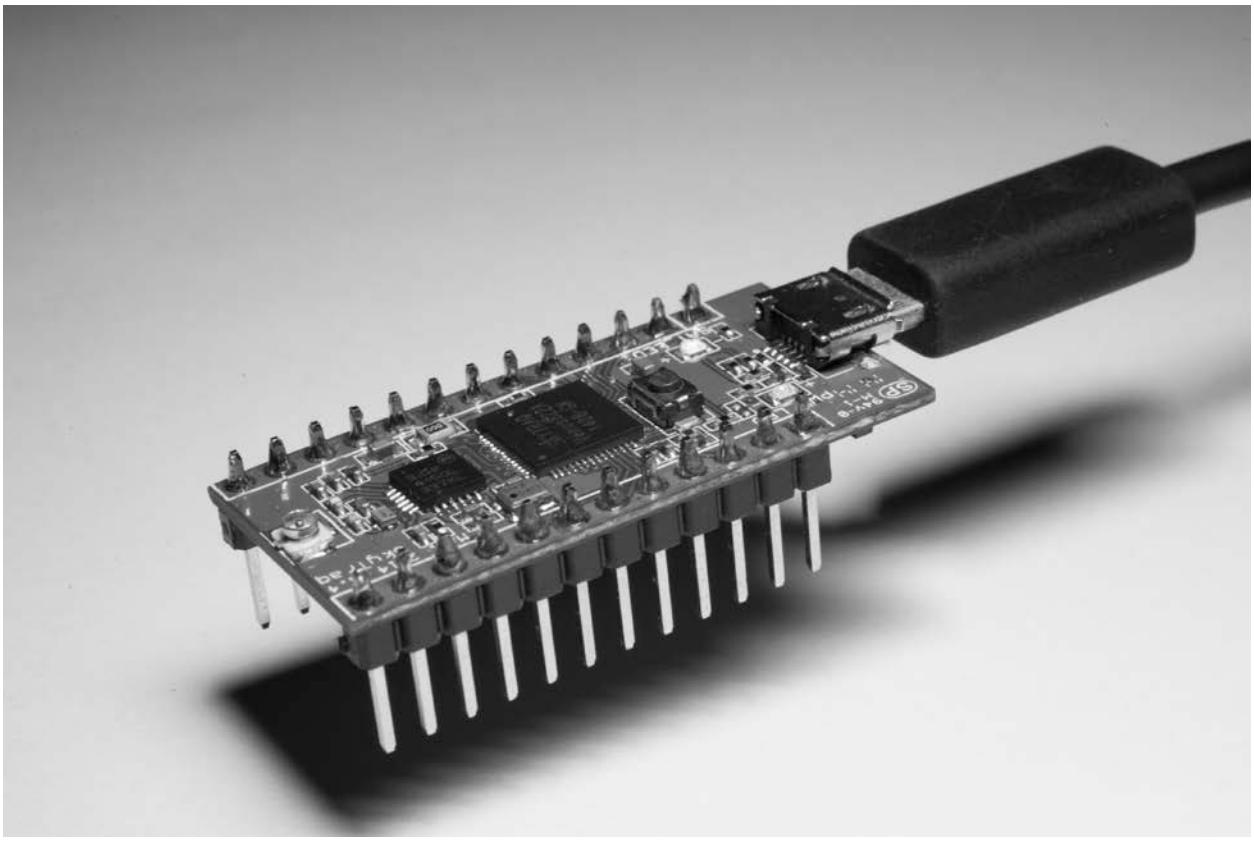
NavSpark



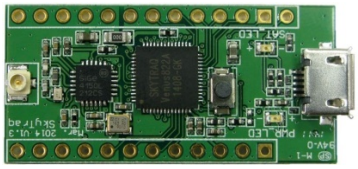
NavSpark User Guide

Rev. 0.8

December 14, 2015

Table of Contents

1. Introduction



NavSpark is a tiny 32bit microcontroller board with satellite navigation capability built around SkyTraq Venus822A baseband. Venus822A is a powerful small chip containing 100MHz-capable 32bit LEON3 Sparc-V8 processor, 64bit IEEE-754 Floating Point Unit, GPS / GLONASS / Beidou/Galileo Global Navigation Satellite System (GNSS) baseband engine, 1MByte Flash memory, 212KByte SRAM, UART, SPI, two-wire, PWM, and GPIO peripherals.

The Venus822A is designed for quad-GNSS application, simultaneously processing signals from GPS, GLONASS, Beidou, and Galileo satellite systems. While used for GPS-only or GPS/GLONASS and GPS/Beidou dual-satellite navigation, it still has plenty of computation power and memory left for user application, far more than a typical 8bit microcontrollers.

We want to provide an Arduino-IDE programmable satellite navigation receiver board small and low-cost enough to fit into any projects requiring location capability. For small projects, the programmable feature of NavSpark removes the need for an extra 8bit microcontroller. NavSpark can be connected to a computer and programmed over USB port just like an Arduino.

NavSpark is 3.3V type. The I/O interfaces are 3.3V LVTTL level. It can be powered from the USB interface or a LiPo battery.
There are three versions of NavSpark boards. They differ in the RF-front end circuitry used.

Name	Feature
NavSpark	32 bit development board with GPS receiver capability
NavSpark-GL	32bit development board with GPS/GLONASS receiver capability
NavSpark-BD	32bit development board with GPS/Beidou receiver capability
NavSpark-mini	32bit development board with GPS/Beidou receiver capability

Each development board can load two different libraries: one with the corresponding GNSS features for GNSS applications, another without GNSS features for non-GNSS applications. When user links Arduino code with library having GNSS feature, the development board will output NMEA messages through UART1, user can also extract the GNSS information in their code by calling appropriate APIs (see examples in later part of the document for details). When user links Arduino code with the library having no GNSS feature, the development board will not output any NMEA messages on UART1, and most of the computation resource and memory can be used by the user.

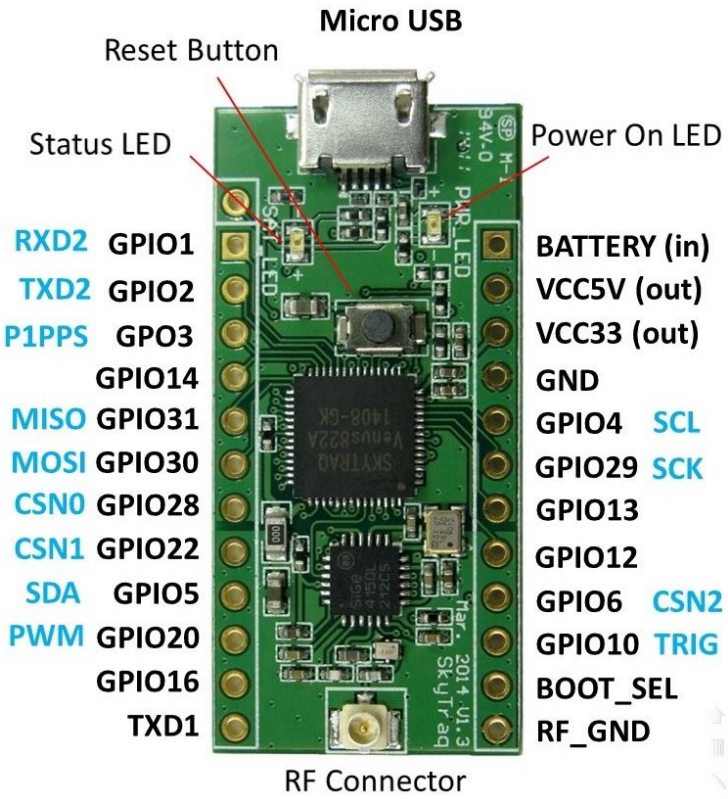
2. Features of NavSpark, NavSpark-BD, and NavSpark-GL

- 100MHz 32bit LEON3 Sparc-V8 + IEEE-754 Compliant FPU
- 1024KB Flash Memory + 212KB RAM
- ~80uA/MHz @ 3.3V
- Powered and programmed by micro USB connector
- one full duplex asynchronous UART
- one asynchronous UART transmit
- one time-shared SPI with 3 chip selects to control 3 slave devices
- one two-wire interface
- one PWM
- seventeen digital I/O pins (shared with above functional pins)
- atomic clock synchronized P1PPS time reference pulse with +/-10nsec accuracy
- power-on green LED indicator
- GPS status blue LED indicator
- fail-safe ROM mode to reprogram in case improperly coded user program hangs
- on-board 3.3V LDO regulator with 250mA output capability
- reset button to restarting program
- breadboard compatible
- 18mm x 38mm size

3. Features of NavSpark-mini

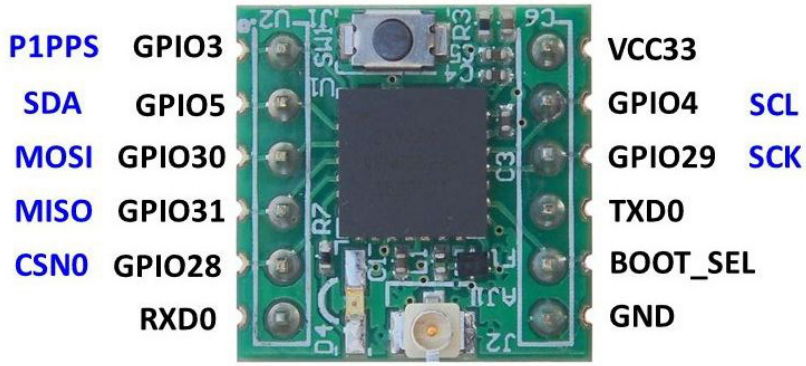
- 100MHz 32bit LEON3 Sparc-V8 + IEEE-754 Compliant FPU
- 1024KB Flash Memory + 212KB RAM
- ~80uA/MHz @ 3.3V
- Powered and programmed by micro USB connector
- one full duplex asynchronous UART
- one time-shared SPI with 1 chip selects to control 1 slave devices
- one two-wire interface
- seven digital I/O pins (shared with above functional pins)
- power-on red LED indicator
- atomic clock synchronized P1PPS time reference pulse with +/-10nsec accuracy
- fail-safe ROM mode to reprogram in case improperly coded user program hangs
- reset button to restarting program
- breadboard compatible
- 17mm x 17 mm size

4. I/O Function Overview of NavSpark, NavSpark-BD, and NavSpark-GL



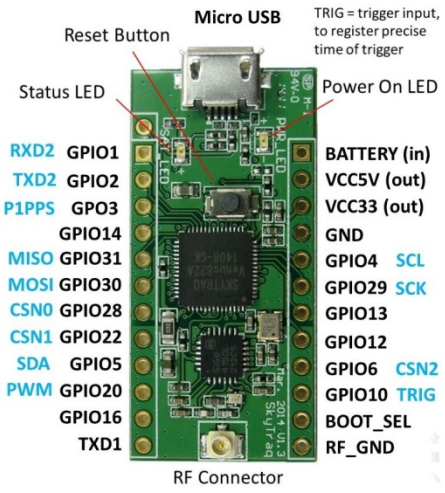
- **Micro USB Connector**: Used to load program to NavSpark, or output NMEA to PC for GNSS applications. Also provides 5V power to the development board.
- **Green Power On LED**: the board is powered up when this LED lights up.
- **Blue GPS Status LED**: With default firmware, the GPS status LED lights up when initially powered on and not getting position fix. It toggles every second when getting position fix.
- **BATTERY Input**: Power NavSpark from an external battery and use it outdoors. This pin can take 3.6V ~ 10V DC input, and has reverse-polarity protections. The board will use either the battery or USB 5V power, whichever is connected. When both are connected, it will use one that has the higher voltage.
- **VCC5V Power Output**: The 5V power from the USB jack can also be used to power other circuit from this pin, up to 400mA can be drawn from this pin.
- **VCC33 Power Output**: The on-board 3.3V LDO regulator has rating of 250mA, provides power to the entire board. NavSpark can power up to 150mA @ 3.3V to other circuitry from this pin.
- **TXD1**: UART1 transmit output, same as the one connected to USB port, also made available here.
- **BOOT_SEL**: Default unconnected, internally pulled-high, Venus822A runs program from internal Flash memory. When connected to GND, Venus822A runs program from internal ROM memory. When improperly written user code hangs the system, it would be necessary connect BOOT_SEL to GND and recover from ROM mode to load corrected code.
- **GPIO**: There are 17 3.3V GPIO pins. Next section provides detailed descriptions.
- **RESET Button**: Pressing the reset button reset the NavSpark hardware and restarts the internal LEON3 processor.
- **RF Connector**: GNSS antenna should be connected via this IPEX or U.FL RF connector for GNSS application.

5. I/O Function Overview of NavSpark-mini



- **Micro USB Connector**: Used to load program to NavSpark, or output NMEA to PC for GNSS applications. Also provides 5V power to the development board.
- **BOOT_SEL**: Default unconnected, internally pulled-high, Venus822A runs program from internal Flash memory. When connected to GND, Venus822A runs program from internal ROM memory. When improperly written user code hangs the system, it would be necessary connect BOOT_SEL to GND and recover from ROM mode to load corrected code.
- **GPIO**: There are 7 3.3V GPIO pins. Next section provides detailed descriptions.
- **RESET Button**: Pressing the reset button reset the NavSpark hardware and restarts the internal LEON3 processor.
- **RF Connector**: GNSS antenna should be connected via this IPEX or U.FL RF connector for GNSS application.

6. Pin out Description of NavSpark, NavSpark-BD, and NavSpark-GL



There are total of 16 general purpose I/O pins named as “GPIOnn” which can be programmed as input or output pin, some of them can also be configured for special functions as described in below table. An exception, “GPO3”, can only be programmed as a general output pin or a dedicated output pin for P1PPS. When the GPIO pins are configured as output, they can sink ~7mA when driven LOW, source ~13mA when driven HIGH; use a transistor switch when needing to drive devices requiring higher current.

PIN INFORMATION		
NAME	**I/O**	**DESCRIPTION**
GPIO1 **UART2_RX**	IN/OUT IN	General purpose I/O pin 1 or the Rx input pin for UART2.
GPIO2 **UART2_TX**	IN/OUT OUT	General purpose I/O pin 2 or the Tx output pin for UART2.
GPO3 **P1PPS**	OUT OUT	General purpose output pin 3 or P1PPS output pin. When operating in P1PPS mode, this pin will toggle every second once the module gets position fix from 4 or more satellites; rising edge of the P1PPS pulse is synchronized to UTC second with +/-10nsec accuracy.
GPIO4 **TW_SCL**	IN/OUT IN/OUT	General purpose I/O pin 4 or the clock pin of two-wire interface. When this pin operates in two-wire master mode, this pin is an output pin for clock output to two-wire slave. When this pin operates in two-wire slave mode, it is an input pin.
GPIO5 **TW_SDA**	IN/OUT IN/OUT	General purpose I/O pin 5 or the bidirectional data pin of two-wire interface.
GPIO6 **SPI_CSN2**	IN/OUT	General purpose I/O pin 6 or the chip select #2 (negative enable) pin of SPI master.
GPIO10 **TRIG**	IN/OUT INPUT	General purpose I/O pin 10 or external trigger input for registering precise time of the trigger.
GPIO12	IN/OUT	General purpose I/O pin 12.
GPIO13	IN/OUT	General purpose I/O pin 13.
GPIO14	IN/OUT	General purpose I/O pin 14.
GPIO16	IN/OUT	General purpose I/O pin 16.
GPIO20 **PWM**	IN/OUT OUT	General purpose I/O pin 20 or the PWM output pin.

GPIO22 **SPI_CSN1**	IN/OUT OUT	General purpose I/O pin 4 or the chip select #1 (negative enable) pin of SPI master.
GPIO28 **SPI_CSN0**	IN/OUT IN/OUT	General purpose I/O pin 28 or the chip select #0 (negative enable) pin of SPI master/slave. When it operates in SPI master mode, this pin is an output pin. When it operates in SPI slave mode, this pin is an input pin.
GPIO29 **SPI_SCK**	IN/OUT IN/OUT	General purpose I/O pin 29 or the serial clock pin of SPI master/slave. When it operates in SPI master mode, this pin is an output pin. When it operates in SPI slave mode, this pin is an input pin.
GPIO30 **SPI_MOSI**	IN/OUT IN/OUT	General purpose I/O pin 30 or the MOSI (Master-Out-Slave-In) pin of SPI master/slave. When it operates in SPI master mode, this pin is an output pin. When it operates in SPI slave mode, this pin is an input pin.
GPIO31 **SPI_MISO**	IN/OUT IN/OUT	General purpose I/O pin 31 or the MISO (Master-In-Slave-Out) pin of SPI master/slave. When it operates in SPI master mode, this pin is an input pin. When it operates in SPI slave mode, this pin is an output pin.
TXD1	OUT	The Tx pin of UART1 is used to output NMEA message when compiled with library having GNSS capability.
BATTERY	PWR IN	External supply input, 3.6V ~ 10V. User can connect Li-Ion or Li-Polymer battery to this pin.
VCC5V	PWR OUT	5V from USB, passing through Schottky diode, to provide power to other circuitry. Source up to 400mA current.
VCC33	PWR OUT	DC 3.3V output pin. A 3.3V LDO regulator taking up to 10V from BATTERY or USB connection and provides regulated 3.3V to power the board. This pin can provide up to 150mA @ 3.3V to power other circuitry.
GND	PWR	Digital ground pin.
BOOT_SEL	IN	Boot mode selection pin. When left open, NavSpark boots from internal Flash memory. When connected to GND, NavSpark boots from internal ROM memory; used to load code to Flash memory when improperly written user program hangs.
RF_GND	PWR	Ground pin for RF circuit.

7. Pin out Description of NavSpark-mini

There are total of 7 general purpose I/O pins named as “GPIOnn” which can be programmed as input or output pin, some of them can also be configured for special functions as described in below table. An exception, “GPIO3”, can only be programmed as a general output pin or a dedicated output pin for P1PPS. When the GPIO pins are configured as output, they can sink ~7mA when driven LOW, source ~13mA when driven HIGH; use a transistor switch when needing to drive devices requiring higher current.

PIN INFORMATION		
NAME	**I/O**	**DESCRIPTION**
GPO3 **P1PPS**	OUT OUT	General purpose output pin 3 or P1PPS output pin. When operating in P1PPS mode, this pin will toggle every second once the module gets position fix from 4 or more satellites; rising edge of the P1PPS pulse is synchronized to UTC second with +/-10nsec accuracy.
GPIO4 **TW_SCL**	IN/OUT IN/OUT	General purpose I/O pin 4 or the clock pin of two-wire interface. When this pin operates in two-wire master mode, this pin is an output pin for clock output to two-wire slave. When this pin operates in two-wire slave mode, it is an input pin.
GPIO5 **TW_SDA**	IN/OUT IN/OUT	General purpose I/O pin 5 or the bidirectional data pin of two-wire interface.
GPIO28 **SPI_CSN0**	IN/OUT IN/OUT	General purpose I/O pin 28 or the chip select #0 (negative enable) pin of SPI master/slave. When it operates in SPI master mode, this pin is an output pin. When it operates in SPI slave mode, this pin is an input pin.
GPIO29 **SPI_SCK**	IN/OUT IN/OUT	General purpose I/O pin 29 or the serial clock pin of SPI master/slave. When it operates in SPI master mode, this pin is an output pin. When it operates in SPI slave mode, this pin is an input pin.
GPIO30 **SPI_MOSI**	IN/OUT IN/OUT	General purpose I/O pin 30 or the MOSI (Master-Out-Slave-In) pin of SPI master/slave. When it operates in SPI master mode, this pin is an output pin. When it operates in SPI slave mode, this pin is an input pin.
GPIO31 **SPI_MISO**	IN/OUT IN/OUT	General purpose I/O pin 31 or the MISO (Master-In-Slave-Out) pin of SPI master/slave. When it operates in SPI master mode, this pin is an input pin. When it operates in SPI slave mode, this pin is an output pin.
VCC33	PWR OUT	DC 3.3V output pin. A 3.3V LDO regulator taking up to 10V from BATTERY or USB connection and provides regulated 3.3V to power the board. This pin can provide up to 150mA @ 3.3V to power other circuitry.
GND	PWR	Digital ground pin.
BOOT_SEL	IN	Boot mode selection pin. When left open, NavSpark boots from internal Flash memory. When connected to GND, NavSpark boots from internal ROM memory; used to load code to Flash memory when improperly written user program hangs.
RF_GND	PWR	Ground pin for RF circuit.

8. Installation of USB Driver

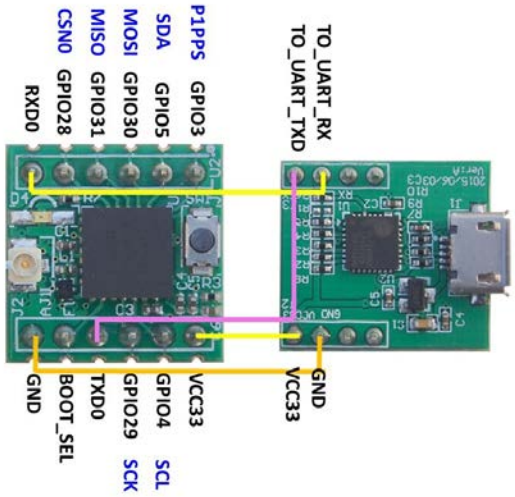
NavSpark uses Prolific's PL-2303HXD to send/receive UART1 data over the micro USB port. The Linux driver is already bundled in the newer Linux platforms and you could use command "dmesg | grep pl2303" to check if your Linux has PL2303 driver already.

```
dmesg | grep 'pl2303'
[   19.765166] USB Serial support registered for pl2303
[   19.765166] pl2303 2-2:1.0: pl2303 converter detected
[   19.816041] usb 2-2: pl2303 converter now attached to ttyUSB0
[   19.816073] usbcore: registered new interface driver pl2303
[   19.816077] pl2303: Prolific PL2303 USB to serial adaptor driver
[  219.414881] pl2303 ttyUSB0: pl2303 converter now disconnected from ttyUSB0
[  219.414932] pl2303 2-2:1.0: device disconnected
[  343.993340] pl2303 2-2:1.0: pl2303 converter detected
[  344.025786] usb 2-2: pl2303 converter now attached to ttyUSB0
[ 1306.836308] pl2303 ttyUSB0: pl2303 converter now disconnected from ttyUSB0
[ 1306.836353] pl2303 2-2:1.0: device disconnected
alex@ubuntu:~$
```

On Windows platform, Windows should search and find the driver automatically the first time NavSparkis connected to the PC. User may need to download and install the driver manually from Prolific's website in case the auto-search fails:
http://www.prolific.com.tw/US/ShowProduct.aspx?p_id=225&pcid=41

9. How to connect NavSpark-mini with UART-to-USB Adapter Board (for NavSpark-mini)

Connect GND to GND, VCC33 to VCC33, TO_UART_RX to RXD0, and TO_UART_TXD to TXD0.

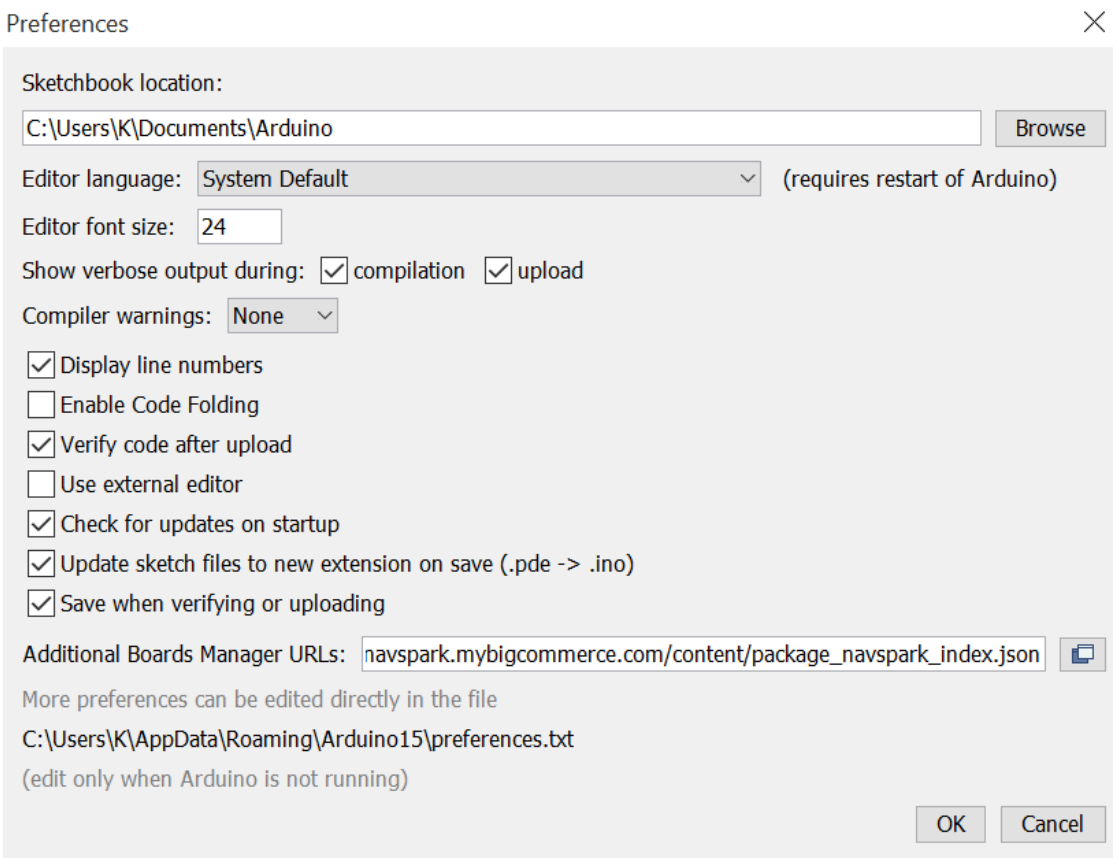


10. Setting Up Arduino IDE for NavSpark

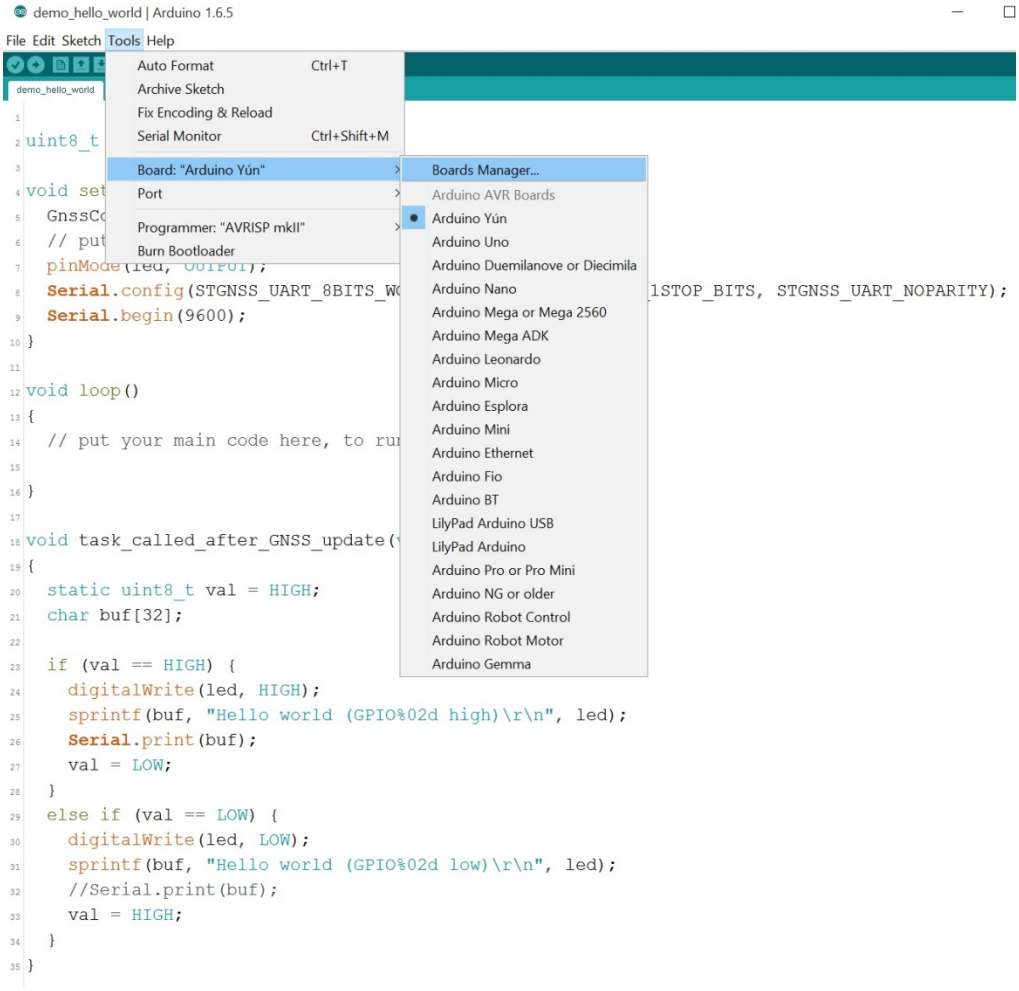
Installation: Step-by-Step instructions for setting up the Arduino Software and connecting it to a NavSpark, NavSpark-GL, NavSpark-BD, and NavSpark-mini.

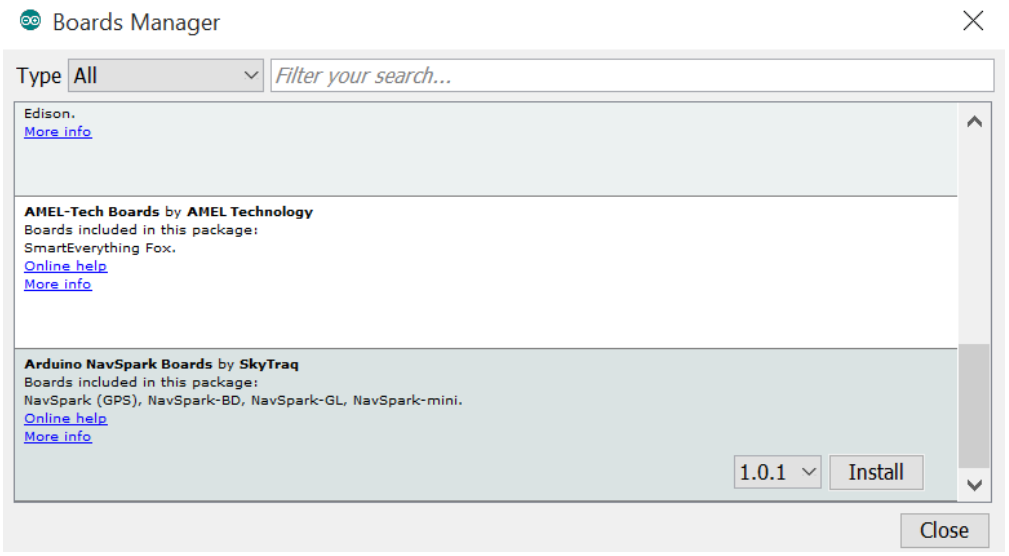
Windows

1. Download and Install **Arduino IDE** from Arduino website: https://www.arduino.cc/en/Main/Software (Both Installer and ZIP file for non admin install are fine).
2. Download "**sparc-elf-3.4.4-mingw.zip**" from our Resources from our website and extract it under "c:\opt". Note the target directory must be "c:\opt" or the tool chains will NOT function correctly.
3. Connect the NavSpark board to your computer using a micro USB cable. The green/red power LED should go on.
4. Launch the Arduino Application (Double-click Arduino application (arduino.exe)).
5. Add the following URL in Additional Board Manager URLs:
   http://navspark.mybigcommerce.com/content/package_navspark_index.json
   (**File → Preferences → Additional Board Manager URLs**)

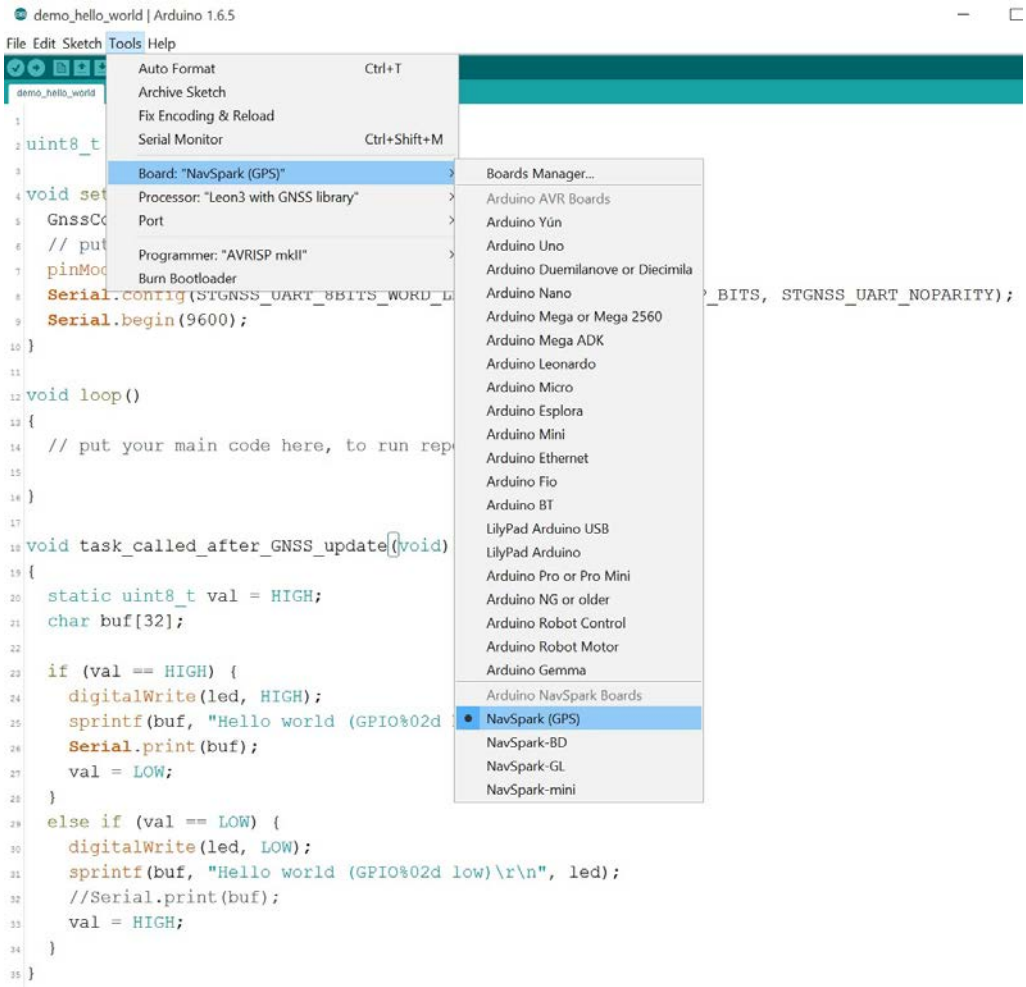


6. Go to **Tools → Boards → Boards Manager**. Scroll down and install **Arduino NavSpark Boards by SkyTraq**



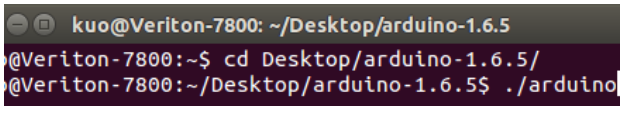


7. After installation, you'll need to select the entry in the **Tools → Board** menu that corresponds to your NavSpark.



Linux (Ubuntu)

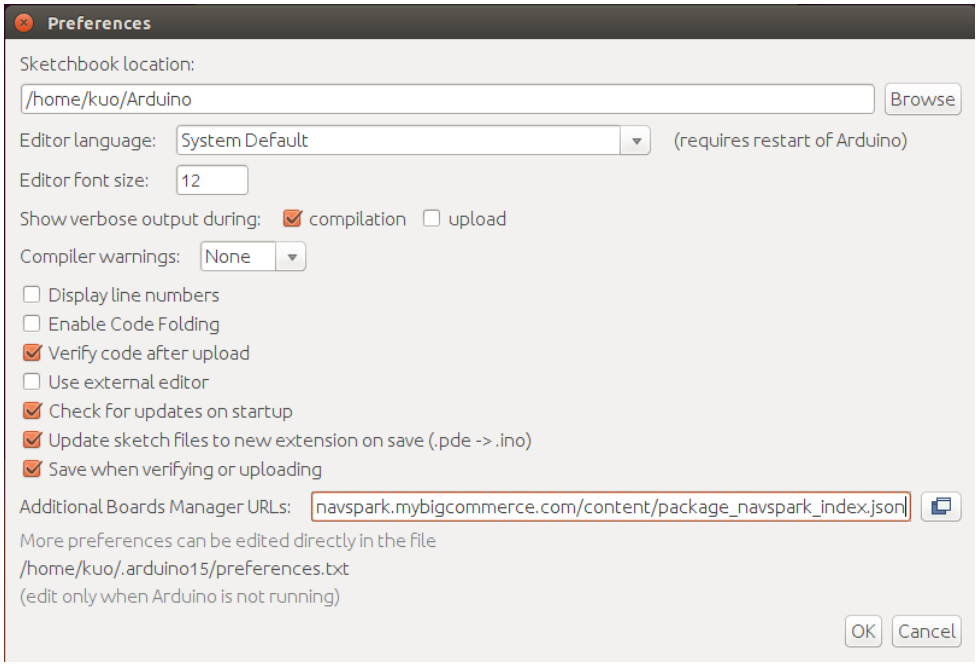
1. Download and Install **Arduino IDE** from Arduino website: https://www.arduino.cc/en/Main/Software (**Linux 32 bits**).
2. Download "**sparc-elf-3.4.4-mingw.tar.gz**" from Resources on our website and untar it to "/opt" by command "**sudo tar xvfz sparc-elf-3.4.4-33.tar.gz –C /opt**". Note the target directory must be "/opt" or the tool chains will NOT function correctly.
3. Go to Ubuntu Terminal, and type "**sudo apt-get update && sudo apt-get install openjdk-7-jre:i386**" to update and download Java.
4. Connect the NavSpark board to your computer using a micro USB cable. The green/red power LED should go on.
5. Launch the Arduino Application using terminal.



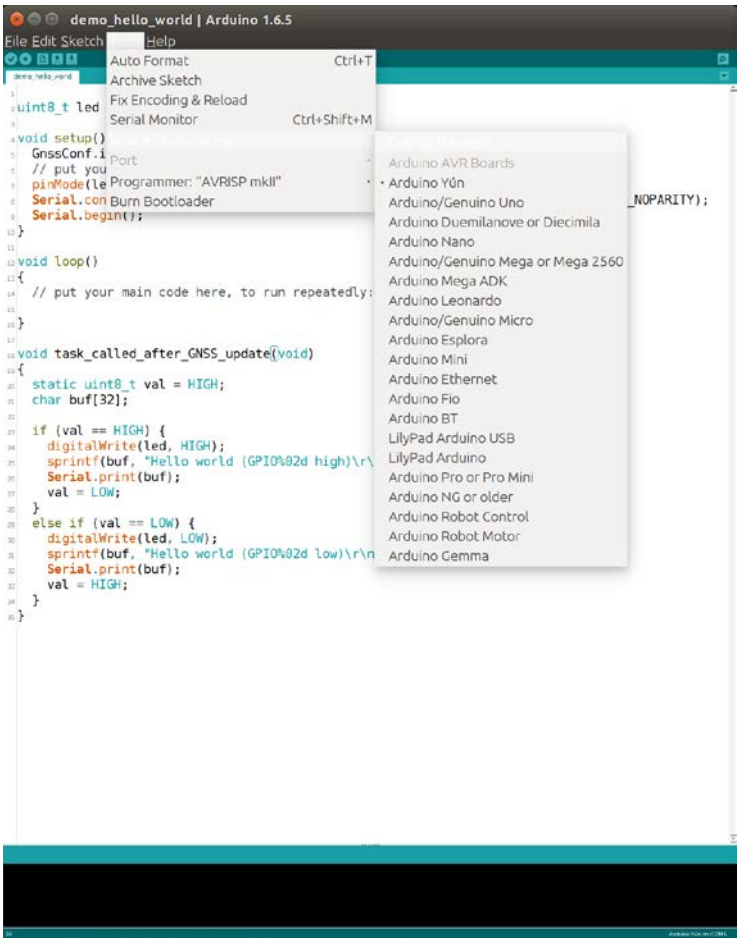
6. Add the following URL in Additional Board Manager URLs

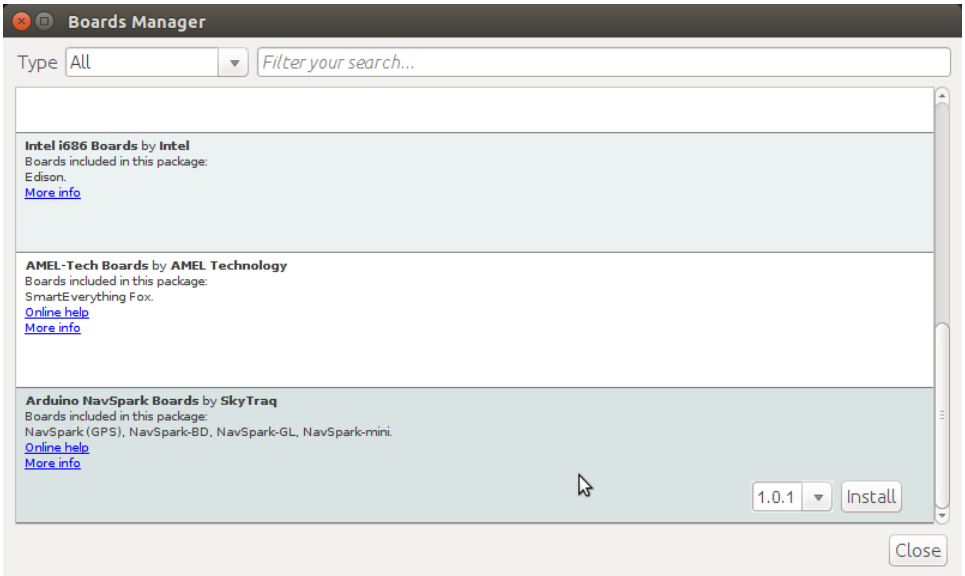
   http://navspark.mybigcommerce.com/content/package_navspark_index.json

(**File → Preferences → Additional Board Manager URLs**.)

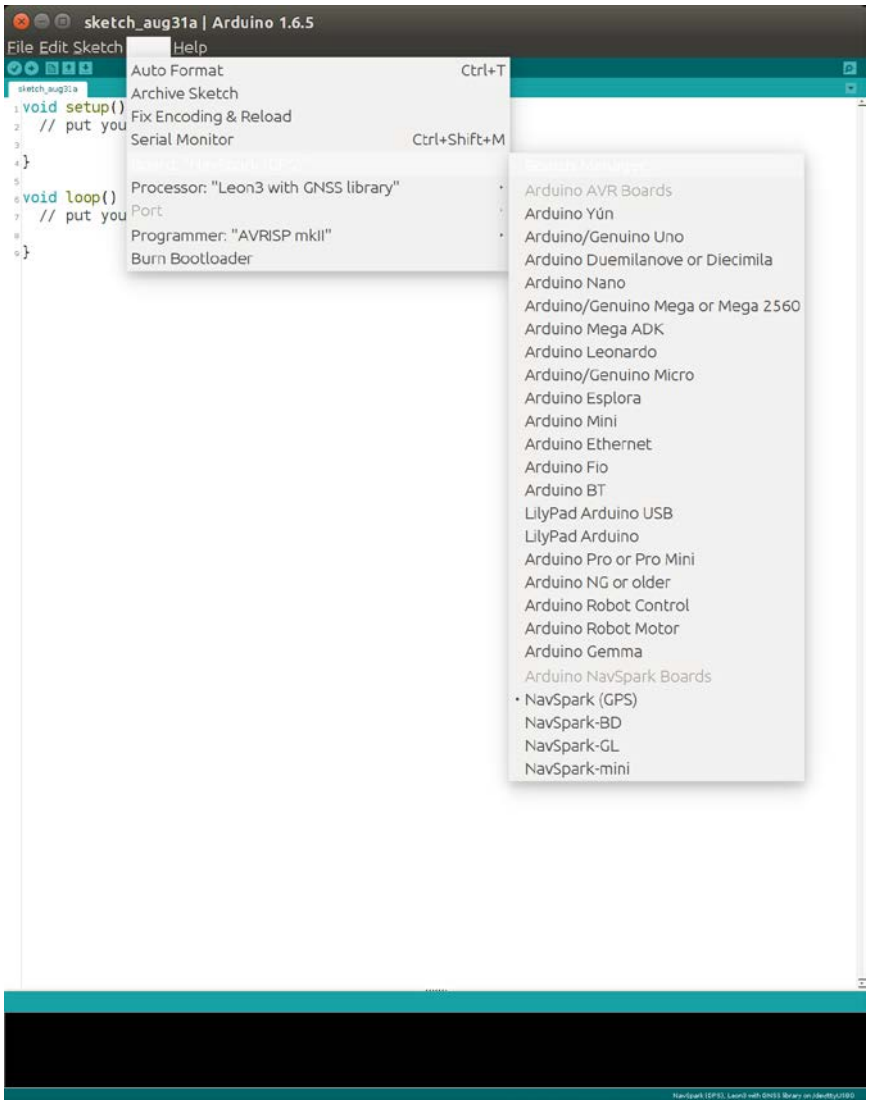


7. Go to **Tools → Boards → Boards Manager**. Scroll down and install **Arduino NavSpark Boards by SkyTraq**.





8. After installation, you'll need to select the entry in the **Tools → Board** menu that corresponds to your NavSpark.



11. How to Build the Default Binary

It is quite easy for users to build the default binary image for NavSpark. First of all, user must run the Arduino IDE and it will open a default sketch, choose “Tools” pull-down menu followed by selecting “Board”, then select which NavSpark board you have listed in the group “Arduino NavSpark Boards” (see Figure 1). The second step is to choose “Tools” pull-down menu again followed by selecting “Processor”, and select which library you prefer. The library “NavSpark-xxx with GNSS library” is used to build binary with GNSS capability (see Figure 2) and the library “NavSpark-xxx without GNSS library” is used build binary without GNSS capability, allowing user more processor resource when not using GNSS (see Figure 3). Add “GnssConf.init();” in void setup().

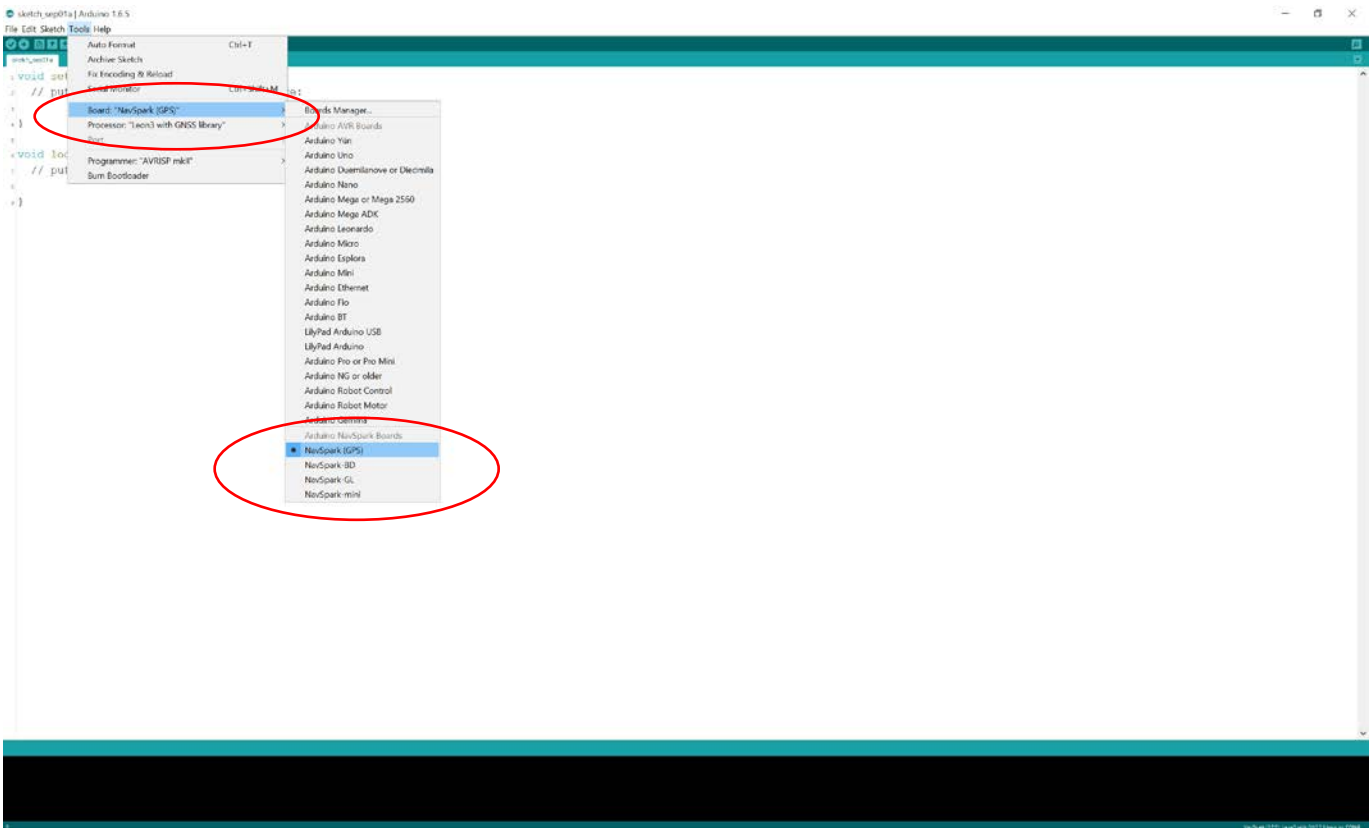


Figure 1

The final step is to choose “sketch” page and select “Verify / Compile” or just press “ctrl + R” to begin the compiling process and wait for the build to finish. If no coding error occurs, the compiled binary will be placed in the directory “Users\USER\AppData\Local\Temp”.

In default setting, command and code upload to NavSpark uses 115200 baud rate.

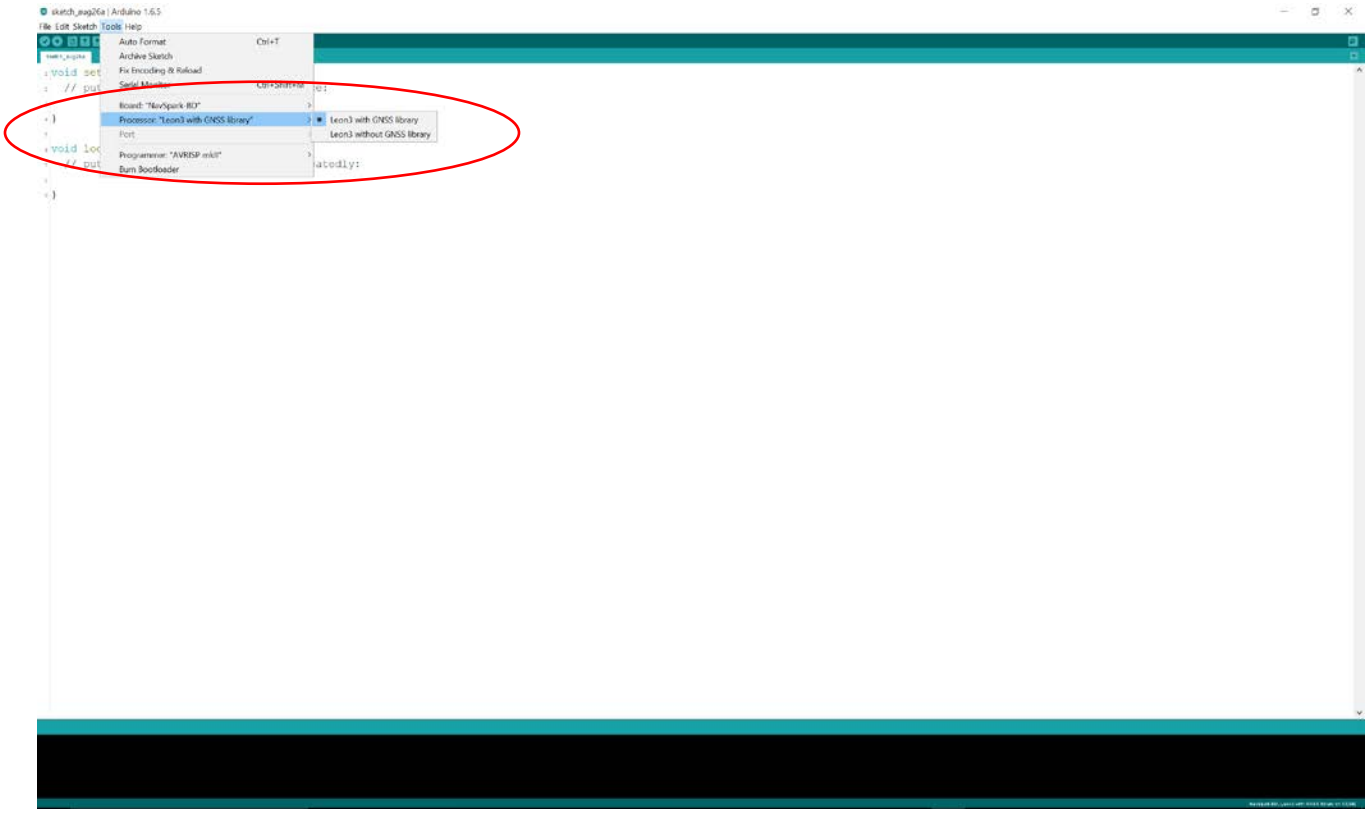


Figure 2

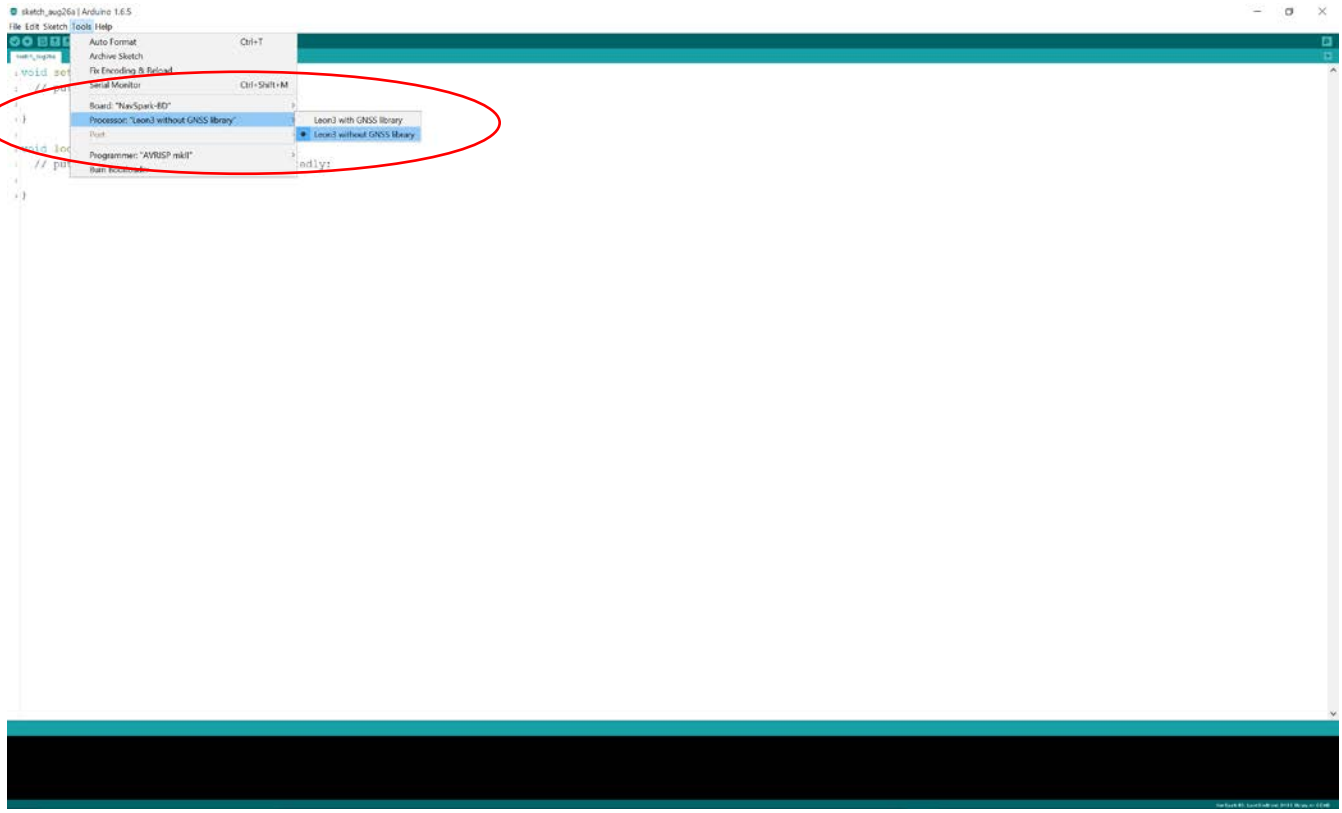


Figure 3

12. List of Member Functions

【GNSSParam】

GNSSParam

- GNSSParam(void)
- void setDefault(void)
- void setNavMode(uint8_t mode)
- void setUpdateRate(uint8_t rate)
- void setDopMaskMode(uint8_t mode)
- void setPdopMask(float pdop)
- void setHdopMask(float hdop)
- void setGdopMask(float gdop)
- void init(void)
- bool init_done(void)

【GNSS】

GNSS

- GNSS(void)
- void update(void)
- uint8_t fixMode(void)
- bool isUpdated(void)
- double distanceBetween(double lat1, double lon1, double lat2, double lon2)
- double courseTo(double lat1, double lon1, double lat2, double lon2)
- GNSSDate
  - void update(uint16_t year, uint8_t month, uint8_t day)
  - uint16_t year(void)
  - uint8_t month(void)
  - uint8_t day(void)
  - uint16_t formatString(char* str)
- GNSSTime
  - void update(uint8_t hour, uint8_t min, float second)
  - uint8_t hour(void)
  - uint8_t minute(void)
  - uint8_t second(void)
  - uint8_t centisecond(void)
  - uint16_t formatString(char* str)
- GNSSLocation
  - void update(double lat, double lon)
  - double latitude()
  - double longitude()
  - uint16_t latitude_formatString(char* str)
  - uint16_t longitude_formatString(char* str)
- GNSSAltitude
  - void update(float altitude)
  - float meters(void)
  - float miles(void)
  - float kilometers(void)
  - float feet(void)
- GNSSGeoSeparation
  - void update(float meters)
  - float meters(void)
  - float miles(void)
  - float kilometers(void)
  - float feet(void)
- GNSSSpeed
  - void update(float speed)
  - float kph(void)
  - float knots(void)
  - float mph(void)
- GNSSCourse
  - void update(float course)
  - float deg(void)
- GNSSSatellites

- void update(PVT_DATA_T* pPvtData, SV_INFO_T* pSvInfo)
- uint16_t numGPSInView(uint16_t *prn)
- uint16_t numBD2InView(uint16_t *prn)
- uint16_t numGLNInView(uint16_t *prn)
- uint16_t numGPSInUse(uint16_t *prn)
- uint16_t numBD2InUse(uint16_t *prn)
- uint16_t numGLNInUse(uint16_t *prn)
- uint16_t elevation(uint8_t constellation, uint16_t prn)
- uint16_t azimuth(uint8_t constellation, uint16_t prn)
- uint16_t CNR(uint8_t constellation, uint16_t prn)

【SPI_MasterSlave】

SPI_MasterSlave

- SPI_MasterSlave(uint8_t type)
- SPI_MasterSlave(void)
- void config(uint8_t spiMode, uint32_t clkRate, bool en_cs1, bool en_cs2)
- void begin(void)
- void resetTx(void)
- void resetRx(void)
- size_t write(uint8_t data)
- size_t write(uint8_t *data, size_t size)
- void slaveSelect(uint8_t slv)
- size_t transfer(void)
- size_t transfer(size_t size)
- int available(void)
- int remaining(void)
- int read(void)
- uint8_t pick(size_t offset)
- void enableBufferForHostWrite(void)
- bool validBufferForHostRead(void)
- uint8_t copyDataToBufferForHostRead(void)
- void enableBufferForHostRead(void)
- void attachInterrupt(uint8_t type, void (*userFunc)(void))
- void detachInterrupt(uint8_t type)
- void isr(uint8_t type)

Remarks:

SPI class in version 1.0.3 has been moved to SPI_MasterSlave. The two pre-instantiated SPI objects remain the same: spiMaster and spiSlave.

【SPIClass】

SPIClass

- SPIClass(uint8_t type)
- SPIClass(void)
- void begin(void)
- void end(void)
- uint8_t transfer(uint8_t data)
- uint16_t transfer16(uint16_t data)
- void beginTransaction(SPISettings settings)
- void endTransaction(void)

Remarks:

This SPI class in version 1.0.3 was created to be compatible with Arduino SPI class. The is a pre-instantiated SPI object: SPI.

【TwoWire_MasterSlave】

TwoWire_MasterSlave

- TwoWire_MasterSlave (uint8_t mode)
- TwoWire_MasterSlave (void)
- void config(uint32_t clkRate)
- void reset(void)
- void resetTx(void)
- void resetRx(void)
- void begin(void)

- void begin(uint8_t addr)
- void setTransmitDeviceAddr(uint8_t addr)
- void setReceiveDeviceAddr(uint8_t addr)
- uint16_t endTransmission(void)
- uint16_t endTransmission(boolxmitStop)
- uint16_t readDevice (uint16_t quantit)
- uint16_t readDevice (uint16_t quantity, boolxmitStop)
- uint16_t readDeviceFromOffset(uint8_t devOffset, uint16_t quantity, boolxmitStop)
- size_t write(uint8_t data)
- size_t write(uint8_t *data, size_t size)
- size_t writeAtOffset(uint8_t devOffset, uint8_t *data, size_t size)
- int available(void)
- int read(void)
- int peek(void)
- void flush(void)
- void onReceive(void (*)(void))
- void onRequest(void (*)(void))
- void isr(void)
- uint8_t requestFrom(uint8_t address, uint8_t quantity)
- uint8_t requestFrom(uint8_t address, uint8_t quantity, uint8_t sendStop)
- uint8_t beginTransmission (uint8_t address)

Remarks:
TwoWire in version 1.0.0 has been moved to TwoWire_MasterSlave. The two pre-instantiated TwoWire objects remain the same: twMaster and twSlave.

【TwoWire】

TwoWire

- TwoWire(void)
- void config(uint32_t clkRate)
- void reset(void)
- void resetTx(void)
- void resetRx(void)
- void begin(void)
- void begin(uint8_t addr)
- void setTransmitDeviceAddr(uint8_t addr)
- void setReceiveDeviceAddr(uint8_t addr)
- uint16_t endTransmission(void)
- uint16_t endTransmission(boolxmitStop)
- uint16_t readDevice (uint16_t quantit)
- uint16_t readDevice (uint16_t quantity, boolxmitStop)
- uint16_t readDeviceFromOffset(uint8_t devOffset, uint16_t quantity, boolxmitStop)
- size_t write(uint8_t data)
- size_t write(uint8_t *data, size_t size)
- size_t writeAtOffset(uint8_t devOffset, uint8_t *data, size_t size)
- int available(void)
- int read(void)
- int peek(void)
- void flush(void)
- void onReceive(void (*)(void))
- void onRequest(void (*)(void))
- void isr(void)
- uint8_t requestFrom(uint8_t address, uint8_t quantity)
- uint8_t requestFrom(uint8_t address, uint8_t quantity, uint8_t sendStop)
- uint8_t beginTransmission (uint8_t address)

Remarks:
TwoWire currently was a duplicate of TwoWire_MasterSlave and was declared in Wire.h which was compatible with Arduino Wire library and header filename. One pre-instantiated TwoWire object has been defined in Wire.cpp.

【HardwareSerial】

HardwareSerial

- HardwareSerial(uint8_t port)

- void config(uint8_t word_length, uint8_t stop_bit, uint8_t party_check)
- void begin(uint32_t baudrate)
- void end(void)
- int available(void)
- int read(void)
- int peek(void)
- void flush(void)
- size_t write(uint8_t value)
- size_t write(uint8_t *buffer, size_t size)
- size_t print(const char str[])
- void isrRx(void)
- void taskTx(void)

【TIMER】

TIMER

- TIMER(uint8_t tmrId)
- TIMER(void)
- bool isIdle(void)
- void isr(void)
- uint8_t every(uint32_t period, void (*callback)(void)) // period is in unit of ms
- uint8_t every(uint32_t period, void (*callback)(void), uint16_t repeatCount)
- uint8_t after(uint32_t period, void (*callback)(void))
- bool expire(void)
- void stop(void)
- uint16_t remainTimes(void)

【Analog】

- void analogPWMPeriod(uint8_t pin, uint32_t value)
- void analogPWMFreq(uint8_t pin, uint32_t value)
- analogWrite(uint8_t pin, uint16_t value)
- void analogADCClock(uint8_t pin, uint32_t value)
- uint16_t analogRead(uint8_t pin)

【Digital】

- unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
- void pinMode(uint8_t pin, uint8_t mode)
- void digitalWrite(uint8_t pin, uint8_t val)
- int digitalRead(uint8_t pin)
- void attachInterrupt(uint8_t pin, void (*taskFunc)(void), int mode)
- void detachInterrupt(uint8_t pin)
- void interrupts(void)
- void noInterrupts(void)
- void hwISRFunc(void)

【SDClass】

SDClass

- SDClass
- boolean begin()
- boolean exist(const char *filepath)
- boolean mkdir(const char *filepath)
- File open(const char *filepath, uinit_8 mode)
- boolean remove(const char *filepath)
- Boolean rmdir(const char * filepath)

【File】

File

- int available()
- void close()
- void flush()
- int peek()
- uint32_t position()
- size_t print(const char* str)
- size println(const char* str)
- boolean seek(uint32_t pos)
- uint32size()
- int read()
- size_t write(uint8_t)

Main.CPP

When NavSpark starts, it first executes all necessary initializations for LEON3 processor and then jump to the C/C++ entry point "main()". The complete code for "main()" can be found in the "main.cpp" along with other source files in directory "hardware/arduino/leon/cores/arduino".

As shown below, more initializations for GNSS is done in "init()" which is defined in "wiring.c", followed by executing "setup()" defined by user for their application in sketch. In the following for-loop, the "gnss_process()" will be executed and it will return true once every update rate interval. For example, if NavSpark is configured to update GNSS at 5Hz, the "gnss_process()" may be called many times each second but it returns true every 0.2 seconds.

Since "gnss_process()" is a library API call and user can NOT modify, NavSpark provides a pre-defined weak function "task_called_after_GNSS_update()" for user to do what they want after every update rate interval. Users can implement their "task_called_after_GNSS_update()" in sketch to overwrite this function (please see examples in the later section for more details). User code may also be added in "loop()" which is defined in sketch.

```
int main(void)
{
	init(); // defined in "wiring.c"
	setup(); // defined in sketch

	for (;;) {
		if (gnss_process())
		{
			// add necessary code which must be executed every update here
			task_called_after_GNSS_update();
		}
		gnss_uart_process();
		loop();// defined in sketch
	}
	gnss_close();

	return 0;
}
```

NavSparksupports GNSS update rate of 1/2/4/5/8/10Hz, default 1Hz. To change this setting, user must edit the code in the file "wiring.c" as show in below; the possible values are defined in "sti_sdk_lib.h".

```
init_mode_p->position_update_rate=STGNSS_POSITION_UPDATE_RATE_1HZ;
```

13. Examples

For demo purpose, several examples are provided for user to learn quickly how to use the NavSpark development board. Users can download the examples from the same resource page where NavSpark Arduino IDE is downloaded. Below are simple descriptions for the examples.

NavSpark Example Compatibility Chart

Example Name	NavSpark	NavSpark-BD	NavSpark-GL	NavSpark-mini
demo_adc_sample	Compatible	Compatible	Compatible	Not Compatible
demo_fractal_on_graphic_lcd_Nokia5110	Compatible	Compatible	Compatible	Partially Compatible*
demo_hello_world	Compatible	Compatible	Compatible	Partially Compatible*
demo_hello_world_nmea	Compatible	Compatible	Compatible	Compatible
demo_how _to_extract_gps_info	Compatible	Compatible	Compatible	Partially Compatible*
demo_how_to_use_timestamp_trig	Compatible	Compatible	Compatible	Not Compatible
demo_interrupt	Compatible	Compatible	Compatible	Partially Compatible*
demo_pwm_freq_400KHz_20_percent	Compatible	Compatible	Compatible	Not Compatible
demo_show_running_time_since_boot	Compatible	Compatible	Compatible	Partially Compatible*
demo_spi_master and demo_spi_slave	Compatible	Compatible	Compatible	Partially Compatible*
Demo_star_time_on_7segment_display	Compatible	Compatible	Compatible	Not Compatible
demo_timer	Compatible	Compatible	Compatible	Partially Compatible*
demo_two_wire_master and demo_two_wire_slave	Compatible	Compatible	Compatible	Partially Compatible*

Partially Compatible* - The example code needs to be changed in order to make it to work.

13.1 ADC Sample

- **demo_adc_sample**
  - This example is a demo for how to use the internal ADC.

```
#include "sti_gnss_lib.h"

void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  analogADCClock(A0, 20000000); // sample freq. = 20MHz

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(115200);
}

void loop() {
  // put your main code here, to run repeatedly:

}

void task_called_after_GNSS_update(void)
{
  int len;
  char buf[64];
  uint16_t adc_in;

  adc_in = analogRead(A0);
  len = sprintf(buf, "ADC = %4d, %1.2f volts\r\n", adc_in, (adc_in*1.0/1023*3.3));
  Serial.print(buf);
}
```

13.2 Fractal on Graphic LCD Nokia 5110

- **demo_fractal_on_graphic_lcd_Nokia5110**
  - NavSpark drawing Mandelbrot

```
#include "wiring_shift.h"
#include "sti_gnss_lib.h"
#include "Nokia_5110.h"

/*
  NOTE: Configure GPIO pins. Note that a 1KOhm resistor is needed between
      LED pins of NavSpark module and Nokia5110.
*/

void setup() {
```

```
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  pinMode(PCD8544_LED_PIN, OUTPUT);
  pinMode(PCD8544_SCE_PIN, OUTPUT);
  pinMode(PCD8544_DC_PIN, OUTPUT);
  pinMode(PCD8544_SCLK_PIN, OUTPUT);
  pinMode(PCD8544_SDIN_PIN, OUTPUT);
  pinMode(PCD8544_RES_PIN, OUTPUT);

  digitalWrite(PCD8544_LED_PIN, HIGH);
  digitalWrite(PCD8544_SCE_PIN, HIGH);
  digitalWrite(PCD8544_DC_PIN, HIGH);
  digitalWrite(PCD8544_SCLK_PIN, LOW);
  digitalWrite(PCD8544_SDIN_PIN, LOW);
  digitalWrite(PCD8544_RES_PIN, HIGH);

  // init the LCD controller
  pcd8544Init();
}

void loop() {
  // put your main code here, to run repeatedly:
  static unsigned char fractal_on = 0;
  static uint32_t cntr = 0;

  // repeat "plot fractal and clean"
  if (cntr == 0) {
    cntr = 5;
    if (!fractal_on) {
      pcd8544FractalPlot();
      fractal_on = 1;
    }
    else {
      pcd8544CleanScreen();
      fractal_on = 0;
    }
  }
  else {
    delay(100);
    cntr --;
  }
}
```

13.3 Hello World

- **demo_hello_world**
  - The following example prints “Hello world” and toggle GPIO 13 each second.

```
uint8_t led = 13;
```

```
void setup() {
  GnssConf.init();
  // put your setup code here, to run once:
  pinMode(led, OUTPUT);
  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(9600);
}

void loop()
{
  // put your main code here, to run repeatedly:

}

void task_called_after_GNSS_update(void)
{
  static uint8_t val = HIGH;
  char buf[32];

  if (val == HIGH) {
    digitalWrite(led, HIGH);
    sprintf(buf, "Hello world (GPIO%02d high)\r\n", led);
    Serial.print(buf);
    val = LOW;
  }
  else if (val == LOW) {
    digitalWrite(led, LOW);
    sprintf(buf, "Hello world (GPIO%02d low)\r\n", led);
    //Serial.print(buf);
    val = HIGH;
  }
}
```

13.4 How to Extract GPS Information

- **demo_how_to_extract_gps_info**
  - This example shows how to get GNSS information and send over UART2. A predefined object “GnssInfo” belongs to C++ class GNSS is provided for extracting all GNSS data. Please see list of member functions for more information.

```
#include "sti_gnss_lib.h"
#include "GNSS.h"

/*
  NOTE: To initialize the UART console port with baud rate 19,200
*/
void setup() {
  // put your setup code here, to run once:
  GnssConf.setNavMode(STGNSS_NAV_MODE_AUTO);
  GnssConf.setUpdateRate(STGNSS_POSITION_UPDATE_RATE_1HZ);
```

```
  GnssConf.setDopMaskMode(STGNSS_DOP_MASK_AUTO);
  GnssConf.setPdopMask(30.0);
  GnssConf.setHdopMask(30.0);
  GnssConf.setGdopMask(30.0);
  GnssConf.init(); /* do initialization for GNSS */

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(19200);
}

void loop() {
  // put your main code here, to run repeatedly:

}

/*
  NOTE: "task_called_after_GNSS_update()" will be called about every second
       (for 1Hz update rate), so we display the info. here.
*/
void task_called_after_GNSS_update(void)
{
  char buf[64];
  uint16_t prnList[STGNSS_GPS_NCHAN];

  // Get info. of setallite
  GnssInfo.update();
  if (GnssInfo.isUpdated() == true) {
    Serial.print("\r\n");

    // display the date from GNSS
    GnssInfo.date.formatString(buf);
    Serial.print(buf);
    Serial.print(", ");
    // display the time from GNSS
    GnssInfo.time.formatString(buf);
    Serial.print(buf);
    Serial.print("\r\n");

    // display how many satellites are in the sky and how many of them are
    // used for position fix
    sprintf(buf, "NumGPSInView = %2d, NumGPSInUse = %2d\r\n",
      GnssInfo.satellites.numGPSInView(prnList),
      GnssInfo.satellites.numGPSInUse(NULL));
    Serial.print(buf);

    // an example to show the info. of 1st satellite in prnList
    sprintf(buf, "Satellite %d: elv = %2d, azi = %3d, CNR = %d\r\n",
       prnList[0],
      GnssInfo.satellites.elevation(CONSTELLATION_GPS, prnList[0]),
      GnssInfo.satellites.azimuth(CONSTELLATION_GPS, prnList[0]),
      GnssInfo.satellites.CNR(CONSTELLATION_GPS, prnList[0]));
    Serial.print(buf);
```

```
  // display the longitude
  GnssInfo.location.longitude_formatString(buf);
  Serial.print(buf);
  Serial.print(", ");
  GnssInfo.location.latitude_formatString(buf);
  Serial.print(buf);
  Serial.print(", ");

  // display the altitude in meters
  sprintf(buf, "height = %.2f(m), ", GnssInfo.altitude.meters());
  Serial.print(buf);

  // display the course in degree
  sprintf(buf, "course = %.2f (deg), ", GnssInfo.course.deg());
  Serial.print(buf);

  // display the speed in KM per hour
  sprintf(buf, "speed = %.2f (km/s)\r\n", GnssInfo.speed.kph());
  Serial.print(buf);
 }
}
```

13.5 How to Use Time Stamp Trigger

- **demo_how_to_use_timestamp_trig**
  - This example shows user how to use the time-stamp trig mechanism provided by NavSpark. Every time a pulse appears on the pin GPIO10, NavSpark will generate a timestamp to record the time and user may use this timestamp for their application such as taking photograph with time mark.

```
#include "sti_gnss_lib.h"
#include "GNSS.h"

/*
 NOTE: To initialize the UART console port with baud rate 19,200
*/

void setup() {
 // put your setup code here, to run once:
 GnssConf.init(); /* do initialization for GNSS */

 Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
 Serial.begin(19200);

 GnssInfo.timestamp.setTrigCapture(TS_TRIG_ON, TS_TRIG_RISING, (void*)userFuncforTimeStamp);
}

void loop() {
 char buf[64];
```

```
  // for demo purpose, we just print out those timestamps on console
  while (GnssInfo.timestamp.numRecord() != 0) {
    if (GnssInfo.timestamp.pop() == true) {
      GnssInfo.timestamp.convertTimeStampToUTC();
      GnssInfo.timestamp.formatGPSString(buf);
      Serial.print(buf);
      Serial.print("; ");
      GnssInfo.timestamp.formatUTCString(buf);
      Serial.print(buf);
      Serial.print("\r\n");
    }
  }
}

/* NOTE: Since this callback function will occupy the time slot for GNSS interrupt,
*       user should NOT do complex jobs here or otherwise the NavSpark may fail to
*       lock GNSS.
*/
void userFuncforTimeStamp(TIME_STAMPING_STATUS_T ts) {
  if (ts.time_is_valid) {
    if (GnssInfo.timestamp.push(ts) == true) {
      // add necessary code for successful data push here, but note the more
      // code added, the higher possibility to loss GNSS lock
    }
  }
}
```

13.6 Interrupt

- **demo_interrupt**
  - This example shows how to configure user-defined function tohandle interrupt triggered by GPIO pins.

```
/*
  NOTE: In this demo, the GPIO 3/6 are wired to GPIO 14/10, the program will assert
        the GPIO 3 and 6 to high level every 3 and 7 seconds, it will trig the ISRs
        registered by users and ISR will deassert the GPIO 3 and 6.
*/

#include "wiring_intr.h"
#include "sti_gnss_lib.h"

#define INTR1_OUT_PIN 3
#define INTR1_IN_PIN  14
#define INTR2_OUT_PIN 6
#define INTR2_IN_PIN  10

/*
  NOTE: 1. Set the modes of GPIO pins,
        2. deassert GPIO 3 and 6,
        3. register ISR functions.
```

```
*/
void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  pinMode(INTR1_OUT_PIN, OUTPUT);
  pinMode(INTR2_OUT_PIN, OUTPUT);
  pinMode(INTR1_IN_PIN, INPUT);
  pinMode(INTR2_IN_PIN, INPUT);
  digitalWrite(INTR1_OUT_PIN, LOW);
  digitalWrite(INTR2_OUT_PIN, LOW);
  attachInterrupt(INTR1_IN_PIN, gpio_intr1_task, RISING);
  attachInterrupt(INTR2_IN_PIN, gpio_intr2_task, RISING);

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin();
}

void loop() {
  // put your main code here, to run repeatedly:

}

/*
  NOTE: Task to assert GPIO 3 and 6 repeatedly.
*/
void task_called_after_GNSS_update(void)
{
  static uint16_t cntr = 0;

  Serial.print("task_called_after_GNSS_update() entered!!\r\n");
  cntr ++;
  if ((cntr%3)==0) { digitalWrite(INTR1_OUT_PIN, HIGH); }
  if ((cntr%7)==0) { digitalWrite(INTR2_OUT_PIN, HIGH); }
}

/*
  NOTE: ISRs defined by users
*/
void gpio_intr1_task() {
  digitalWrite(INTR1_OUT_PIN, LOW);
  Serial.print("GPIO ISR-1 called!!\r\n");
}

void gpio_intr2_task() {
  digitalWrite(INTR2_OUT_PIN, LOW);
  Serial.print("GPIO ISR-2 called!!\r\n");
}
```

13.7 PWM Frequency 400 KHz with 20 Percent Duty Cycle

- **demo_pwm_freq_400KHz_duty_20_percent**

- This example shows how to control the pin “GPIO20” to generate a 400kHzPWM waveform with 20% duty cycle.

```
void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  analogPWMPeriod(20, 2500); // 400KHz => one cycle is 2,500ns
  analogWrite(20, 51); // 255 * 20% = 51
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

13.8 Show Running Time Since Boot

- demo_show_running_time_since_boot
  - “millis()” function is called every time after GNSS updated to display on UART2 the time since the NavSpark began running program. “GPIO2” is UART2 Tx output.

```
#include "sti_gnss_lib.h"

/*
  NOTE: To initialize the UART console port with baud rate 115,200
*/
void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */
  Serial.begin(115200);
}

void loop() {
  // put your main code here, to run repeatedly:

}

/*
  NOTE: "task_called_after_GNSS_update()" will be called about every 1 second in
      case of update rate is 1Hz, so we display the running time here.
*/
void task_called_after_GNSS_update(void)
{
  char buf[64];
  unsigned long bootTimeInMilliSecond;
  unsigned long bootTimeHour;
  unsigned long bootTimeMin;
  unsigned long bootTimeSecond;
```

```
  // call built-in function to get the time since booting
  bootTimeInMilliSecond = millis();
  // process the time and format it to UART
  bootTimeSecond = bootTimeInMilliSecond / 1000;
  bootTimeInMilliSecond = bootTimeInMilliSecond % 1000;
  bootTimeMin = bootTimeSecond / 60;
  bootTimeSecond = bootTimeSecond % 60;
  bootTimeHour = bootTimeMin / 60;
  bootTimeMin = bootTimeMin % 60;

  sprintf(buf, "\r\n%02d(hours):%02d(mins):%02d.%03d(secs) after booting ...\r\n",
    bootTimeHour, bootTimeMin, bootTimeSecond, bootTimeInMilliSecond);
  Serial.print(buf);
 }
```

13.9 SPI Master and Slave

- **demo_spi_master and demo_spi_slave**
  - This example shows how to use the SPI master and slave. To run this example, user needs two NavSpark boards and connects the pins “GPIO28/29/30/31” of board A as master to pin “GPI28/29/30/31” of board B as slave in the order. The master and slave will continuously output 7 bytes of changing data to each other, and both will display what they receive on UART2.

demo_spi_master

```
/*
  NOTE: This code configures NavSpark to work as a SPI master and read/wrte
      data from/to remote SPI slave using another NavSpark.
*/

#include "sti_gnss_lib.h"

void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(115200);

  /* case-1: Only 1 external SPI device */
  spiMaster.config(0, 5000000, false, false); // mode 0, 5MHz, CS0 only
  spiMaster.begin();
  spiMaster.slaveSelect(0); // use GPIO28
  pinMode(GPIO22_SPISL_MCS1, OUTPUT); // GPIO22 can be normal GPIO in this case
  pinMode(GPIO6_WHEEL_TIC_MEAS, OUTPUT); // GPIO6 can be normal GPIO in this case
  digitalWrite(GPIO22_SPISL_MCS1, LOW);
  digitalWrite(GPIO6_WHEEL_TIC_MEAS, LOW);


#if 1 /* case-2: 2 external SPI devices */
```

```
  spiMaster.config(0, 5000000, true, false); // mode 0, 5MHz, CS0 and CS1
  spiMaster.begin();
//spiMaster.slaveSelect(0); // use GPIO28
  spiMaster.slaveSelect(1); // use GPIO22
  pinMode(GPIO6_WHEEL_TIC_MEAS, OUTPUT); // GPIO6 can be normal GPIO in this case
  digitalWrite(GPIO6_WHEEL_TIC_MEAS, LOW);
#endif

#if 0 /* case-3: 3 external SPI devices */
  spiMaster.config(0, 5000000, true, true); // mode 0, 5MHz, CS0, CS1 and CS2
  spiMaster.begin();
//spiMaster.slaveSelect(0); // use GPIO28
//spiMaster.slaveSelect(1); // use GPIO22
  spiMaster.slaveSelect(2); // use GPIO6
#endif
}

void loop() {
  // put your main code here, to run repeatedly:

}

#define       CMD_WR_REG0  0x80
#define       CMD_WR_REG1  0x81
#define       CMD_WR_REG2  0x82
#define       CMD_WR_REG3  0x83
#define       CMD_RD_REG0  0xC0
#define       CMD_RD_REG1  0xC1
#define       CMD_RD_REG2  0xC2
#define       CMD_RD_REG3  0xC3
#define       CMD_WR_BUFF  0x88
#define       CMD_RD_BUFF  0xC8

void task_called_after_GNSS_update(void)
{
  short len;
  char buf[64];
  volatile uint8_t reg0 = 0;
  volatile uint8_t reg2 = 0;
  uint8_t k;
  static uint8_t txd[7] = {0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70};

  spiMaster.resetTx();
  spiMaster.resetRx();
  spiMaster.write(CMD_RD_REG0);
  spiMaster.write(0x00); // null data
  spiMaster.transfer(2);
  reg0 = spiMaster.pick(1);

  if (reg0 & 0x2) { // buffer for master write is ready
    // write 0x7 to reg1 for master wants to write 7 bytes of data to buffer
    spiMaster.resetTx();
    spiMaster.resetRx();
```

```
  spiMaster.write(CMD_WR_REG1);
  spiMaster.write(0x7);
  spiMaster.transfer(2);
  // write the 7 bytes data to buffer
  spiMaster.resetTx();
  spiMaster.resetRx();
  spiMaster.write(CMD_WR_BUFF);
  spiMaster.write(txd[0]++);
  spiMaster.write(txd[1]++);
  spiMaster.write(txd[2]++);
  spiMaster.write(txd[3]++);
  spiMaster.write(txd[4]++);
  spiMaster.write(txd[5]++);
  spiMaster.write(txd[6]++);
  spiMaster.transfer(8);
 }

 if (reg0 & 0x4) { // data on slave's buffer for master read is valid
  // write 0x7 to reg2 for master wants to read 7 bytes of data from buffer
  spiMaster.resetTx();
  spiMaster.resetRx();
  spiMaster.write(CMD_RD_REG2);
  spiMaster.write(0x7);
  spiMaster.transfer(2);
  reg2 = spiMaster.pick(1);

  // identify how many bytes are stored in slave's buffer
  if (reg2 <= 7) {
   spiMaster.resetTx();
   spiMaster.resetRx();
   spiMaster.write(CMD_RD_BUFF);
   for (k = 0; k < reg2; k++) {
    spiMaster.write(0x00); // null data
   }
   spiMaster.transfer(); // CMD_RD_BUFF is included
   if (spiMaster.available()) {
    spiMaster.read(); // ignore 1st byte
    Serial.print("\r\nSPI-M recv:\r\n");
    while (spiMaster.available()) {
     if (spiMaster.available() == 1) {
      len = sprintf(buf, "0x%02x", spiMaster.read());
     }
     else {
      len = sprintf(buf, "0x%02x:", spiMaster.read());
     }
     Serial.print(buf);
    }
    Serial.print("\r\n");
   }
  }
 }
}
```

demo_spi_slave

```
/*
  NOTE: This code configures NavSpark to work as a SPI slave and receive/transmit
        data from/to remote SPI master using another NavSpark.
*/

#include "sti_gnss_lib.h"

// NOTE: The NavSpark only supports SPI mode 0 when operation in slave mode !!

void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(115200);

  spiSlave.begin();
}

void loop() {
  // put your main code here, to run repeatedly:

}

void task_called_after_GNSS_update(void)
{
  short len;
  char buf[64];
  uint8_t k;
  static uint8_t cnt = 0;
  static uint8_t txd[7] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77};

  // in case of data from host was received
  if (spiSlave.available()) {
    Serial.print("\r\nSPI-S recv:\r\n");

    while (spiSlave.available()) {
      if (spiSlave.available() == 1) {
        len = sprintf(buf, "0x%02x", spiSlave.read());
      }
      else {
        len = sprintf(buf, "0x%02x:", spiSlave.read());
      }
      Serial.print(buf);
      cnt ++;
    }
    Serial.print("\r\n");
    if (cnt == 7) {
      cnt = 0;
      spiSlave.enableBufferForHostWrite();
    }
```

```
  }

  // in case of data has been transmitted
  if (spiSlave.validBufferForHostRead()) {
    spiSlave.resetTx();
    for (k = 0; k < 7; k++) {
      spiSlave.write(txd[k]++);
    }
    if (spiSlave.copyDataToBufferForHostRead()) {
      spiSlave.enableBufferForHostRead();
    }
  }
}
```

13.10 Star Time on 7 Segment Display

- **demo_star_time_on_7segment_display**
  - This example sends out control/data code over UART2 to control a 7-segment display to show current time from GNSS. User must connect pin “GPIO2” to “RX” pin of the display module. The particular blue serial 7-segment display used can be found at https://www.sparkfun.com/products/11442 and red one can be found at https://www.sparkfun.com/products/11441.

```
#include "sti_gnss_lib.h"
#include "GNSS.h"

/*
  NOTE:
*/
void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  Serial.begin(9600);
  Serial.write(0x76); // Clear display
}

void loop() {
  // put your main code here, to run repeatedly:

}

void task_called_after_GNSS_update() {
  static uint8_t toggle = 0;
  char hourDigit[4];
  char minDigit[4];

  GnssInfo.update();

  // set cursor to left-most
```

```
 Serial.write(0x79);
 Serial.write(0x00);

 if (GnssInfo.time.hour() == 0) {
  Serial.write('0');
  Serial.write('0');
 }
 else {
  // convert GNSS hours to 2 digits
  utoa(GnssInfo.time.hour(), hourDigit, 10);
  if (GnssInfo.time.hour() > 9) {
   Serial.write(hourDigit[0]);
   Serial.write(hourDigit[1]);
  }
  else {
   Serial.write('0');
   Serial.write(hourDigit[0]);
  }
 }

 // toggle the ':'
 toggle = (toggle == 1) ? 0 : 1;
 Serial.write(0x77);
 Serial.write(toggle<<4);

 // convert GNSS minutes to 2 digits
 if (GnssInfo.time.minute() == 0) {
  Serial.write('0');
  Serial.write('0');
 }
 else {
  utoa(GnssInfo.time.minute(), minDigit, 10);
  if (GnssInfo.time.minute() > 9) {
   Serial.write(minDigit[0]);
   Serial.write(minDigit[1]);
  }
  else {
   Serial.write('0');
   Serial.write(minDigit[0]);
  }
 }
}
```

13.11 Timer

- **demo_timer**
  - This example shows user how to setup a timer. Currently, NavSpark provides 3 timers.

```
/*
 NOTE: This example shows how to setup a timer and associated callback function.
*/
```

```
#ifdef __cplusplus
extern "C" {
#endif
extern void isrTimerFunc(void);
#ifdef __cplusplus
}
#endif

void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(115200);

  Serial.print("Go go go !!\r\n");

  Timer0.every(1000, timer0_test, 50); // setup timer 0 for alarm every 1 second with 50 rounds
  Timer1.every(2500, timer1_test, 20); // setup timer 1 for alarm every 2.5 seconds with 20 rounds
//Timer2.every(5000, timer2_test, 10); // setup timer 2 for alarm every 5 seconds with 10 rounds
  Timer2.every(5000, timer2_test); // setup timer 2 for alarm every 5 seconds, never stop
//Timer2.after(5000, timer2_test); // setup timer 2 for alarm every 5 seconds and execute once
}

void timer0_test(void)
{
  char buf[32];
  sprintf(buf, "Timer-0 active, still have %d times!\r\n", Timer0.remainTimes());
  Serial.print(buf);
}

void timer1_test(void)
{
  char buf[32];
  sprintf(buf, "Timer-1 active, still have %d times!\r\n", Timer1.remainTimes());
  Serial.print(buf);
}

void timer2_test(void)
{
  char buf[32];
  sprintf(buf, "Timer-2 active, still have %d times!\r\n", Timer2.remainTimes());
  Serial.print(buf);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

13.12 Two Wire Master and Slave

- **demo_two_wire_master and demo_two_wire_slave**
  - This example shows how to use the two-wire master and slave. To run this example, user must have two NavSpark boards and connect master board-A “GPIO4” to slave board-B “GPIO4”, and the same for “GPIO5”. The master will keep output to the slave, and the slave will also keep output to the master, both boards will display what they receive on UART2.

demo_two_wire_master

```
/*
  NOTE: This code configures NavSpark to work as a two-wire master and read/wrte
        data from/to remote two-wire slave using another NavSpark.
*/

#include "sti_gnss_lib.h"

void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(115200);

  twMaster.config(400000); // 400KHz
  twMaster.begin();
  twMaster.setTransmitDeviceAddr(0x57); // 0x57 is the device address
  twMaster.setReceiveDeviceAddr(0x57); // 0x57 is the device address
}

void loop() {
  // put your main code here, to run repeatedly:

}

void task_called_after_GNSS_update(void)
{
  static uint8_t txdata[] = {0xde, 0xad, 0xbe, 0xef, 0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xaa};
  static char buf[64];
  static uint16_t txPtr = 0;
  static uint16_t txSize = sizeof(txdata);
        uint16_t txNum;
        uint16_t rxSize;
        uint16_t rxNum;
        uint16_t k;
        uint8_t twSlvSts;
        uint8_t twSlvCmd;

  // poll the status of another V822 which is configured as slave
```

```
 twMaster.reset();
 twMaster.readDeviceFromOffset(0x08, 1, true);
 twSlvSts = twMaster.read();
 twSlvCmd = twSlvSts;

 if (twSlvSts & 0x02) { // data from slave is ready to read
  rxSize = (twSlvSts & 0xe0) >> 5;
  rxNum = (rxSize > TWOWIRE_BUFFER_SIZE) ? TWOWIRE_BUFFER_SIZE : rxSize;
  twMaster.resetRx();
  twMaster.readDeviceFromOffset(TWOWIRE_SLAVE_TX_FIFO_BASE, rxNum, true);
  rxNum = twMaster.available();
  if (rxNum) {
   for (k = 0; k < rxNum; k++) {
    sprintf(buf, "2-wire master recvs 0x%2x\r\n", twMaster.read());
    Serial.print(buf);
   }
   twSlvCmd &= ~(1 << 1); // clear bit 1
   twSlvCmd &= ~(7 << 5); // clear bit 7 ~ 5
  }
 }

 if ((twSlvSts & 0x01) == 0) { // xmit data to slave
  if (txSize) {
   txNum = (txSize > TWOWIRE_SLAVE_RX_FIFO_SIZE) ? TWOWIRE_SLAVE_RX_FIFO_SIZE : txSize;
   twMaster.resetTx();
   txNum = twMaster.writeAtOffset(TWOWIRE_SLAVE_RX_FIFO_BASE, &txdata[txPtr], txNum);
   txSize -= txNum;
   txPtr += txNum;
   twSlvCmd &= ~(7 << 2); // clear bit 4 ~ 2
   twSlvCmd |= ((txNum & 0x07) << 2);
   twSlvCmd |= 0x1; // set bit 0
  }
  else {
   txSize = sizeof(txdata);
   txPtr = 0;
  }
 }

 if (twSlvSts != twSlvCmd) {
  twMaster.writeAtOffset(0x08, &twSlvCmd, 1);
 }
}
```

demo_two_wires_slave

```
/*
 NOTE: This code configures NavSpark to work as a two-wire slave and receive/transmit
       data from/to remote two-wire master using another NavSpark.
*/

#include "sti_gnss_lib.h"

uint8_t txdata[] = {0x19, 0x68, 0x05, 0x31, 0x19, 0x70, 0x03, 0x12, 0x19, 0x71, 0x07, 0x14};
```

```
void setup() {
  // put your setup code here, to run once:
  GnssConf.init(); /* do initialization for GNSS */

  Serial.config(STGNSS_UART_8BITS_WORD_LENGTH, STGNSS_UART_1STOP_BITS, STGNSS_UART_NOPARITY);
  Serial.begin(115200);

  twSlave.begin(0x57); // 0x57 is my device address
  twSlave.reset();

  twSlave.write(txdata, 12);
}

void loop() {
  // put your main code here, to run repeatedly:
  static char buf[64];
  static uint16_t i;
  uint8_t k;

  // print out the data received from master
  k = twSlave.available();
  if (k) {
    for (i = 0; i < k; i++) {
      sprintf(buf, "2-wirev slave recvs 0x%2x\r\n", twSlave.read());
      Serial.print(buf);
    }
  }

  // fill the data to be sent
  if (twSlave.remaining() == 0) {
    twSlave.resetTx();
    twSlave.write(txdata, 12);
  }
}
```

14. Recovering from a Hanged Program

Sometimes improperly written user program might hang the system. When this happens, it's not possible to upload new code in the usual way as NavSpark will no longer respond to upload commands from the Arduino IDE. To recover:

1. Connect BOOT_SEL pin to GND to run program off internal ROM memory.
2. Check state of GPIO4 and GPIO5, these two pins determine baud rate used for UART1 (USB) communication when booting from internal ROM memory. See next section on how to set command and code upload baud rate to use for Arduino IDE in case GPIO[5:4] connected to differ from default 2'b11.
   GPIO[5:4] = 2'b00 for 9600
   GPIO[5:4] = 2'b01 for 4800
   GPIO[5:4] = 2'b10 for 38400
   GPIO[5:4] = 2'b11 for 115200 (default)
3. Press the reset button and release it to restart the LEON3 processor.
4. Use Arduino IDE to do the code upload procedure with the correct binary code.
5. Once code upload completes, disconnect BOOT_SEL from GND to run from Flash memory, press the reset button and release it to restart.

15. Setting Command & Code Upload Baud Rate for Arduino IDE

Open the file "boards.txt" under "C:\Users\USER\AppData\Roaming\Arduino15\packages\navspark\hardware\navspark\1.0.0", the values of "upload.cmd_baudrate" and "upload.data_baudrate" sets the baud rate to use by Arduino IDE to send command and upload code to NavSpark. Normally 9600/38400/115200 are recommended for command and 115200/460800 are recommended for code upload. The Arduino IDE must be closed before the modification, and restarted after the modification for it to take effect. Default setting is 115200 for both.

On Windows platform it is possible to have different UART baud rate for sending command and uploading code. When BOOT_SEL is connected to GND running program from ROM, before uploading code from Arduino IDE, "upload.cmd_baudrate" must be set to same baud rate as selected by GPIO4 and GPIO5.

On Linux platform, same baud rate need to be used for sending command and uploading code, set by state of GPIO[5:4].

16. Setting Baud Rate for UART1 NMEA Message

Open the file “boards.txt” under “C:\Users\USER\AppData\Roaming\Arduino15\packages\navspark\hardware\navspark\1.0.0”, modifies the value of “build.baudrate” to the baud rate preferred.

17. UART 1, UART 2 vs NavSpark Boards

NavSpark (GPS/GL/BD) and NavSpark mini firmware can be built with or without GNSS library (GNSS mode or MCU mode). The roles of UART 1 and UART 2 can have several combinations and usages. When NavSpark GPS/GL/BD has GNSS feature, NMEA will be output to the UART 1 through micro USB and UART 2 (TXD2, RXD2) is used as console. When NavSpark GPS/GL/BD has no GNSS feature, UART 1 is not used and UART 2 remains used as Console. When NavSpark mini has GNSS feature, NMEA will be output to the UART 1 through micro USB and since NavSpark mini has no UART 2, no console is available. When NavSpark mini has no GNSS feature, UART is not used and no console is available.

Name	UART 1 (Micro USB)	UART 2
NavSpark (GPS/GL/BD)with GNSS feature	NMEA (GNSS taken)	Serial (User default Console)
NavSpark without GNSS feature	Serial1(open for user)	Serial(User default Console)
NavSpark mini with GNSS feature	NMEA (GNSS taken)	N/A (no default console)
NavSpark mini without GNSS feature	Serial1(open for user)	N/A (no default console)

18. Venus 8 Baseband Memory Map

The memory map of SkyTraq Venus 8 is shown in the table below. The on-chip flash is used for users to place their program codes. The on-chip 192KB of RAM is used for data section and system stack. The 20KB on-chip RAM is reserved for internal use by SkyTraq kernel if built with GNSS features.

Address space	size	Mapping	Cached
0x00000000 ~ 0x000FFFFF	1024KB	On-chip flash	Cacheable
0x50000000 ~ 0x50004FFF	20KB	On-chip RAM	Cacheable
0x60000000 ~ 0x6002FFFF	192KB	On-chip RAM	Cacheable

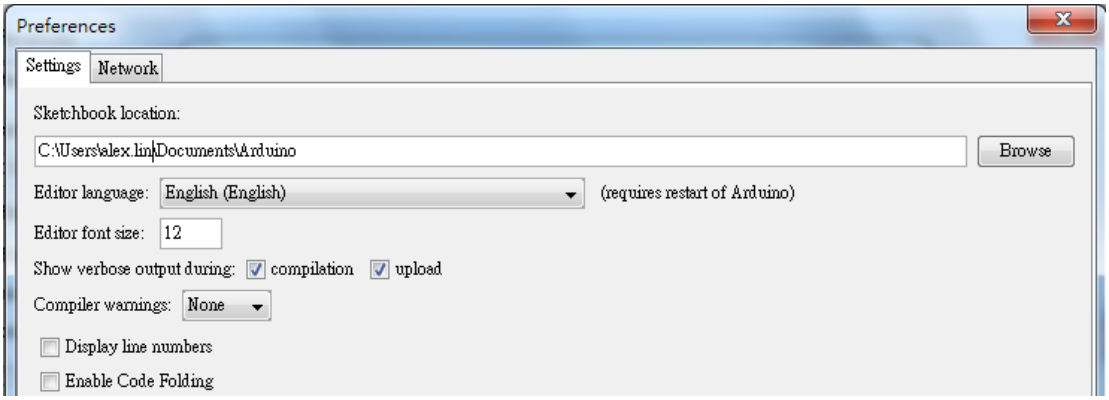
The software layout in the flash is shown below.

Address space	Size	Software layout
0x00000000 ~ 0x000FCFFF	1012KB	Program codes
0x000FD000 ~ 0x000FFFFF	12KB	Internal used by SkyTraq Kernel that is built with GNSS features

The size of program codes in Arduino IDE can be found as “Sketch uses 610,560 bytes (58%) of program storage space. Maximum is 1,048,576 bytes” after you choose “sketch” page and select “Verify / Compile”.

On the other hand, both the size of RAM and program codes can be got by using sparc-elf-size.exe command on prom.out which located in “*%TEMP%*”.

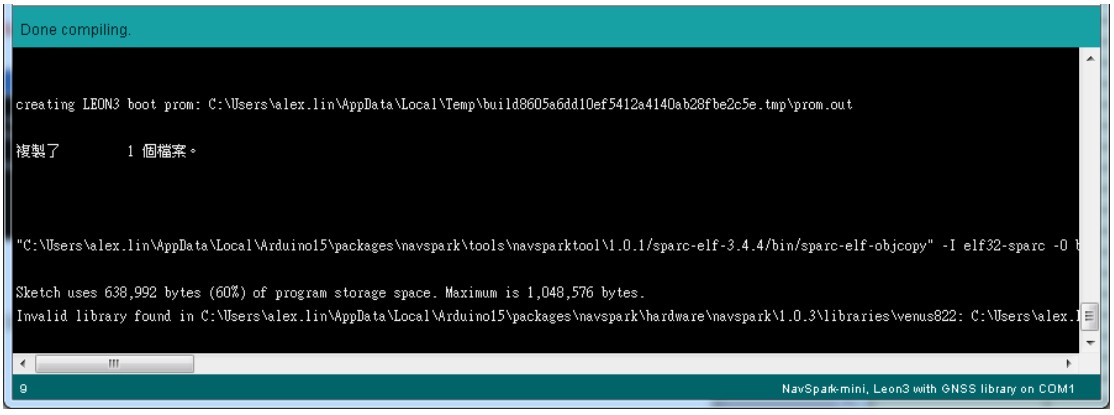
If you turn on the compilation message as below,



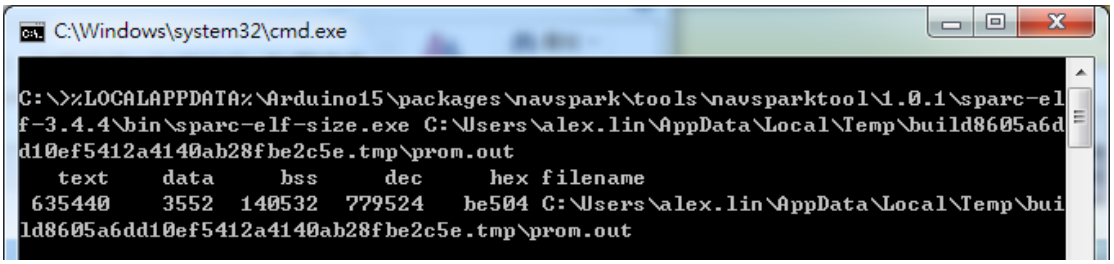
You can find it in Arduino IDE like

creating LEON3 boot prom:

C:\Users\alex.lin\AppData\Local\Temp\build8605a6dd10ef5412a4140ab28fbe2c5e.tmp\prom.out



In DOS window, you can execute %LOCALAPPDATA%\Arduino15\packages\navspark\tools\navsparktool\1.0.1\sparc-elf-3.4.4\bin\sparc-elf-size.exe C:\Users\alex.lin\AppData\Local\Temp\build8605a6dd10ef5412a4140ab28fbe2c5e.tmp\prom.out



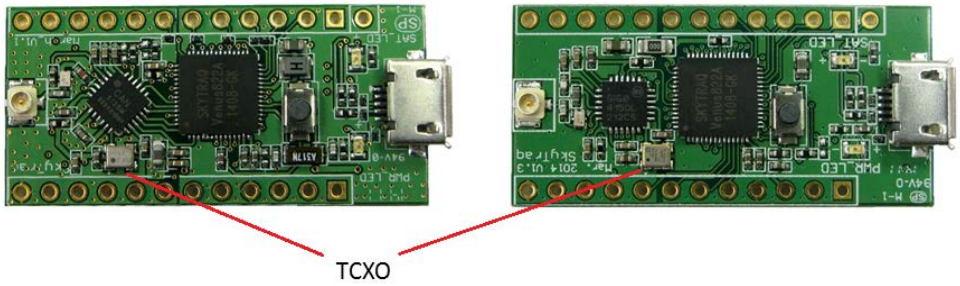
The program size is text size + data size, that is, 635440+3552=638992 bytes. The RAM size is 3552+140532=144084.

19. Usage Suggestions

1. Generally, GPS/GNSS receivers are electrostatic sensitive device, require special precaution when handling. Standard ESD safety practice needs to be applied. Particular care must be taken when connecting the antenna, due to the risk of electrostatic discharge. Below measure should be considered whenever handling NavSpark receiver:
   a. Discharge body electrostatic charge by touching some large ground, or the ground of USB cable connector assuming USB cable is connected to laptop and laptop is connected to AC outlet.
   b. First point of contact when handling NavSpark should always be the PCB ground, take it by micro USB connector.
   c. If antenna used is internal active antenna, avoid touching metal film coating of ceramic patch antenna, as it goes directly to sensitive LNA in the active antenna circuitry.
2. Special care must be applied when connecting antenna to NavSpark. The small U.FL connector, although not fragile, is not as robust as large SMA connectors that could withstand large force. When plugging the antenna connector to NavSpark RF connector, make sure one connector is on top of the other and apply force squarely. When unplugging the antenna connector, make sure force is applied evenly on two-sides to pull up; uneven force might cause the female antenna connector to bend, resulting in later bad connection contact.
3. Due to construct of the U.FL female antenna connector on the active antenna, it has rating of less than 50 plug/unplug durability. The U.FL male receptacle on NavSpark has typical durability rating of 500 plug/unplug cycles. Thus good to avoid frequent plugging/unplugging the antenna. If the antenna is left in place without much plug/unplug, it should last for long time.
4. No hurry to solder the pin sticks immediately upon receiving NavSpark. Download and run GNSS Viewer from Store/Resource page of www.navspark.com.tw . Connect the antenna, and then connect NavSpark to Laptop using USB cable. Testing outdoors at place having clear view of the sky, it should acquire signal and have position fix pretty quickly. After using it for a while, you might have better idea of how you like the connector pins soldered.

5. Three 12-pin pin-sticks are supplied with each NavSpark. When soldering the pin-sticks to NavSpark, it is best to first fitting it onto bread board before soldering, to ensure it fit. Since shorting BOOT_SEL to GND is needed to recover from hanged user program, might be easier to have BOOT_SEL and adjacent GND pin upward.

6. TCXO is temperature sensitive, when there is large temperature change (temperature gradient), GPS/GNSS signal might lose lock temporarily. Avoid temperature changing air flow directly flowing over the TCXO, having some enclosure or wrapping would ensure the performance.



20. Commonly Asked Questions

Q1: Why can't I upload binary code to NavSparkand get message like "Unable to open /dev/ttyUSB0" on GUI?

A1: NavSparkmodule uses USB port to upload binary code to NavSpark and access to the USB port in Linux platform needs root's privilege. You may need to login as root to run Arduino or change the permission of device file "/dev/ttyUSB0" to be readable/writeable by command "chmodo+rw /dev/ttyUSB0".

Q2: Why I get "sparc-elf-g++: installation problem, cannot exec `cc1plus': No such file or directory" when compiling Arduino code on some Vista platforms?

A2: There are two ways to resolve this problem, either can be applied.

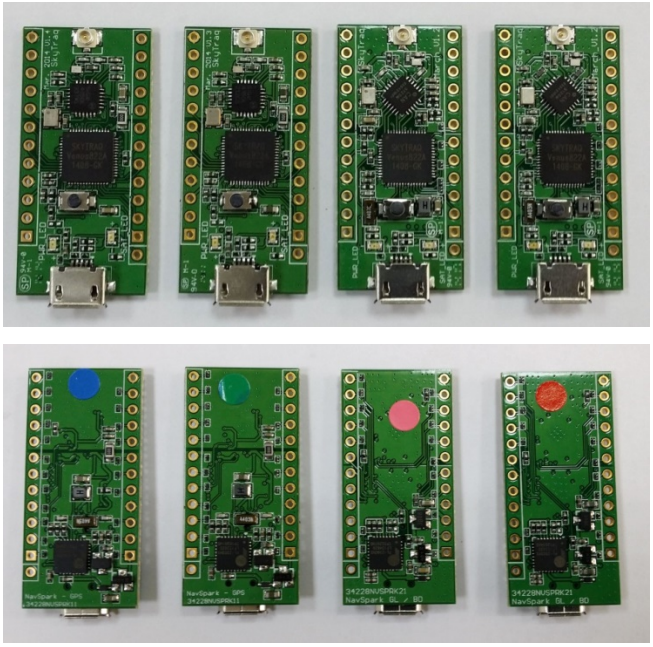
- Add "C:\opt\sparc-elf-3.4.4-mingw\libexec\gcc\sparc-elf\3.4.4" to the PATH system environment variable.
- Copy "cc1.exe" and "cc1plus.exe" under directory "C:\opt\sparc-elf-3.4.4-mingw\libexec\gcc\sparc-elf\3.4.4" to the directory "C:\opt\sparc-elf-3.4.4-mingw\bin".

Q3: Why I get "sparc-elf-g++: installation problem, cannot exec `as': No such file or directory" when compiling Arduino code on some Vista platforms?

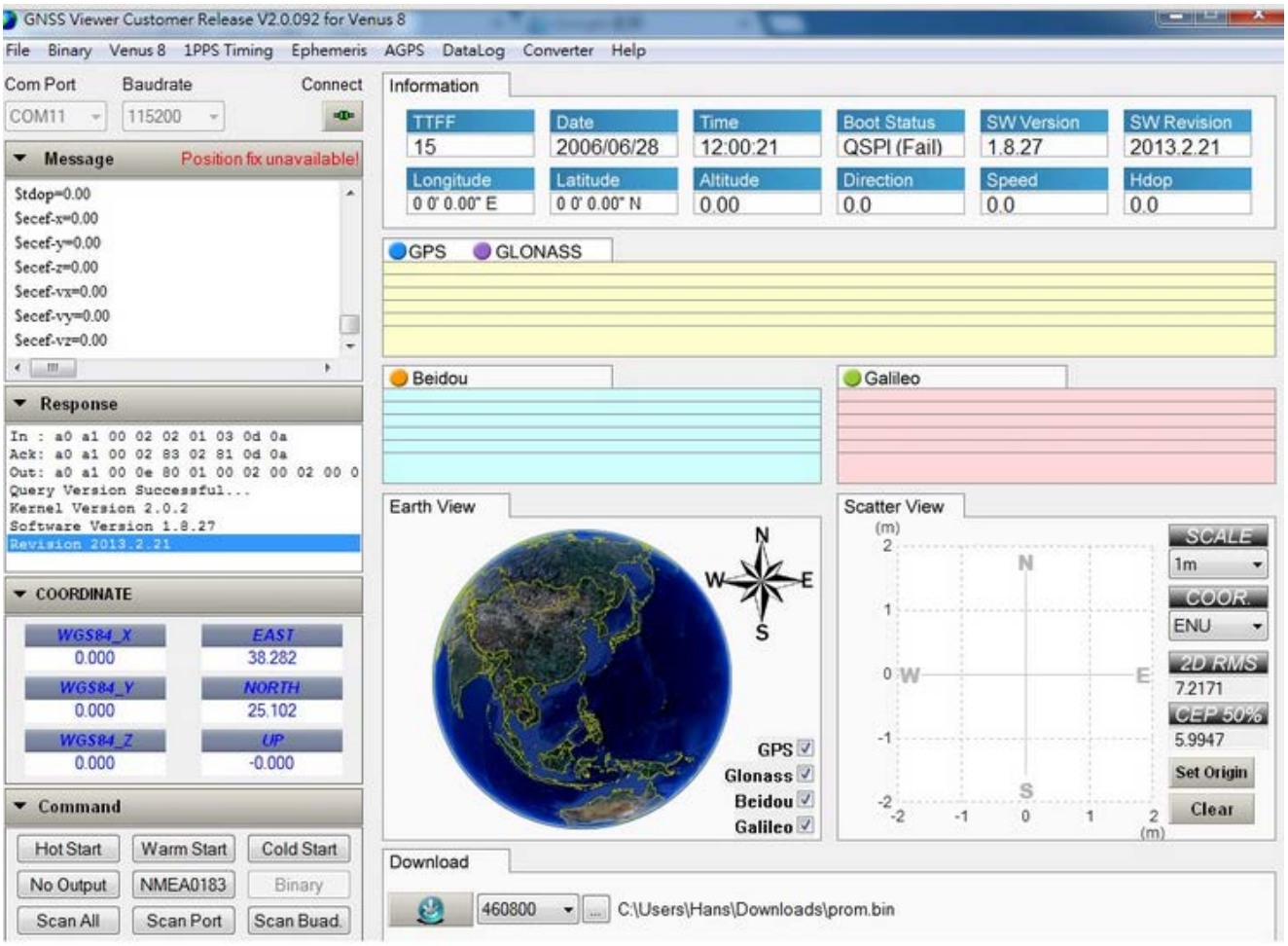
A3: Add "C:\opt\sparc-elf-3.4.4-mingw\sparc-elf\bin" to the PATH system environment variable.

Q4: NS-RAW and NavSpark look alike, NavSpark-BD and NavSpark-GL look alike. How to tell which is which?

A4: NavSpark has blue sticker, NS-RAW has a green sticker, NavSpark-GL has pink sticker, NavSpark-BD has red sticker.



Q5: After uploading example code to my NavSpark-BD, it can no longer catch Beidou signal, firmware revision date became 2013.2.21, and previously it was 2014.4.x. How can I restore back to 2014.4.x?



A5: Whenever seeing 2013.2.21 for SW Revision date in GNSS Viewer, it means you are booting/running NavSpark from internal ROM code, which is dated 2013.2.21; you must have shorted BOOT_SEL to GND. The internal ROM code is GPS-only firmware so you won't see any Beidou signal with NavSpark-BD. With NavSpark-GL, no GPS nor GLONASS signal will be seen when booting/running from ROM code. Booting/running from ROM code is mainly for user to recover from hanged user program, thus not to worry whether ROM code catches signal or not.

To recover to factory shipped firmware state, first get the corresponding shipped firmware from individual NavSpark product page on the NavSpark web store, at bottom of the product page. To download the firmware to NavSpark using GNSS Viewer:

1. Unzip the firmware to a local directory
2. Connect the GNSS Viewer to NavSpark
3. Select the unzipped .bin file using the small "…" button to the right of the Download button

4. Change download baud rate from 460800 to 230400 just to be safe.

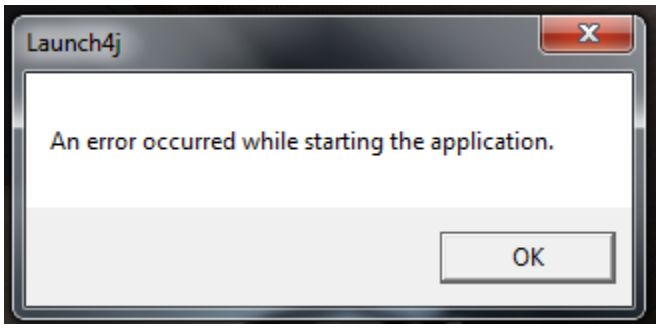
5. Click the Download button, first boot-loader will be downloaded, and then main firmware will be downloaded. After downloading is done, the pop-up window will close and NavSpark will reboot.

6. Leave BOO_SEL open, not shorting it to GND, such that program will run from Flash

7. Press and release the reset button and NavSpark should then run from Flash, running the firmware you just loaded. SW Revision date should show 2014.4.x

We recently found that step #6 ~ #7 is not 100% fail safe, occasionally with power applied disconnecting BOOT_SEL from GND and pressing reset the new loaded code doesn't always run correctly and occasionally QPSI (Fail) shows up on GNSS Viewer; we are still trying to fix this. In step #6, if USB cable is unplugged from PC side to remove power, leave BOOT_SEL open to boot/run from Flash, connect USB cable at PC side to re-apply power, then it'll always runs correctly from newly loaded Flash code.

Q6: When running Windows Arduino IDE, it shows below error message. What to do?



A6: This is mostly related to needing 32bit JAVA on 64bit Windows. Uninstall 64bit JAVA and install 32bit JAVA.

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Java SE Development Kit 8u5

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	133.58 MB	jdk-8u5-linux-i586.rpm
Linux x86	152.5 MB	jdk-8u5-linux-i586.tar.gz
Linux x64	133.87 MB	jdk-8u5-linux-x64.rpm
Linux x64	151.64 MB	jdk-8u5-linux-x64.tar.gz
Mac OS X x64	207.79 MB	jdk-8u5-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	135.68 MB	jdk-8u5-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	95.54 MB	jdk-8u5-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	135.9 MB	jdk-8u5-solaris-x64.tar.Z
Solaris x64	93.19 MB	jdk-8u5-solaris-x64.tar.gz
Windows x86	151.71 MB	jdk-8u5-windows-i586.exe
Windows x64	155.18 MB	jdk-8u5-windows-x64.exe

Q7: I have problem with my first “Hello World” program, what to do?

A7: Please see this tutorial: http://www.navspark.com.tw/blog/tutorial-hello-world

Q8: Why is my NavSpark-mini default ROM baud rate is at 9600?

A8: Some of our previous NavSpark-mini is missing a 4.7K pull-up resistor in GPIO 4 and 5. To solve this issue, connect GPIO 4 and 5 with a 4.7K resistor to VCC33. If you are not using I2C functions or do not have any resistors, connect GPIO 4 and 5 to VCC33 to change the default ROM baud rate to 115200.

Change Log

Version 0.8, December 14, 2015

1. Added SPIClass, TwoWire, SDClass and File
2. Added UART 1, UART 2 vs NavSpark Boards
3. Added Venus 8 Baseband Memory Map

Version 0.7, August 26, 2015

1. Added table of contents
2. Added features, I/O function, and pin out description of NavSpark-mini
3. Added “How to Connect NavSpark-mini with UART-to-USB Adapter Board (for NavSpark-mini)”
4. Modified “Setting Up Arduino IDE for NavSpark” for Arduino IDE 1.6.5
5. Added examples compatibility chart
6. Placed examples in alphabetical order
7. Removed unrelated commonly asked questions for Arduino IDE 1.6.5
8. Added NavSpark-mini default ROM baud rate issue to Commonly Asked Questions Q8/A8

Version 0.6, May 23, 2014

1. Added Arduino sketch compile error problem treatment to Commonly Asked Questions Q8/A8.
2. Added “Hello World” tutorial to Commonly Asked Questions Q9/A9

Version 0.5, May 21, 2014

3. Added Windows 7 Arduino IDE Launch4j problem treatment to Commonly Asked Questions Q6/A6.
4. Added Linux Arduino IDE on Linux 64bit problem treatment to Commonly Asked Questions Q7/Q7.

Version 0.4, May 20, 2014

1. Added ROM recovery mode behavior explanation to Commonly Asked Questions Q5/A5.

Version 0.3, May 15, 2014

1. Modify SPI related description to support 3 SPI slave device

Version 0.2, May 14, 2014

1. Change the Arduino IDE from version 1.5.2 to 1.5.6-r2.
2. Add new class GNSSParam, GNSSTimeStamp.
3. Modify API list.
4. Modify example “demo_how_to_extract_gps_info” for setting GNSS parameters.
5. Add example for “demo_how_to_use_timestamp”
6. Updated images

Version 0.1, March 27, 2014

1. Initial release

The information provided is believed to be accurate and reliable. These materials are provided to customers and may be used for informational purposes only. No responsibility is assumed for errors or omissions in these materials, or for its use. Changes to specification can occur at any time without notice.

These materials are provides “as is” without warranty of any kind, either expressed or implied, relating to sale and/or use including liability or warranties relating to fitness for a particular purpose, consequential or incidental damages, merchantability, or infringement of any patent, copyright or other intellectual property right. No warrant on the accuracy or completeness of the information, text, graphics or other items contained within these materials. No liability assumed for any special, indirect, incidental, or consequential damages, including without limitation, lost revenues or lost profits, which may result from the use of these materials.

The product is not intended for use in medical, life-support devices, or applications involving potential risk of death, personal injury, or severe property damage in case of failure of the product.