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

Team tiny_hack

October 4, 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 Zant project (so inside /Z-Ant), produce a Cortex-M7 static library with XIP support:

```
zig build lib-gen \
    -Dmodel="my_model" \
    -Dxip=true \
    -Ddo_export \
    -Denable_user_tests

zig build lib-test
    -Dmodel="my_model" \
    -Dxip=true \
    -Denable_user_tests

zig build lib \
  -Dmodel="my_model"\
  -Dtarget=thumb-freestanding \
  -Dcpu=cortex_m7 \
```

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:

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

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

```
# library.properties (example)
name=Zant
version=1.0.0
author=AnonymousZanter <AnonymousZanter@zant.com>
maintainer=AnonymousZanter <AnonymousZanter@zant.com>
sentence= Static Cortex-M7 library.
paragraph= Library description
category=Uncategorized
url=https://zantfoundation.github.io/Website/
architectures=mbed_nicla
precompiled=full
includes=lib_zant.h
precompiled=true
ldflags=-lzant
```

Your Arduino libraries directory should look like this:

```
~/Arduino/libraries/Zantlib
                      |...library.properties
                      |...src
                          |...cortex-m7/libzant.a
                          |...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 and 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` .

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

extern "C" {
  #ifndef STM32H747xx
  #define STM32H747xx
  #endif
  #ifndef HAL_QSPI_MODULE_ENABLED
  #define HAL_QSPI_MODULE_ENABLED
  #endif
```

```
  #include "stm32h7xx_hal.h"
  #include "stm32h7xx_hal_qspi.h"
}

// Required by the Zig library:
extern "C" __attribute__((used))
const uint8_t* flash_weights_base = (const uint8_t*)0x90000000u;

#include <Arduino.h>
#include <lib_zant.h> // int predict(float*, uint32_t*, uint32_t, float**)

static QSPI_HandleTypeDef hqspi;

static const uint8_t CMD_RDID = 0x9F, CMD_WREN = 0x06;
static const uint8_t CMD_RDSR1= 0x05, CMD_RDSR2= 0x35, CMD_WRSR = 0x01;
static const uint8_t CMD_READ_QO = 0x6B;

// MSP init (GPIO+clock)
extern "C" void HAL_QSPI_MspInit(QSPI_HandleTypeDef* h){
  if(h->Instance != QUADSPI) return;
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_QSPI_CLK_ENABLE();

  GPIO_InitTypeDef GPIO = {0};
  // CLK PB2 (AF9)
  GPIO.Pin = GPIO_PIN_2; GPIO.Mode=GPIO_MODE_AF_PP; GPIO.Pull=GPIO_NOPULL;
  GPIO.Speed=GPIO_SPEED_FREQ_VERY_HIGH; GPIO.Alternate=GPIO_AF9_QUADSPI;
  HAL_GPIO_Init(GPIOB, &GPIO);
  // CS PG6 (AF10)
  GPIO.Pin = GPIO_PIN_6; GPIO.Alternate=GPIO_AF10_QUADSPI;
  HAL_GPIO_Init(GPIOG, &GPIO);
  // IO0..IO3 PD11..PD14 (AF9)
  GPIO.Pin = GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14;
  GPIO.Alternate=GPIO_AF9_QUADSPI;
  HAL_GPIO_Init(GPIOD, &GPIO);
}

static HAL_StatusTypeDef qspi_init_16mb(QSPI_HandleTypeDef* h){
  h->Instance = QUADSPI;
  h->Init.ClockPrescaler = 7;
  h->Init.FifoThreshold = 4;
  h->Init.SampleShifting = QSPI_SAMPLE_SHIFTING_NONE;
  h->Init.FlashSize = 23; // 2^24 = 16MB -> set 23
  h->Init.ChipSelectHighTime = QSPI_CS_HIGH_TIME_2_CYCLE;
  h->Init.ClockMode = QSPI_CLOCK_MODE_0;
  h->Init.FlashID = QSPI_FLASH_ID_1;
  h->Init.DualFlash = QSPI_DUALFLASH_DISABLE;
  return HAL_QSPI_Init(h);
}

static HAL_StatusTypeDef qspi_cmd(QSPI_HandleTypeDef* h, uint8_t inst,
                                  uint32_t addrMode, uint32_t dataMode,
                                  uint32_t addr, uint32_t dummy,
                                  uint8_t* data, size_t len, bool rx){
  QSPI_CommandTypeDef c = {0};
  c.InstructionMode = QSPI_INSTRUCTION_1_LINE;
  c.Instruction = inst;
  c.AddressMode = addrMode;
```

```
  c.Address = addr;
  c.AddressSize = QSPI_ADDRESS_24_BITS;
  c.DataMode = dataMode;
  c.NbData = len;
  c.DummyCycles = dummy;
  if(HAL_QSPI_Command(h, &c, HAL_MAX_DELAY) != HAL_OK) return HAL_ERROR;
  if(len == 0) return HAL_OK;
  return rx ? HAL_QSPI_Receive(h, data, HAL_MAX_DELAY)
            : HAL_QSPI_Transmit(h, data, HAL_MAX_DELAY);
}

static HAL_StatusTypeDef rd_sr(QSPI_HandleTypeDef* h, uint8_t cmd, uint8_t* val){
  return qspi_cmd(h, cmd, QSPI_ADDRESS_NONE, QSPI_DATA_1_LINE, 0, 0, val, 1, true);
}
static HAL_StatusTypeDef wren(QSPI_HandleTypeDef* h){
  return qspi_cmd(h, CMD_WREN, QSPI_ADDRESS_NONE, QSPI_DATA_NONE, 0, 0, nullptr, 0, true);
}
static HAL_StatusTypeDef wr_sr12(QSPI_HandleTypeDef* h, uint8_t sr1, uint8_t sr2){
  uint8_t buf[2] = {sr1, sr2};
  return qspi_cmd(h, CMD_WRSR, QSPI_ADDRESS_NONE, QSPI_DATA_1_LINE, 0, 0, buf, 2, false);
}
static HAL_StatusTypeDef wait_wip_clear(QSPI_HandleTypeDef* h, uint32_t timeout_ms){
  uint32_t t0 = millis();
  for(;;){
    uint8_t sr1=0; if(rd_sr(h, CMD_RDSR1, &sr1) != HAL_OK) return HAL_ERROR;
    if((sr1 & 0x01) == 0) return HAL_OK;
    if((millis()-t0) > timeout_ms) return HAL_TIMEOUT;
    delay(1);
  }
}
static HAL_StatusTypeDef enable_quad(QSPI_HandleTypeDef* h){
  uint8_t sr1=0, sr2=0;
  if(rd_sr(h, CMD_RDSR1, &sr1) != HAL_OK) return HAL_ERROR;
  if(rd_sr(h, CMD_RDSR2, &sr2) != HAL_OK) return HAL_ERROR;
  if(sr2 & 0x02) return HAL_OK; // QE already 1
  if(wren(h) != HAL_OK) return HAL_ERROR;
  sr2 |= 0x02;
  if(wr_sr12(h, sr1, sr2) != HAL_OK) return HAL_ERROR;
  if(wait_wip_clear(h, 500) != HAL_OK) return HAL_ERROR;
  if(rd_sr(h, CMD_RDSR2, &sr2) != HAL_OK) return HAL_ERROR;
  return (sr2 & 0x02) ? HAL_OK : HAL_ERROR;
}

static HAL_StatusTypeDef qspi_enter_mmap(QSPI_HandleTypeDef* h){
  QSPI_CommandTypeDef c = {0};
  c.InstructionMode = QSPI_INSTRUCTION_1_LINE;
  c.Instruction = CMD_READ_QO; // 0x6B
  c.AddressMode = QSPI_ADDRESS_1_LINE;
  c.AddressSize = QSPI_ADDRESS_24_BITS;
  c.Address = 0x000000;
  c.AlternateByteMode = QSPI_ALTERNATE_BYTES_NONE;
  c.DataMode = QSPI_DATA_4_LINES;
  c.DummyCycles = 8;
  #ifdef QSPI_DDR_MODE_DISABLE
  c.DdrMode = QSPI_DDR_MODE_DISABLE;
  c.DdrHoldHalfCycle = QSPI_DDR_HHC_ANALOG_DELAY;
  #endif
  #ifdef QSPI_SIOO_INST_EVERY_CMD
  c.SIOOMode = QSPI_SIOO_INST_EVERY_CMD;
  #endif
```

```
  QSPI_MemoryMappedTypeDef mm = {0};
  mm.TimeOutActivation = QSPI_TIMEOUT_COUNTER_DISABLE;
  mm.TimeOutPeriod = 0;
  return HAL_QSPI_MemoryMapped(h, &c, &mm);
}

// ---- Predict demo ----
#ifndef ZANT_OUTPUT_LEN
#define ZANT_OUTPUT_LEN 4
#endif
static const int OUT_LEN = ZANT_OUTPUT_LEN;
static const uint32_t IN_N=1, IN_C=3, IN_H=32, IN_W=32;
static const uint32_t IN_SIZE = IN_N*IN_C*IN_H*IN_W;
static float inputData[IN_SIZE];
static uint32_t inputShape[4] = {IN_N, IN_C, IN_H, IN_W};

static void printOutput(const float* out, int len){
  if(!out || len<=0){ Serial.println("Output nullo"); return; }
  Serial.println("=== Output ===");
  for(int i=0;i<len;++i){
    Serial.print("out["); Serial.print(i);
    Serial.print("] = "); Serial.println(out[i], 6);
  }
  Serial.println("==============");
}

void setup(){
  Serial.begin(115200);
  uint32_t t0 = millis(); while(!Serial && (millis()-t0)<4000) delay(10);
  Serial.println("\n== Nicla Vision  QSPI XIP (HAL) + predict ==");

  if(qspi_init_16mb(&hqspi) != HAL_OK){ Serial.println("QSPI init FAIL"); for(;;){} }
  if(enable_quad(&hqspi) != HAL_OK){ Serial.println("Enable QE FAIL"); for(;;){} }
  if(qspi_enter_mmap(&hqspi) != HAL_OK){ Serial.println("XIP FAIL"); for(;;){} }

  // Prepare NCHW input (simple constant pattern per channel)
  for(uint32_t c=0;c<IN_C;++c)
    for(uint32_t h=0;h<IN_H;++h)
      for(uint32_t w=0;w<IN_W;++w){
        uint32_t idx = c*(IN_H*IN_W) + h*IN_W + w;
        inputData[idx] = (c==0) ? 0.8f : (c==1 ? 0.5f : 0.2f);
      }

  float* out=nullptr;
  Serial.println("[Predict] Calling predict()...");
  unsigned long t_us0 = micros();
  int rc = predict(inputData, inputShape, 4, &out);
  unsigned long t_us1 = micros();

  Serial.print("[Predict] rc="); Serial.println(rc);
  Serial.print("[Predict] us="); Serial.println((unsigned long)(t_us1-t_us0));
  if(rc==0 && out){ printOutput(out, OUT_LEN); }
  else { Serial.println("[Predict] FAIL"); }
}

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:

```
/* Place .flash_weights into external QSPI (XIP) */
MEMORY
{
  QSPI (rx) : ORIGIN = 0x90000000, LENGTH = 16M
}

SECTIONS
{
  .flash_weights :
  {
    . = ALIGN(32);
    __flash_weights_start__ = .;
    KEEP(*(.flash_weights))
    KEEP(*(.flash_weights.*))
    KEEP(*(.rodata.flash_weights*))
    . = ALIGN(32);
    __flash_weights_end__ = .;
  } > QSPI
}

PROVIDE(__flash_weights_size__ =
  __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`:

```
cd my_path/tiny_hack
FQBN="arduino:mbed_nicla:nicla_vision"

arduino-cli compile \
  --fqbn "$FQBN" \
  --export-binaries \
  --libraries ~/Arduino/libraries \
  --build-property "compiler.c.elf.extra_flags=-Wl,-T$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`

```
ARD="$HOME/.arduino15/packages/arduino/tools/arm-none-eabi-gcc/7-2017q4/bin"
READELF="$ARD/arm-none-eabi-readelf"
OBJCOPY="$ARD/arm-none-eabi-objcopy"

ELF=./build/arduino.mbed_nicla.nicla_vision/tiny_hack.ino.elf
```

```
# Inspect sections
"$READELF" -S "$ELF" | egrep -n "\.flash_weights|\.text|Name"

# Program for internal flash (strip .flash_weights)
"$OBJCOPY" -O binary -R .flash_weights "$ELF" nicla_internal.bin

# Weights only
"$OBJCOPY" -O binary -j .flash_weights "$ELF" nicla_weights.bin
ls -lh nicla_*.bin
```

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

```
dfu-util -l
# Expect:
# alt=0 "@Internal Flash /0x08000000/01*128Ka,15*128Kg"
# alt=1 "@External Flash /0x90000000/4096*4Kg"

# Flash internal firmware (note: sketch starts at 0x08040000)
dfu-util -a 0 -s 0x08040000:leave -D nicla_internal.bin

# Flash weights to QSPI (XIP base)
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.