Main source file, main.cpp

The main.cpp source file contains the default main function where to start building your project. First the board.h header file has been included. This file includes your application specific function prototypes, which are defined in board.cpp. For your convenience a small board::init() function is provided by default. This function is called within the main function at line 11. It's the first function call here. The second call is to set the LED pin high.

```
#include "board.h"

using namespace aery;


int main(void)

{

  /*

   * Put your application initialization sequence here. The default

   * board initializer defines the LED pin as output and sets the CPU

   * clock speed to 66 MHz.

   */

  board::init();

  gpio_set_pin_high(LED);


  for(;;) {

    /* Put your application code here */


  }


  return 0;

}
```

Board specific functions, board.h and .cpp

Use these two files for your board specific functions and macro definitions. A macro definition is for example the following pin declaration

#define LED AVR32_PIN_PC04

Now you don't have to always recall which pin the LED was connected when you want to switch it on. So instead of using this

aery::gpio_set_pin_high(AVR32_PIN_PC04);

you can use this

aery::gpio_set_pin_high(LED);

You can find this default LED macro definition from board.h. There are also other default board related definitions, which you may need to change according to your project. Those are for example

#define ADC_VREF 3.0

#define ADC_BITS 10

These two macro definitions are related to the analog to digital converter (ADC). To change the reference voltage of the ADC, modify the ADC_VREF. Similarly if you decide to use, for example, only eight bits accuracy alter ADC_BITS accordingly. You may like to reduce the accuracy in favor throughput rate of the analog to digital converter. With smaller accuracy ADCs generally work faster.

From these ADC related settings, we get to one of the functions declared in the default version of board.h. That's board::cnv2volt().

```
static inline double cnv2volt(uint32_t cnv)
{
  return cnv * ((double) ADC_VREF / (1UL << ADC_BITS));
}
```

This function han't been declared in a library because it's highly dependant of the reference voltage and accuracy of ADC. Note how it uses ADC_VREF and ADC_BITS internally to calculate the correct voltage for the conversion.

It's intended that you define all your board related functions in board.h and then implement those in board.cpp. Example programs coming with the framework are built in one file with the main function in purpose, but when used in real application those should be refactored into board.h and .cpp.

For example, consider that you had a device which to communicate via SPI. To take an advance of the board abstraction you could write a function like this

```
uint8_t board::write_to_device(uint8_t byte)

{

   return aery::spi_transmit(spi0, 2, byte);

}
```

See how the above function abstracts which SPI and slave select you are using.