny_hack/tiny_hack.ino`.

```
1 /***** Nicla Vision QSPI XIP (Memory-Mapped) + predict *****/
2 * Core: arduino:mbed_nicla (4.4.1)
3 * - HAL QSPI (QUADSPI)
4 * - Pin: CLK=PB2, CS=PG6, IO0..3=PD11..PD14 (AF QUADSPI)
5 * - QE su SR2, poi READ 0x6B 1-1-4 con 8 dummy
6 *****/
7
```

```

8 extern "C" {
9     #ifndef STM32H747xx
10    #define STM32H747xx
11    #endif
12    #ifndef HAL_QSPI_MODULE_ENABLED
13    #define HAL_QSPI_MODULE_ENABLED
14    #endif
15    #include "stm32h7xx_hal.h"
16    #include "stm32h7xx_hal_qspi.h"
17 }
18
19 // Required by the Zig library:
20 extern "C" __attribute__((used))
21 const uint8_t* flash_weights_base = (const uint8_t*)0x90000000u;
22
23 #include <Arduino.h>
24 #include <lib_zant.h> // int predict(float*, uint32_t*, uint32_t, float**)
25
26 static QSPI_HandleTypeDef hqspi;
27
28 static const uint8_t CMD_RDID = 0x9F, CMD_WREN = 0x06;
29 static const uint8_t CMD_RDSR1= 0x05, CMD_RDSR2= 0x35, CMD_WRSR = 0x01;
30 static const uint8_t CMD_READ_Q0 = 0x6B;
31
32 // MSP init (GPIO+clock)
33 extern "C" void HAL_QSPI_MspInit(QSPI_HandleTypeDef* h){
34     if(h->Instance != QUADSPI) return;
35     __HAL_RCC_GPIOB_CLK_ENABLE();
36     __HAL_RCC_GPIOD_CLK_ENABLE();
37     __HAL_RCC_GPIOG_CLK_ENABLE();
38     __HAL_RCC_QSPI_CLK_ENABLE();
39
40     GPIO_InitTypeDef GPIO = {0};
41     // CLK PB2 (AF9)
42     GPIO.Pin = GPIO_PIN_2; GPIO.Mode=GPIO_MODE_AF_PP; GPIO.Pull=GPIO_NOPULL;
43     GPIO.Speed=GPIO_SPEED_FREQ_VERY_HIGH; GPIO.Alternate=GPIO_AF9_QUADSPI;
44     HAL_GPIO_Init(GPIOB, &GPIO);
45     // CS PG6 (AF10)
46     GPIO.Pin = GPIO_PIN_6; GPIO.Alternate=GPIO_AF10_QUADSPI;
47     HAL_GPIO_Init(GPIOG, &GPIO);
48     // IO0..IO3 PD11..PD14 (AF9)
49     GPIO.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14;
50     GPIO.Alternate=GPIO_AF9_QUADSPI;
51     HAL_GPIO_Init(GPIOD, &GPIO);
52 }
53
54 static HAL_StatusTypeDef qspi_init_16mb(QSPI_HandleTypeDef* h){
55     h->Instance = QUADSPI;
56     h->Init.ClockPrescaler = 7;
57     h->Init.FifoThreshold = 4;
58     h->Init.SampleShifting = QSPI_SAMPLE_SHIFTING_NONE;
59     h->Init.FlashSize = 23; // 2^24 = 16MB -> set 23
60     h->Init.ChipSelectHighTime = QSPI_CS_HIGH_TIME_2_CYCLE;
61     h->Init.ClockMode = QSPI_CLOCK_MODE_0;
62     h->Init.FlashID = QSPI_FLASH_ID_1;
63     h->Init.DualFlash = QSPI_DUALFLASH_DISABLE;
64     return HAL_QSPI_Init(h);
65 }
66
67 static HAL_StatusTypeDef qspi_cmd(QSPI_HandleTypeDef* h, uint8_t inst,

```

```

68         uint32_t addrMode, uint32_t dataMode,
69         uint32_t addr, uint32_t dummy,
70         uint8_t* data, size_t len, bool rx){
71     QSPI_CommandTypeDef c = {0};
72     c.InstructionMode = QSPI_INSTRUCTION_1_LINE;
73     c.Instruction = inst;
74     c.AddressMode = addrMode;
75     c.Address = addr;
76     c.AddressSize = QSPI_ADDRESS_24_BITS;
77     c.DataMode = dataMode;
78     c.NbData = len;
79     c.DummyCycles = dummy;
80     if(HAL_QSPI_Command(h, &c, HAL_MAX_DELAY) != HAL_OK) return HAL_ERROR;
81     if(len == 0) return HAL_OK;
82     return rx ? HAL_QSPI_Receive(h, data, HAL_MAX_DELAY)
83             : HAL_QSPI_Transmit(h, data, HAL_MAX_DELAY);
84 }
85
86 static HAL_StatusTypeDef rd_sr(QSPI_HandleTypeDef* h, uint8_t cmd, uint8_t* val){
87     return qspi_cmd(h, cmd, QSPI_ADDRESS_NONE, QSPI_DATA_1_LINE, 0, 0, val, 1, true);
88 }
89 static HAL_StatusTypeDef wren(QSPI_HandleTypeDef* h){
90     return qspi_cmd(h, CMD_WREN, QSPI_ADDRESS_NONE, QSPI_DATA_NONE, 0, 0, nullptr, 0, true);
91 }
92 static HAL_StatusTypeDef wr_sr12(QSPI_HandleTypeDef* h, uint8_t sr1, uint8_t sr2){
93     uint8_t buf[2] = {sr1, sr2};
94     return qspi_cmd(h, CMD_WRSR, QSPI_ADDRESS_NONE, QSPI_DATA_1_LINE, 0, 0, buf, 2, false);
95 }
96 static HAL_StatusTypeDef wait_wip_clear(QSPI_HandleTypeDef* h, uint32_t timeout_ms){
97     uint32_t t0 = millis();
98     for(;;){
99         uint8_t sr1=0; if(rd_sr(h, CMD_RDSR1, &sr1) != HAL_OK) return HAL_ERROR;
100         if((sr1 & 0x01) == 0) return HAL_OK;
101         if((millis()-t0) > timeout_ms) return HAL_TIMEOUT;
102         delay(1);
103     }
104 }
105 static HAL_StatusTypeDef enable_quad(QSPI_HandleTypeDef* h){
106     uint8_t sr1=0, sr2=0;
107     if(rd_sr(h, CMD_RDSR1, &sr1) != HAL_OK) return HAL_ERROR;
108     if(rd_sr(h, CMD_RDSR2, &sr2) != HAL_OK) return HAL_ERROR;
109     if(sr2 & 0x02) return HAL_OK; // QE already 1
110     if(wren(h) != HAL_OK) return HAL_ERROR;
111     sr2 |= 0x02;
112     if(wr_sr12(h, sr1, sr2) != HAL_OK) return HAL_ERROR;
113     if(wait_wip_clear(h, 500) != HAL_OK) return HAL_ERROR;
114     if(rd_sr(h, CMD_RDSR2, &sr2) != HAL_OK) return HAL_ERROR;
115     return (sr2 & 0x02) ? HAL_OK : HAL_ERROR;
116 }
117
118 static HAL_StatusTypeDef qspi_enter_mmap(QSPI_HandleTypeDef* h){
119     QSPI_CommandTypeDef c = {0};
120     c.InstructionMode = QSPI_INSTRUCTION_1_LINE;
121     c.Instruction = CMD_READ_QO; // 0x6B
122     c.AddressMode = QSPI_ADDRESS_1_LINE;
123     c.AddressSize = QSPI_ADDRESS_24_BITS;
124     c.Address = 0x000000;
125     c.AlternateByteMode = QSPI_ALTERNATE_BYTES_NONE;
126     c.DataMode = QSPI_DATA_4_LINES;
127     c.DummyCycles = 8;

```

```

128 #ifdef QSPI_DDR_MODE_DISABLE
129 c.DdrMode = QSPI_DDR_MODE_DISABLE;
130 c.DdrHoldHalfCycle = QSPI_DDR_HHC_ANALOG_DELAY;
131 #endif
132 #ifdef QSPI_SIOO_INST_EVERY_CMD
133 c.SIOOMode = QSPI_SIOO_INST_EVERY_CMD;
134 #endif
135 QSPI_MemoryMappedTypeDef mm = {0};
136 mm.TimeoutActivation = QSPI_TIMEOUT_COUNTER_DISABLE;
137 mm.TimeoutPeriod = 0;
138 return HAL_QSPI_MemoryMapped(h, &c, &mm);
139 }
140
141 // ---- Predict demo ----
142 #ifndef ZANT_OUTPUT_LEN
143 #define ZANT_OUTPUT_LEN 4
144 #endif
145 static const int OUT_LEN = ZANT_OUTPUT_LEN;
146 static const uint32_t IN_N=1, IN_C=3, IN_H=32, IN_W=32;
147 static const uint32_t IN_SIZE = IN_N*IN_C*IN_H*IN_W;
148 static float inputData[IN_SIZE];
149 static uint32_t inputShape[4] = {IN_N, IN_C, IN_H, IN_W};
150
151 static void printOutput(const float* out, int len){
152     if(!out || len<=0){ Serial.println("Output nullo"); return; }
153     Serial.println("=== Output ===");
154     for(int i=0;i<len;++i){
155         Serial.print("out["); Serial.print(i);
156         Serial.print("] = "); Serial.println(out[i], 6);
157     }
158     Serial.println("=====");
159 }
160
161 void setup(){
162     Serial.begin(115200);
163     uint32_t t0 = millis(); while(!Serial && (millis()-t0)<4000) delay(10);
164     Serial.println("\n== Nicla Vision QSPI XIP (HAL) + predict ==");
165
166     if(qspi_init_16mb(&hqspi) != HAL_OK){ Serial.println("QSPI init FAIL"); for(;;){} }
167     if(enable_quad(&hqspi) != HAL_OK){ Serial.println("Enable QE FAIL"); for(;;){} }
168     if(qspi_enter_mmap(&hqspi) != HAL_OK){ Serial.println("XIP FAIL"); for(;;){} }
169
170     // Prepare NCHW input (simple constant pattern per channel)
171     for(uint32_t c=0;c<IN_C;++c)
172         for(uint32_t h=0;h<IN_H;++h)
173             for(uint32_t w=0;w<IN_W;++w){
174                 uint32_t idx = c*(IN_H*IN_W) + h*IN_W + w;
175                 inputData[idx] = (c==0) ? 0.8f : (c==1 ? 0.5f : 0.2f);
176             }
177
178     float* out=NULLPTR;
179     Serial.println("[Predict] Calling predict()...");
180     unsigned long t_us0 = micros();
181     int rc = predict(inputData, inputShape, 4, &out);
182     unsigned long t_us1 = micros();
183
184     Serial.print("[Predict] rc="); Serial.println(rc);
185     Serial.print("[Predict] us="); Serial.println((unsigned long)(t_us1-t_us0));
186     if(rc==0 && out){ printOutput(out, OUT_LEN); }
187     else { Serial.println("[Predict] FAIL"); }

```

```

188 }
189
190 void loop(){ delay(500); }

```

5 Custom linker: map `.flash_weights` to QSPI

Provide a minimal linker script (e.g. `Zant/examples/tiny_hack/custom.ld`) that captures your weights into a dedicated section at the QSPI XIP base:

```

1  /* Place .flash_weights into external QSPI (XIP) */
2  MEMORY
3  {
4      QSPI (rx) : ORIGIN = 0x90000000, LENGTH = 16M
5  }
6
7  SECTIONS
8  {
9      .flash_weights :
10     {
11         . = ALIGN(32);
12         __flash_weights_start__ = .;
13         KEEP(*(.flash_weights))
14         KEEP(*(.flash_weights.*))
15         KEEP(*(.rodata.flash_weights*))
16         . = ALIGN(32);
17         __flash_weights_end__ = .;
18     } > QSPI
19 }
20
21 PROVIDE(__flash_weights_size__ =
22     __flash_weights_end__ - __flash_weights_start__);

```

Important. Your Zig build must emit weights into a section named `.flash_weights` (or adjust names consistently in both the codegen and this script). The Arduino sketch exports `flash_weights_base` at `0x90000000` for the Zig library to reference. So ensure that the flag `-Dxip` is set to true when codegenerating the library.

6 Compile with Arduino CLI using the custom linker

From the sketch folder containing `tiny_hack.ino` and `custom.ld`:

```

1  cd my_path/tiny_hack
2  FQBN="arduino:mbed_nicla:nicla_vision"
3
4  arduino-cli compile \
5      --fqbn "$FQBN" \
6      --export-binaries \
7      --libraries ~/Arduino/libraries \
8      --build-property "compiler.c.elf.extra_flags=-Wl,-T$PWD/custom.ld"

```

7 Split firmware and weights & flash with DFU

Let ELF be the path of the compiled ELF: The following procedure is reported inside `Zant/examples/tiny_hack/flash_nicla_xip.sh` for an easier deployment. To flash the board connect it to your PC and follow the instructions after launching `./flash_nicla_xip.sh`

```

1 ARD="$HOME/.arduino15/packages/arduino/tools/arm-none-eabi-gcc/7-2017q4/bin"
2 READELF="$ARD/arm-none-eabi-readelf"
3 OBJCOPY="$ARD/arm-none-eabi-objcopy"
4
5 ELF=./build/arduino.mbed_nicla.nicla_vision/tiny_hack.ino.elf
6
7 # Inspect sections
8 "$READELF" -S "$ELF" | egrep -n "\.flash_weights|\.text|Name"
9
10 # Program for internal flash (strip .flash_weights)
11 "$OBJCOPY" -O binary -R .flash_weights "$ELF" nicla_internal.bin
12
13 # Weights only
14 "$OBJCOPY" -O binary -j .flash_weights "$ELF" nicla_weights.bin
15 ls -lh nicla_*.bin

```

Put Nicla Vision into DFU (double-tap RESET button), then:

```

1 dfu-util -l
2 # Expect:
3 # alt=0 "@Internal Flash /0x08000000/01*128Ka,15*128Kg"
4 # alt=1 "@External Flash /0x90000000/4096*4Kg"
5
6 # Flash internal firmware (note: sketch starts at 0x08040000)
7 dfu-util -a 0 -s 0x08040000:leave -D nicla_internal.bin
8
9 # Flash weights to QSPI (XIP base)
10 dfu-util -a 1 -s 0x90000000:leave -D nicla_weights.bin

```

8 Verify at runtime

Open Serial at 115200 baud (suggested `$ screen /dev/ttyACM0 115200`). A typical successful run shows:

- QSPI init OK; JEDEC ID printed.
- “Enter Memory-Mapped... OK”
- Optional dump of first bytes at `0x90000000`.
- [Predict] `rc=0` and reasonable timing.

9 Troubleshooting

HardFault right after “reading 0x90000000” Make sure you called `HAL_QSPI_MemoryMapped` with a proper command for 1-1-4 fast read (0x6B) and enabled QE in SR2. Verify that external flash actually contains weights (dump a few bytes).

undefined reference to ‘flash_weights_base’ Ensure your sketch defines
`extern "C" const uint8_t* flash_weights_base = (const uint8_t*)0x90000000u;`
 Compile order also matters; keep that definition at the top-level of the sketch.

class UART has no member ‘printf’ Use `Serial.print/println` instead of `printf` on Arduino.

OSPI vs QSPI types Use `HAL_QSPI_*` (QUADSPI) for Nicla Vision’s H747 HAL in this core.

Sketch too big You must remove weights from the internal binary and place them in `.flash_weights` in QSPI as shown; otherwise internal `.text` will overflow.

No DFU alt=1 Update bootloader/core; on Nicla Vision you should see alt=1 mapping `0x90000000`.
Replug in DFU mode if needed.

Credits

Thanks to the tiny_hack team and contributors. Adapt the scripts/addresses if your core or bootloader differs.