Les variables publiques sur les classes sont conseillées par soucis de simplification lorsqu’elles sont des variables de configuration dont les modifications ne déclenchent pas d’action.

Risques :

-Dépassement de stack : ??

Quand un composant a besoin d’un composant du layer d’en dessous, il utilise le « Layer.h » associé. Les composants d’un level ne doivent pas inclure le « Layer.h » de leur level.

Setup gather all global vars so that thread safety problems are easier to catch (as they are always related to a global var)

Always pass via ArdOs to create OS objects so that statistic are possible

Dynamic memory allocation (new/malloc/free/delete) are forbidden at runtime to prevent :

* thread safety in memory allocation (usually not thread safe in light os)
* memory fragmentation

New are allowed at setup but discouraged (prefer a static assembly). Delete and free are not required as the CPU is always reset after use.

# Debug

When the “L” amber LED (on pin 13) is blinking, it mean that something provoked an assert. Assert are managed here:

ard2017\robot\Common\0\_BSP\FreeRTOS\FreeRTOS\_ARM.c

## History tracing

When the log is too heavy and the debugger is not suitable, you need the “debug\_history” feature (typically: debug interrupts). See comments in : robot/Common/0\_BSP/FreeRTOS/debug\_history.h

# Tooling

## Atmel

In order to add modules when a project is created with ASF, you need to go to “ASF->ASF Wizard”.

In case it’s needed, the CPU reference is ATSAM3X8E.

Here is a simple main file to blink the amber led “L” (orange) :

#include <asf.h>

int main (void)

{

sysclk\_init();

board\_init();

gpio\_configure\_pin(LED0\_GPIO, LED0\_FLAGS);

gpio\_set\_pin\_low(LED0\_GPIO);

while(1)

{

gpio\_toggle\_pin(LED0\_GPIO);

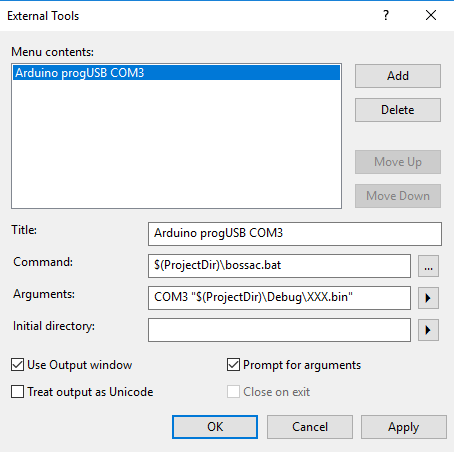
delay\_ms(500);

}

}

## Atmel Prog without JTAG

In order to program the Arduino due from Atmel witout JTAG , we use “bossac”. In “Tools->External tools…” :



Add the following “bossac.bat” file in your project :

@mode %1:1200,n,8,1

@timeout 1

@$(USERPROFILE)\AppData\Local\Arduino15\packages\arduino\tools\bossac\1.6.1-arduino\bossac.exe -i -U false --port=%1 -e -w -b %2 -R

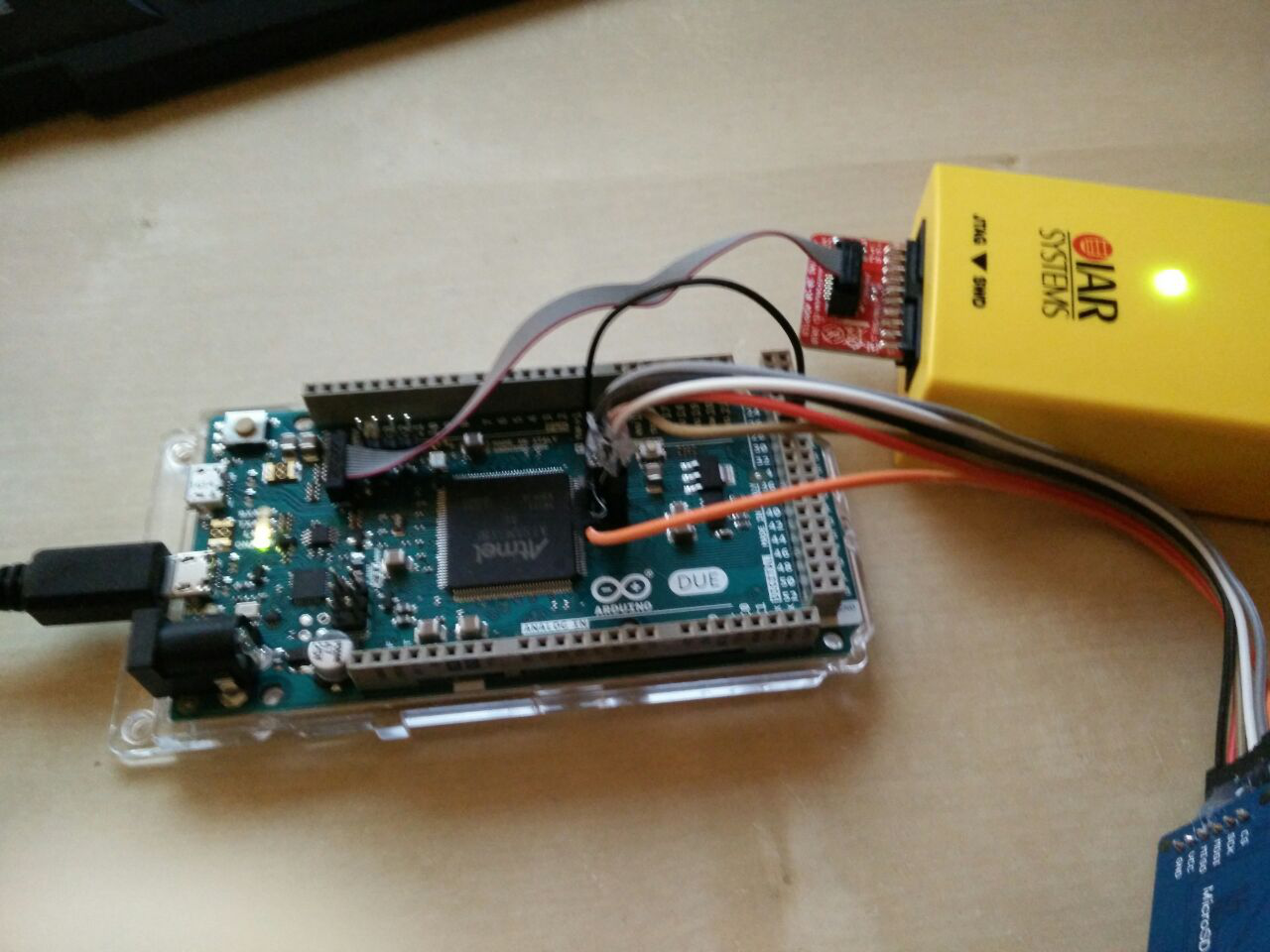
Note : normally we should be able to call bossac directly from the dev env, but for a strange reason, the serial reconfiguration is used to provoke an hard-reset on the board which is necessary.

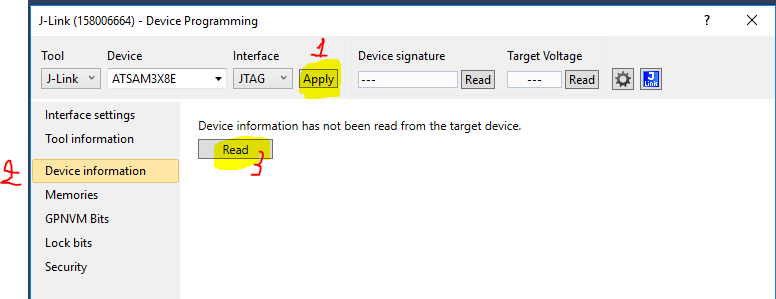
Note2: If you can configure Atmel for this, you can also configure an Eclipse. The difference is that Atmel provide facilities to compile as the dev env is integrated.

## Atmel with JTAG:

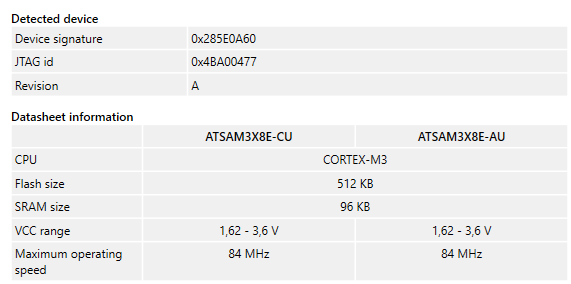
### Jtag setup:

Connect the board with the following wiring (plug the due-jtag first, plug the jtag usb wire to your PC, then power the board):



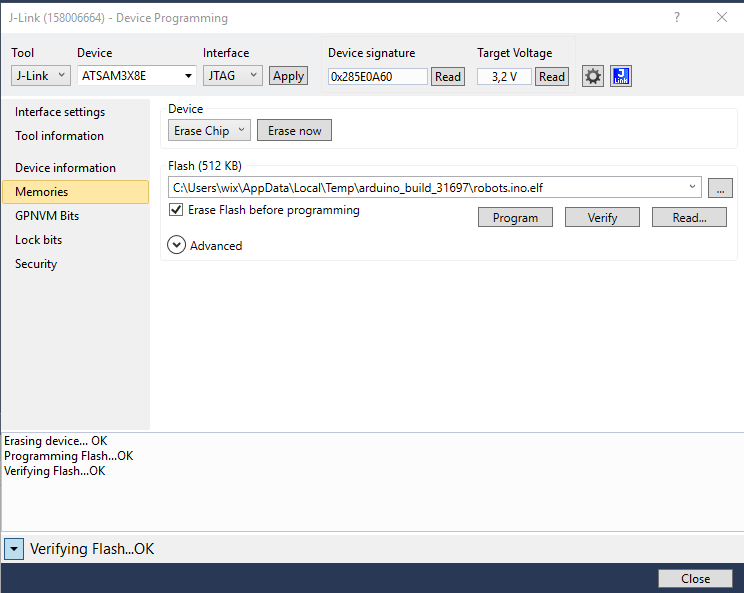
It’s possible to test the device connectivity with “Tools->Device programming”, configure the Jtag connection (Tool/Device/Interface) and clic on apply, then go to the “Device information” tab, and clic read.

You should see this :



### Programming an existing binary

In order to program an existing binary, got to the “Memories” tab, select your file and clic on program:



## Sbrk missing

By default no syscall is possible, so if you include a standard library it’ll miss the \_sbrk function that is responsible for allocating the heap so that malloc and co can work. In order to instanciate a fake sbrt add the following option in compiler miscellaneous text field :

--specs=nosys.specs

# My application bugs at boot

You may have used some reference/pointer in an object contructor that has not been initialized yet. Atmel compiler is very sensitive to this. You have to make it in 2 passes : first create your object with a null pointer to your dependency, build your dependency, then attach your dependency to your object.

# Trucs louches

En compilant sans les options –Os il y a une hard fault au boot…

Quand on build un exe qui dépend d’une lib, l’exe n’est pas toujours mis à jour quand on rebuild l’exe. => solution : ajouter une commande de suppression des binaires dans le dossier Debug/Release en pre-builds

# Add a network message

#### Proto

1. describe your message in a proto file in ard/com/msg folder.
2. generate serialization code : goto ard/com folder and execute generateCom.bat
3. the generated code is in the ard/com/generated folder

#### Vizu

1. add an API to send/receive message in the ard/vizu/com/RemoteControl.py file
2. add a widget to generate a trigger to send the message or a widget to display the message data. Ex : in ard/vizu/gui/TabRobot.py
3. connect your trigger/display to the message api in ard/vizu/vizu.py

#### Robot

1. add your message in robot/Common/2\_AL/RemoteControl.[cpp|hpp]