Intel® Curie™ Manufacturing Process Example
About this document

Purpose
This document describes the main steps of the manufacturing flow used for the Arduino* 101 boards. It can be used as a reference guide to follow when setting up the manufacturing flow for a board including an Intel® Curie™ module.

Scope
This document is for makers of printed circuit boards that include an Intel Curie module. The process described is strictly specific to Arduino 101, so customers will need to evaluate the steps based on their board design and make changes as required.

Overview
This document presents the main steps of board manufacturing for the Arduino 101: from assembling the components on the printed circuit board, to testing the board and flashing it with a custom image.
Main Phases of the Manufacturing Process

The following diagram provides an overview of the manufacturing process for an Intel® Curie™
module-based printed circuit board. The process illustrated is applicable generally, and the rest of
this chapter describes the process in more detail, as it was followed for the Arduino® 101 board:
Phase 1 – Making the Printed Circuit Board (PCB)

Schematics diagrams

Parts list – Bill of material (BOM)

Board type and dimensions

PCB layout

- Placing the components
- Placing the power and ground traces
- Placing the signal traces

Phase 2 – Assembling the Printed Circuit Board

Components are assembled to the boards using SMT technology. The I/O pins and connectors are assembled using through-hole process (this is the technology used for Arduino 101 product specifically).

Surface Mount Technology (SMT)
In the SMT process, the solder paste is deposited onto the connection pads of the printed circuit board, components are placed onto these pads, reflowed in a reflow furnace to melt the solder paste to form the electrical and mechanical connection of the component leads to the printed circuit board.

Through-hole assembly process
The component leads are placed onto the board and the leads are soldered.

Compliance with RoHS
Customers are recommended to go through their own RoHS compliance process for the targeted geography and market segment.

Phase 3 - Testing the Assembled Printed Board

The Arduino 101 boards are subject to triple testing to make sure they are not faulty.

- First, the printed circuit boards are thoroughly tested for short circuits and open connections
- Second, the boards are powered and the bootloader is programmed, to verify that there are no macroscopic problems preventing the board from turning on
- Last, the board is placed into a custom bed of nails and programmed by a test program which tests its overall functionality.
Factory testing of the Arduino* 101 board

This section translates the scripts used for the Arduino 101 production test into readable testing instructions. The scripts can be adapted for a customized board including the Intel Curie module. The factory testing includes four stages:

- Factory test configuration
- BLE testing
- User configuration setup
- Functional testing

Stage 1 – Factory test configuration

Main tasks of this Factory Test Configuration are as follows:

1. Flash the device under test (DUT) with the bootloader image.
2. Flash the x86 core with the factory test image.
3. Flash the Nordic core with its image.

Test fixture

The following diagrams show the construct of the Stage 1 test fixture:
Notes:

- The USB Powered relay shown in the test fixture is removed from the final design of the fixture.
- The USB hub (white box) in the test fixture is also eliminated in the final design. Both the DUT and the Flyswatter USB interfaces are connected directly to the test control computer.
The following is a brief description of the test sequence used for the Arduino 101 board for Stage 1:

1. Operator mounts DUT onto the test fixture and secures it by closing the compress cover. Please refer to the text fixture pictures.
2. Test equipment setup and configuration.
3. Test controller verifies the DUT SKU is an Arduino 101 board. Failure terminates test sequence.
4. The DUT is flashed with a proprietary factory test code that runs on the x86 core. The sole purpose of this test code is to perform two main tests in Stage 2 testing: External SPI serial flash test and BLE functional test. The factory test code controls the Arduino 101’s Serial-1 interface, communicating with the test controller (or the operator). A serial terminal is connected to the DUT and the test code produces a command line menu for the user to invoke the tests.

Although the factory test code source is proprietary, the description of its function is as follows:

- Use the Arduino 101’s Serial-1 interface to interact with the user (test controller).
- Operation control is through a command line Interface using ASCII text format.
- External SPI serial flash test: Exercise the Arduino 101’s external serial flash via the SPI bus. Perform read/write tests on the entire flash, 2 Mb, erase it upon completion, and issue a pass/fail text string as output for the test result.
- BLE test: Put the Nordic BLE stack in and out of the Direct Test Mode (DTM). Exercise the stack by generating and receiving special test pattern BLE frames on a specific channel (frequency). Please note that this exercise should be conducted in an isolated environment to prevent unintended interference to the surrounding RF spectrum.

5. Board is reset upon flash completion. Insert a delay to allow for the board to be ready.
6. Flash the Nordic image via USB/DFU mode. The Nordic BLE firmware is released in a customer image flash pack called flash-ARDUINO101-PVT-USER.4.zip. Unzip the package, then the test controller executes the following command inside the flash_pack directory:
   ```
   bin\dfu-util.exe -d VendorID:ProductID -D images\firmware\ble_core\image.bin -v --alt 8 -R
   ```
   Where,
   ```
   VendorID = 8087H,
   ProductID = 0ABAH,
   ```
   -R = Initiate USB reset upon completion.
   Timeout is set at 90 seconds, 3 retries allowed. Failure terminates test sequence.

---

1 Upon the completion of JTAG flashing, the bootloader and the x86 factory test binary image are loaded.
Report test result upon exit to the operator who will determine if the DUT should proceed to Stage 2 testing.

**Stage 2 – BLE testing**

The main tasks performed by the BLE Testing are as follows:

1. Test the external SPI serial flash using the x86 Factory Test image (loaded in Stage 1).
2. Enable the test mode in the Nordic BLE chip.
3. Perform BLE measurement and testing.

**Test fixture**

The following are pictures of the Stage 2 test fixture:

*Figure 6: Top view Stage 2 test fixture*
Figure 7: Inside view Stage 2 test fixture

Figure 8: Arduino 101 hooked up in BLE Testing fixture
Please note that the final design of the test fixture for the BLE testing has the following changes:

- Removal of the USB hub.
- Operator no longer needs to connect the USB and the Serial interface to the DUT.
- Operator needs to mount the DUT to the fixture and secure it with the plastic hold down clips.
- Test point contacts are used for Serial interface and power supply connections at the USB socket.
- USB to serial link remains unchanged and thus the test controller still interfaces directly with the DUT via USB.

**Figure 9: Inside view BLE testing fixture**

**Figure 20: USB to serial interface**

**Figure 11: BLE analyzer antenna**
The following is the test sequence for the Stage 2, BLE Testing:

1. Operator connects USB to the DUT and then hooks up the two Serial-1 header pins to the USB to serial connection as shown in figure 8.
2. Operator mounts the DUT onto the fixture and closes the cover.
3. Test equipment setup and configuration is performed.
4. Test controller verifies the DUT SKU is an Arduino 101 board. Failure terminates test sequence.
5. Perform the SPI Flash test. The actual testing of the SPI flash is done by the factory test code\(^2\) that is running on the x86 core. The test controller invokes the test by sending in the following ASCII text string via the Arduino 101 Serial-1 connection:
   \[
   \text{spi\_flash run <CR>}
   \]
   Test controller expects an ASCII text output on the Serial-1 interface within 10 second or fails the test with a timeout error. Allow 3 retries. Failure terminates test sequence.
6. Enable Direct Test Mode (DTM), in the Nordic core. Test controller sends in the command in form of the following ASCII text, to the x86 core via Serial-1:
   \[
   \text{ble enable 1 1 <CR>}
   \]
   Test controller expects an ASCII text output with “OK”, within a small delay. Failure terminates test sequence.
7. Test controller initiates the BLE Tx test by sending the following command in ASCII text to the x86 core:
   \[
   \text{ble tx\_test start iChannel iLength iPacketType <CR>}
   \]
   Where,
   - Transmission frequency, \(iFreq = 2440 \text{ MHz}\),
   - \(iChannel = (iFreq - 2402) / 2\),
   - \(iLength = 25H\),
   - \(iPacketType = 2\)
   Test controller expects an ASCII text output with, “OK”, within a small delay.
8. Test controller initiates the BLE analyzer to measure the following parameters,
   - Average power,
   - Peak power,
   - Frequency offset,
   - Delta F1 average,
   - Delta F2 maximum.

\(^2\) Image loaded at the end of Stage 1 test.
Test controller expects the measurement\(^3\) within a small delay. Failure terminates test sequence.

9. Test controller examines the measurements as follows:
   \[
   0 \text{ dBm} \leq \text{Average power} \leq 6 \text{ dBm},
   \]
   \[
   0 \text{ dBm} \leq \text{Peak power} \leq 9 \text{ dBm},
   \]
   \[
   -72000 \text{ Hz} \leq \text{Frequency offset} \leq 72000 \text{ Hz},
   \]
   \[
   -99999.99 \leq \Delta F1 \text{ average} \leq 0,
   \]
   \[
   -500000 \leq \Delta F2 \text{ maximum} \leq 500000
   \]

Measurement outside of the specified range causes failure and terminates test sequence.

10. The test controller halts the BLE Tx test by sending the following command in ASCII text to the x86 core,
   \[
   \text{ble tx_text stop <CR>}
   \]

Test controller expects an ASCII text output with “OK”, within a small delay.

11. Test controller repeats step 7 to 10 one more time except this round with:
   \[
   iFreq = 2480 \text{ MHz}
   \]

12. Test controller proceeds with the BLE receive testing. Test controller initializes the BLE analyzer with the following parameters:
   \[
   \text{Frequency, } iFreq = 2402 \text{ MHz},
   \]
   \[
   \text{Signal Level} = -45 \text{ dBm}
   \]

13. Test controller initiates the BLE Rx test by sending the following command in ASCII text to the x86 core:
   \[
   \text{ble rx_test start } iChannel <CR>
   \]

Where,
   \[
   iChannel = (iFreq - 2402) / 2,
   \]

Test controller expects an ASCII text output with “OK”, within a small delay. Failure terminates test sequence.

14. Test controller invokes the BLE analyzer to send 1,000 packets. Wait for transmission completion, timeout set at 10 seconds. Failure terminates test sequence.

15. Test controller stops the BLE Rx test by sending the following command in ASCII text to the x86 core:
   \[
   \text{ble rx_test stop <CR>}
   \]

Test controller expects an ASCII text output with “OK”, within a small delay, in addition to the number received packets. Failure terminates test sequence.

16. Test controller checks the number of received packets to be in the range of 950 to 1050 packets. Otherwise, the test fails and terminates test sequence.

17. Repeat step 12 to 16 once again but with \(iFreq\) set to 2440 MHz.

18. Repeat step 12 to 16 once again but with \(iFreq\) set to 2480 MHz.

---

\(^3\) The protocol to the BLE analyzer is vendor equipment specific and, thus, not included in this document.
19. The stage 2 BLE testing is complete.

Stage 3 – User configuration setup

The main tasks of stage 3, User Configuration Setup, are as follows:

1. Generate the provision information for the DUT.
2. Flash the cores with user binary images.
3. Flash the factory test sketch to the ARC core.
4. Flash the provision data.

The DUT is prepared for the stage 4, functional testing. Please note that the binary images for the x86 and Nordic cores are the latest official release code. The test sketch is a special testing code that will be replaced upon successful completion of the stage 4 testing.

Test fixture

The stage 3 test fixture is similar to the one used in stage 1 and is as follows:

Figure 12: Stage 3 test fixture side view.

Figure 13: Stage 3 test fixture test point connection.

Figure 14: Stage 3 test fixture test point configuration.
Note:

- The USB hub was removed from the final test fixture design. Both the DUT and the flyswatter are directly connected to the PC USB port.

Test sequence

The following is the test sequence for Stage 3, User Configuration Setup:

1. Operator mounts the DUT onto the test fixture and secures it by closing the compressing cover.
2. Test controller initialization. Identify the DUT SKU is an Arduino 101 board. Failure to identify terminates the sequence.
3. Test controller generates the following provision data\(^4\) for the DUT,
   - Factory serial number
   - Hardware version.
   - Model name/number.
4. Using the flash_pack extracted from flash-ARDUINO101-PVT-USER.4.zip, the test controller has all the tools to flash the DUT with the bootloader and x86 image via JTAG/openOCD. The test controller executes the following command at the flash_pack directory:
   ```
   bin\openocd.exe  -f scripts\interface\ftdi\flyswatter2.cfg  -f scripts\board\firestarter.cfg   -f scripts\flash-jtag.cfg
   ```
   Timeout is set at 75 seconds with 3 retries allowed. Failure terminates test sequence.
5. The board is reset upon flash completion. Insert a delay to allow for the board to be ready.
6. Flash the Nordic image via USB/DFU mode. The test controller executes the following command at the flash_pack directory:
   ```
   bin\dfu-util.exe  -d VendorID:ProductID  -D images\firmware\ble_core\image.bin  -v --alt 8  -R
   ```
   Where,
   - VendorID = 8087H,
   - ProductID = 0ABAH,
   - R = Initiate USB reset upon completion.
   Timeout set at 90 seconds and 3 retries are allowed. Failure terminates test sequence.
7. The board is reset upon flash completion. Insert a delay to allow for the board to be ready.
8. Flash the factory test Sketch, Arduino101_TestShield_v2.cpp.bin\(^5\), to the ARC core via USB/DFU mode. The test controller executes the following command at the flash_pack directory:
   ```
   bin\dfu-util.exe  -d VendorID:ProductID  -D Arduino101_TestShield_v2.cpp.bin  -v  -alt 7  -R
   ```

---

\(^4\) The process of provision data generation is omitted due to the fact it is manufacturer specific.

\(^5\) This is a special test sketch, Arduino101_TestShield_v2.cpp.bin, that runs on the ARC core to exercise various component of the DUT for stage 4 testing.
Where,

VendorID = 8087H,
ProductID = 0ABAH,
-R = Initiate USB reset upon completion.

Timeout is set at 10 seconds with no retry allowed. Failure terminates test sequence.

7. Board is reset upon flash completion. Insert a delay to allow for the board to be ready.

8. Flash the DUT OTP with the provision data. The OTP flash beginning address for the Arduino 101 provision is at 0xFFFFFE200. The test controller executes the following command at the flash_pack directory:

```
  bin\openocd.exe  -f  scripts\interface\ftdi\flyswatter2.cfg  -f
  scripts\board\firestarter.cfg  -f  scripts\flash-jtag_provision.cfg
```

Timeout is set at 75 seconds with no retry allowed. Failure terminates the test sequence.

The **flash-jtag_provision.cfg** file is appended here:

9. Stage 3 testing is complete.

**OTP Provision Data**

The provision data for the Arduino 101 board is saved at the customer segment of the OTP flash, 0xFFFFFE200, for 512 bytes. The following is a memory dump of that area of flash,

![Memory Dump](image)

**Stage 4 – Functional testing**

The last stage of testing is Functional Testing and it performs the following tasks:

1. Verify proper operation of the power supply regulator.
2. Verify the functionality of all the I/O signals/pins using a factory test sketch running on the DUT ARC core against a special test shield.
3. Upon successful completion, print labels for the DUT, remove the factory test sketch, and replace a blank sketch as the ARC image.

Test fixture

The following diagrams show the making of the Functional Test fixture:
Functional test shield

A special test shield is used in the functional tests to exercise the I/O signals of the DUT. The shield provides all the necessary interfaces that can be connected to an Arduino 101. Using the factory test sketch, the test controller can instruct the DUT to exercise each interface individually to perform certain task with the test shield. The test sketch verifies the result of each task and reports back to the test controller.

Notes:

- The P1, USB connection provides the test controller to have access to the DUT serial interface.
The P2, USB connection is for the test controller direct access to the DUT USB. The power supply to the DUT USB is controlled by a power relay and can be turned off by the test controller.

![Factory test sketch](https://via.placeholder.com/150)

*Figure 23: Test Shield used in the Functional Test fixture*

The following is the test shield schematic drawing:

A sketch is a program developed under the Arduino IDE that runs on the ARC core of the Arduino 101 board. The Arduino development platform provides a comprehensive library that has substantial coverage of the Arduino 101’s hardware features. Thus, a developer can easily create a project that utilizes these features without the need to implement low level code/drivers.

The factory test sketch is developed to exercise some of the Arduino 101’s interfaces (hardware features) to the test shield to perform various tasks as tests. The sketch waits on the serial input to perform the following tests:

- I2C-1 and I2C-2 testing. Same exercise for two physical links.
- SPI-1 and SPI-2 testing. Same exercise for two physical links.
- ADC, analog to digital conversion testing.
The following is the factory test sketch source code:

Arduino101_TestShield_d_v2.ino

Test sequence

The procedure of the functional tests are as follows:

1. Test controller initialization and configuration.
2. Operator mounts the DUT onto the test fixture and secures it by closing up the cover.
3. Test controller selects a channel from the external power supply to power the DUT.
4. Test controller disables the power channel.
5. Test controller sets the power channel output voltage to 7v.
6. Test controller limits the power channel current to 1A.
7. Using a power relay, the test controller disables the power of the USB connection to the DUT. (The DUT is solely powered by the external power supply.)
8. Test controller enables the power channel of the external power supply. DUT is powered up.
9. Test controller verifies the DUT is an Arduino 101 board.
10. Test controller queries the test shield for its version number via the P1 USB connection. Test controller expects an ASCII text string as a response within a small delay. Allows 3 retries. Failure terminates the test sequence. Log the test shield version number in the test log.
11. Test controller queries the DUT for board information by executing the following steps. Allows 3 retries. Failure terminates the sequence.
   i. Test controller queries the DUT via P2 USB, for the Curie serial number by sending the following ASCII text command, via the serial/USB interface:
      `getcuriesn <CR>`
      Test controller expects an ASCII string output with the serial number of the Curie module and logs it. Allows 3 retries. Logs result.
   ii. Test controller queries the DUT for the Curie identification by sending the following ASCII text command, via the serial/USB interface:
       `gethwid <CR>`

---

6 This is an initial version of the test sketch. Not all commands mentioned in this document are supported.
7 The instructions to the power supply are equipment specific and thus not included in this document.
8 Instructions for operating the power relay is equipment specific and thus not included in this document.
9 This function is specific to the test shield board and thus its support is vendor specific.
10 Info saved in the OTP flash, Intel segment.
11 This command not supported in the initial version of the test sketch.
12 This command not supported in the initial version of the test sketch.
Test controller expects an ASCII string output of the Curie ID and compares it against the expecting ID. Logs result.

iii. Test controller queries the DUT for its MAC address\(^{13}\) by sending the following ASCII text command\(^ {14}\), via the serial/USB interface:

\[
\text{getmac} \ <\text{CR}>
\]

Test controller expects a 32 bit value of the MAC address and compares it with a value of 984FEEH. Logs result.

iv. Test controller queries the DUT for the board hardware version by sending the following ASCII text command\(^ {15}\), via the serial/USB interface:

\[
\text{gethwver} \ <\text{CR}>
\]

Test controller expects a 32 bit value and compares it with an expected version. Logs result.

v. Test controller queries the DUT for the board serial number by sending the following ASCII text command\(^ {16}\), via the serial/USB interface:

\[
\text{getsn} \ <\text{CR}>
\]

Test controller expects an ASCII string output of the DUT board serial number and compares it against the upper limit. Logs result.

12. The Arduino 101 I2C bus is available (connected internally) on two connectors: header socket J11 (pin 9 and 10) and header socket J14 (pin 5 and 6). To check out the I2C bus on both sockets, the test controller utilizes a multiplexer (MUX) in the test shield and the test sketch to select one socket to test at a time. The connection arrangement is described in the following diagram:

---

\(^{13}\) Info saved in the OTP flash, Intel segment.

\(^{14}\) This command not supported in the initial version of the test sketch.

\(^{15}\) The initial sketch returns a hard coded value.

\(^{16}\) This command not supported in the initial version of the test sketch.
Test controller performs I2C bus testing with the following steps:

i. To test one of the two I2C connections, the test controller sends the following ASCII text command to the Test Sketch running on the DUT via serial/USB interface:

   I2C1 <CR>

Upon the detection of I2C1 command, the test sketch sets pin 8 output to low (I2C2 command will set pin 8 to high). One of the I2C connections is selected to access the I2C target on the test shield. The test sketch proceeds with the following operations:

a. Write to register 0 at address 0x50 with an 8-bit unsigned integer of 0.
b. Read the content of register 0 at address 0x50.
c. Check if the read result is 0. If not, log failure condition to serial output and exit the test.
d. Write to register 0 at address 0x50 with an 8-bit unsigned integer of 0xFF.
e. Read the content of register 0 at address 0x50.
f. Check if the read result is 0xFF. If not, log failure condition to serial output and exit the test.

Test sketch log “PASSED” on serial output upon test completion.

Test controller expects an ASCII string output stating, “PASSED”, from DUT. Allows 3 retries. Failure terminates the test sequence.

ii. Test the second I2C connection. Test controller executes the previous step one more time with the exception of issuing the ASCII text command:

   I2C2 <CR>
13. The Arduino SPI bus is made available (internally connected) at these connectors: J11 and J12. Similar to the testing of the I2C bus setup, the test controller performs SPI bus testing with the following steps. The test controller utilizes a MUX in the test shield and the test sketch to select one socket to test at a time. Test controller performs I2C bus testing with the following steps:

i. To test the one of the two SPI connections, the test controller sends the following ASCII text command to the Test Sketch running on the DUT via serial/USB interface:

   SPI1  <CR>

   Upon the detection of SPI1 command, the test sketch sets pin 9 output to low (SPI2 command will set pin 9 to high). One of the SPI connections is selected to access the SPI Target on the Test Shield. The test sketch proceeds with the following operations:

   a. Enable chip select by setting SPI_CS low.
   b. Transmit Write Enable opcode.
   c. Disable chip select, set SPI_CS high.
   d. Enable chip select by setting SPI_CS low.
   e. Transmit Write opcode.
   f. Transmit register address, 0.
   g. Transmit a value of 0.
   h. Disable chip select, set SPI_CS high.
   i. Enable chip select by setting SPI_CS low.
   j. Transmit Read opcode.
   k. Transmit register address, 0.
   l. Receive a value.
   m. Disable chip select, set SPI_CS high.
   n. Check the received value is 0. If not, fail and exit the test.
   o. Repeat the above steps one more time except this time with writing a value of 0xFF to register 0. Check the read back value with 0xFF.

   Test sketch log “PASSED” on serial output upon test completion.

   Test controller expects an ASCII string output stating, “PASSED”, from DUT. Allows 3 retries. Failure terminates the test sequence.

ii. Test the second SPI connection. Test controller executes previous step one more time with the exception of issuing the ASCII text command:

   SPI2  <CR>
14. Proceed with the testing of D2, D3, D4, and D7 I/O signals using the special arrangement of the Test Shield shown as follows:

![Test Shield Diagram]

The D2 and D4 signals are internally connected in the test shield and so are the D3 and D7 signals. Test controller tests these signals as follows:

i. To test D2 and D4, the test controller issues the following ASCII test command\(^\text{17}\) via the serial connection to the test sketch:

   D2D4  <CR>

   Test sketch exercises the connection and reports the result via serial connection. Test controller is expecting an ASCII string of, “PASSED”. Allows 3 retries. Failure terminates the test sequence.

ii. Test controller repeats the previous step one more time for testing D3 and D7 by issuing the following ASCII text command:

   D3D7  <CR>

**Conclusion**

The Arduino 101 manufacturing test flow follows the steps of the four test stages detailed in this document. In addition, this document includes:

- Information on the hardware shield and test equipment involved.
- Functional description of the software involved, and their image locations if applicable.
- Pictures, drawings and schematics of the actual equipment.

\(^\text{17}\) Initial version of the test sketch does not support this command.