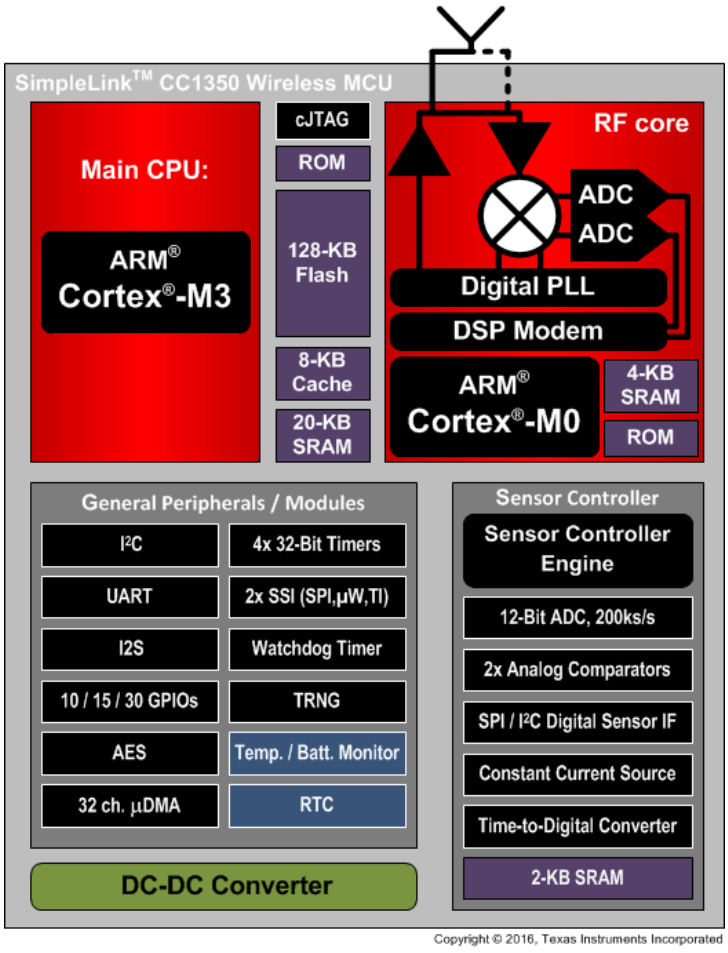
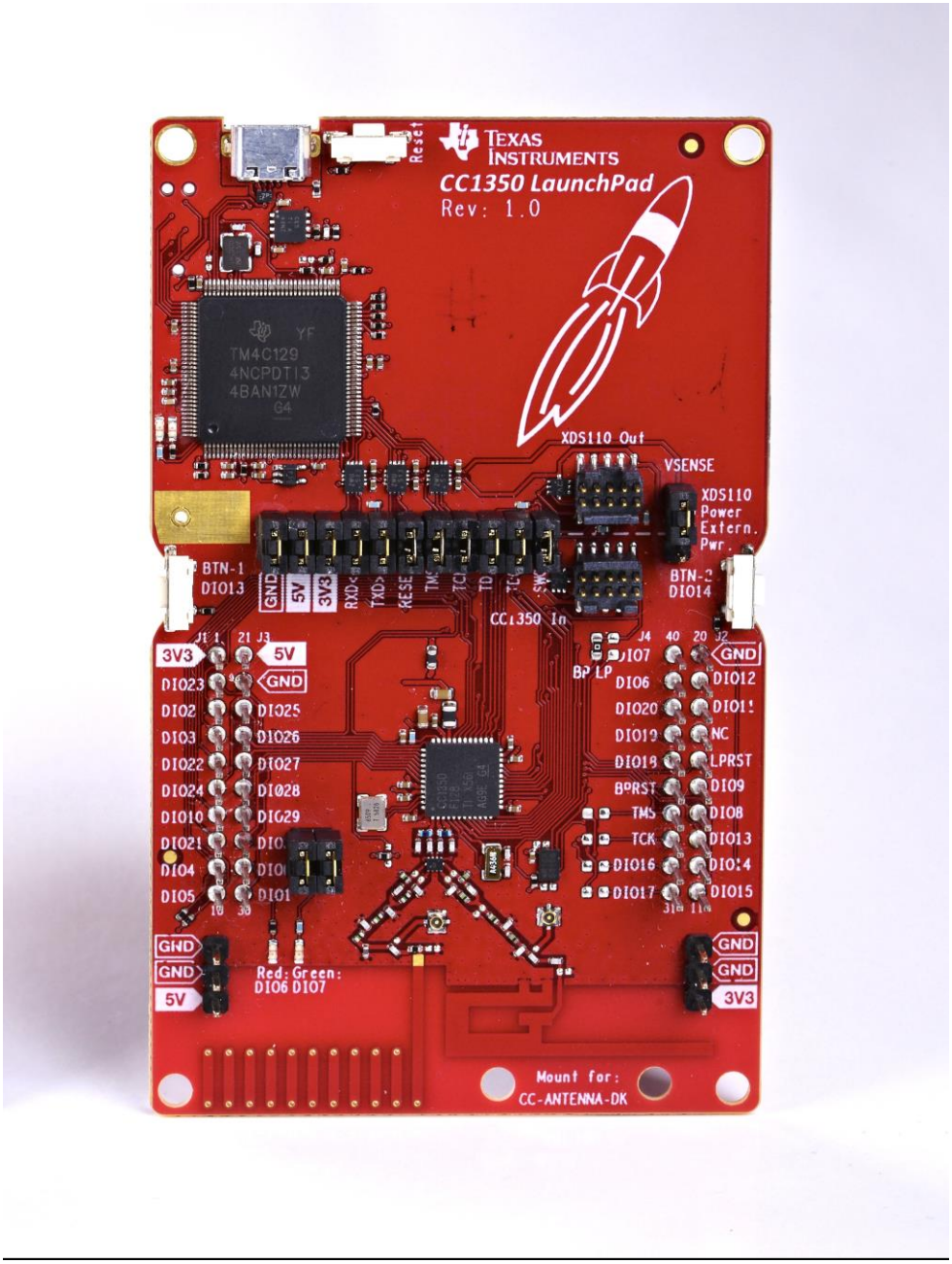
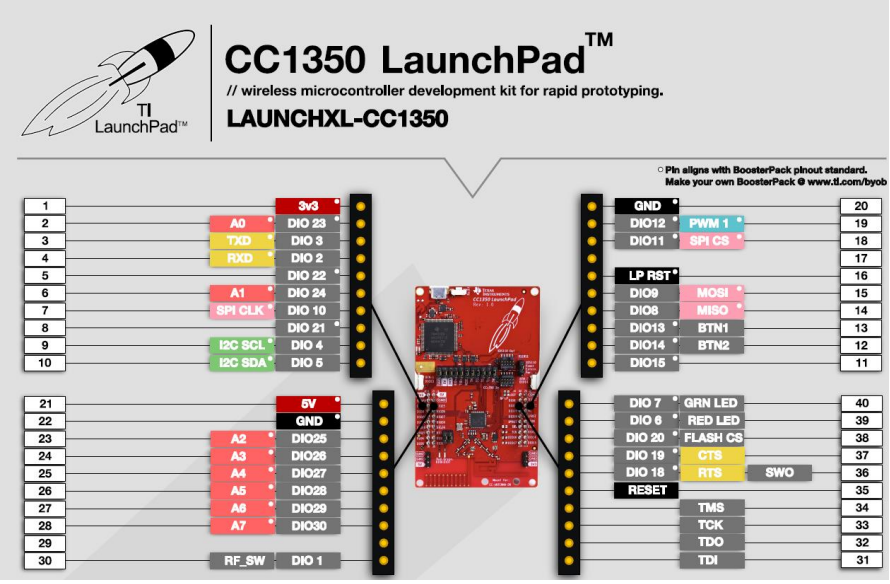
CHAPTER 1 INTRODUCTION TO THE C1350 MICROCONTROLLER AND THE CODE COMPOSER STUDIO 2

CHAPTER 1 Introduction to the C1350 Microcontroller and the Code Composer studio

1.1 Introduction to the C3150




TEXAS INSTRUMENTS
CC1350 LaunchPad
Rev: 1.0
Reset
XDS110 Out
VSENSE
XDS110 Power
Extern. Pwr.
BTN-1
DIO13
BTN-2
DIO14
CC1350 In
3V3
5V
GND
DIO23
DIO2
DIO3
DIO22
DIO24
DIO10
DIO21
DIO4
DIO5
DIO25
DIO26
DIO27
DIO28
DIO29
DIO30
DIO1
DIO7
DIO6
DIO20
DIO19
DIO18
BPRST
TMS
TCK
DIO16
DIO17
DIO12
DIO11
NC
LPRST
DIO9
DIO8
DIO13
DIO14
DIO15
Red: DIO6
Green: DIO7
Mount for:
CC-ANTENNA-DK



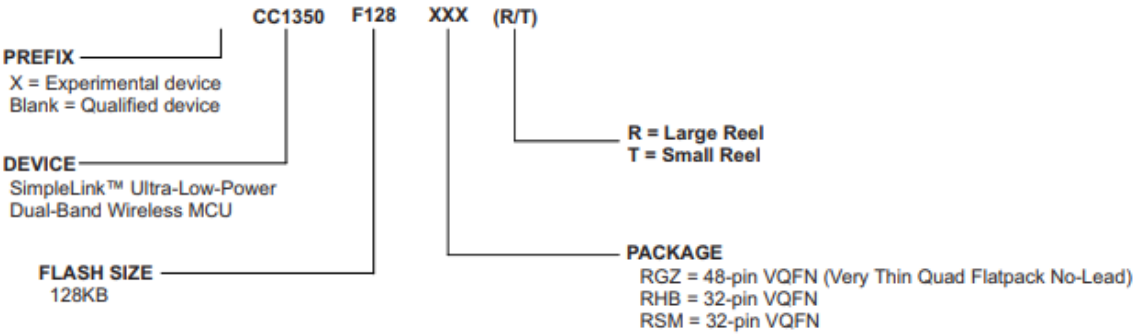
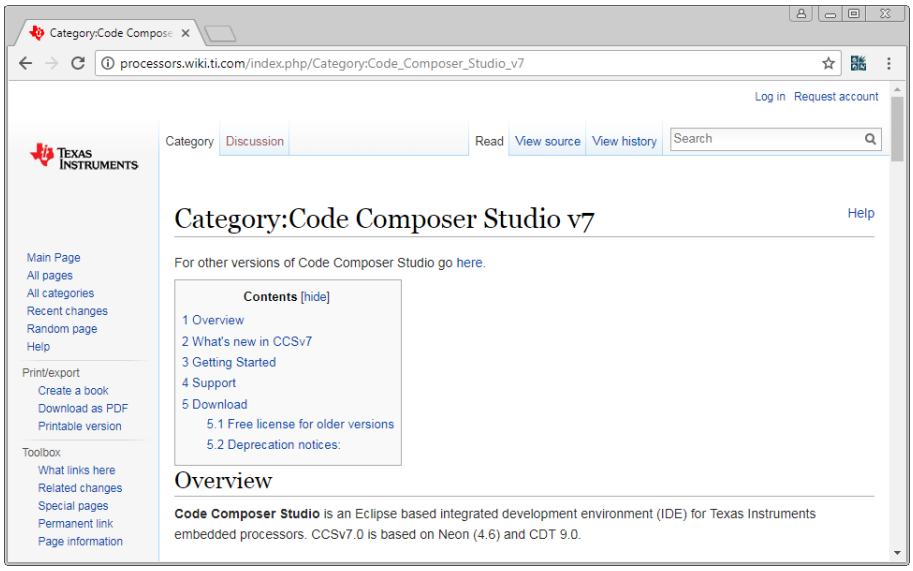


Figure 8-1. Device Nomenclature

http://www.ti.com/lit/ds/symlink/cc1350.pdf

1.2 Introduction to CCS

1.2.1 Install the Code Composer Studio, CCS



http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v7

1.2.2 Install the SDK

simplelink_cc13x0_sdk_1_60_00_21.exe

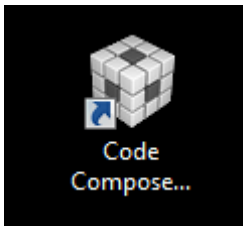
1.2.3 Introduction to CCS

The aim of this laboratory exercise is to become familiar with the tools.

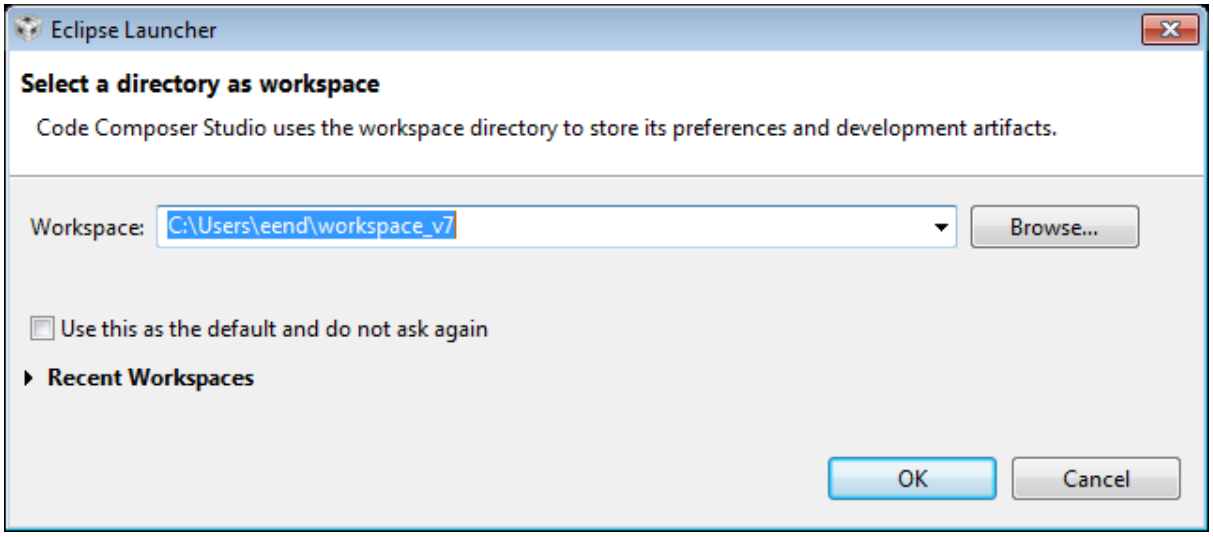
This section shows how to use development tools, create a project and modify the compiler switches in order to achieve the best performance

Starting the experiment

1) Launch CCS by double-clicking on the desktop icon of your PC:



2) Select the directory workspace



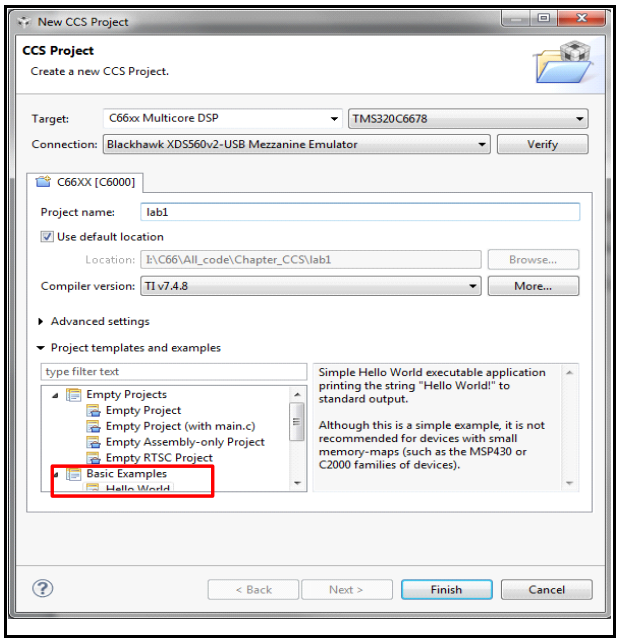
3) Create a Project (NOT REQUIRED IN THIS LAB)

A project stores all the information needed to build an individual program or library, including:

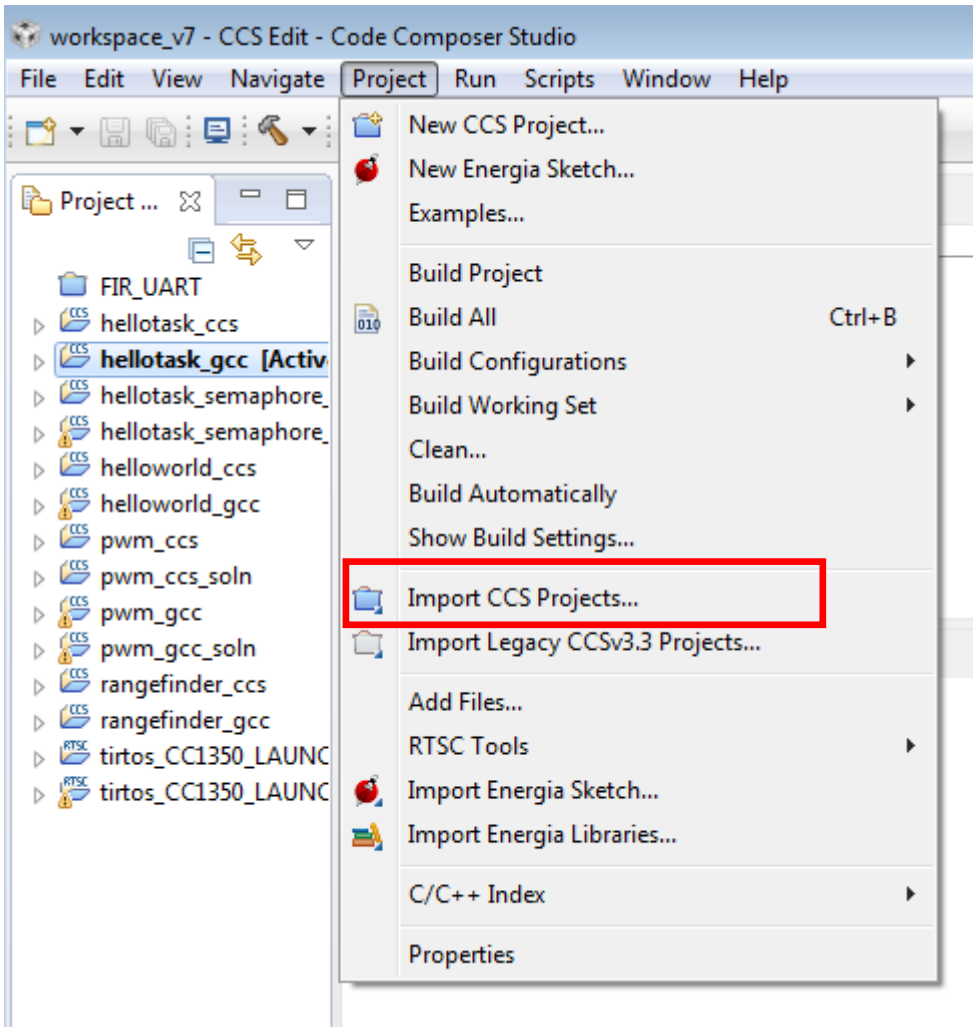
- File names of source code and object libraries.
- Build-tool options.
- File dependencies.

Build-tool version used to build the project.

Select: **File > New > CCS Project** (see Figure below).

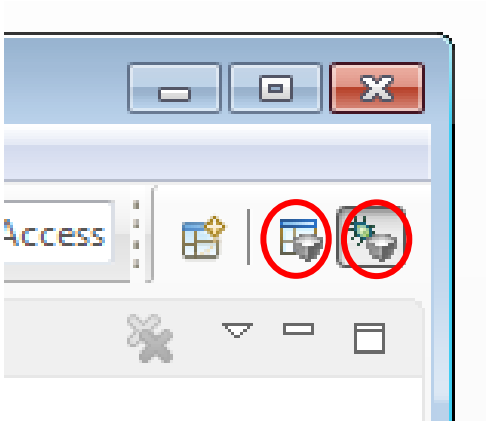


4) Import a project

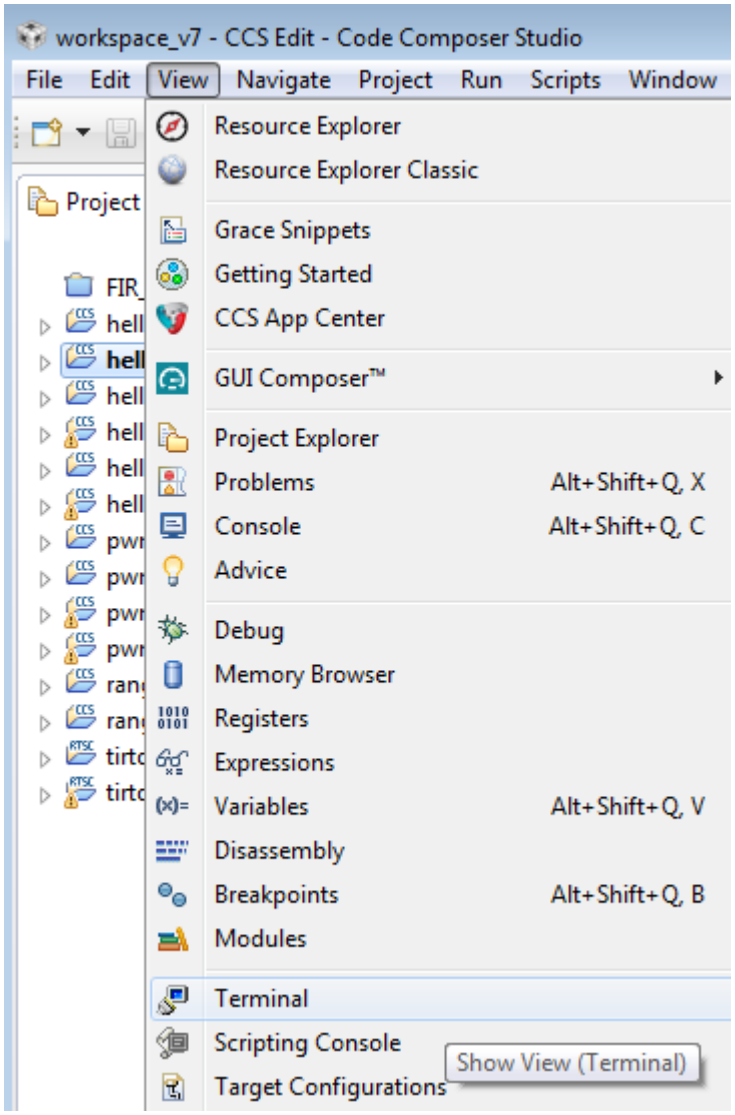


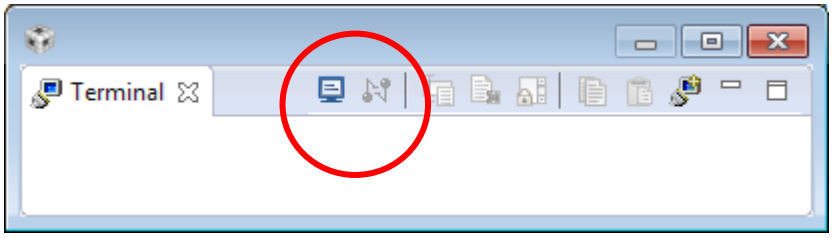
5) Edit Perspective (see Figure below)

Once you create a project, you will have two default perspectives, one for editing (**CCS Edit**) and one for debugging (**CCS Debug**). Make sure you select the appropriate perspective for what you want to do.

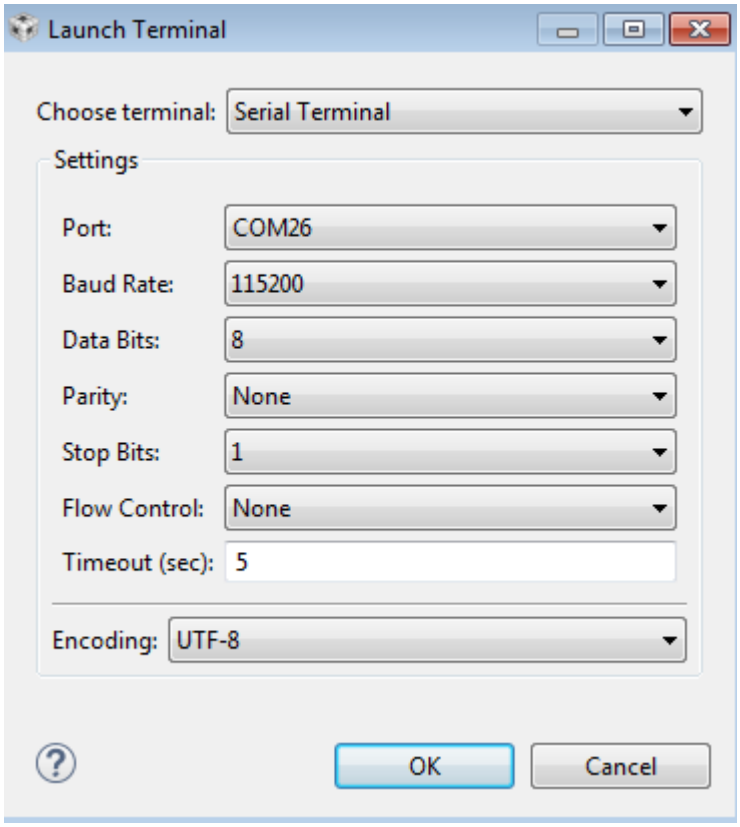


6) Open a Terminal





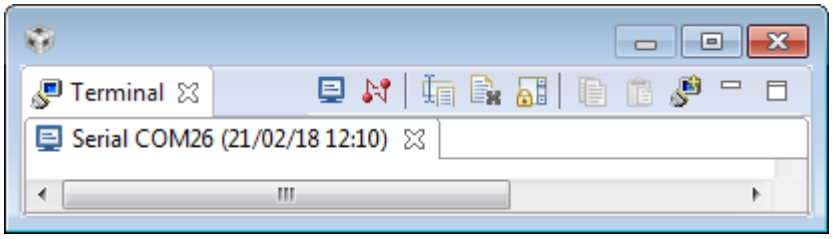
7) Set up a terminal



Unicode Character Set and UTF-8, UTF-16, UTF-32 Encoding

https://naveenr.net/unicode-character-set-and-utf-8-utf-16-utf-32-encoding/

8) Check that the terminal is connected



.

1.3 Pulse Width Modulation (PWM)

```
/*!
 *  @brief PWM Parameters
 *
 *  PWM Parameters are used to with the PWM_open() call. Default values for
 *  these parameters are set using PWM_Params_init().
 *
 *  @sa     PWM_Params_init()
 */
typedef struct PWM_Params_ {
    PWM_Period_Units periodUnits; /*!< Units in which the period is specified */
    uint32_t         periodValue; /*!< PWM initial period */
    PWM_Duty_Units   dutyUnits;   /*!< Units in which the duty is specified */
    uint32_t         dutyValue;   /*!< PWM initial duty */
    PWM_IdleLevel    idleLevel;   /*!< Pin output when PWM is stopped. */
    void            *custom;      /*!< Custom argument used by driver
                                       implementation */
} PWM_Params;
```

```
/*!
 *  @brief   PWM duty cycle unit definitions.  Refer to device specific
 *  implementation if using PWM_DUTY_COUNTS (raw PWM/Timer counts).
 */
typedef enum PWM_Duty_Units_ {
    PWM_DUTY_US,       /*!< Duty cycle in microseconds */
    PWM_DUTY_FRACTION, /*!< Duty as a fractional part of PWM_DUTY_FRACTION_MAX */
    PWM_DUTY_COUNTS    /*!< Duty in timer counts  */
} PWM_Duty_Units;
```

```
//PWM period in microseconds
#define PWM_PER_US 25

//PWM duty cycle in microseconds
#define PWM_INIT_DUTY (PWM_PER_US / 2)
```