How to setup and run different test examples on the Atmel SAME70 X PLAINED board.

Tools installed

- Eclipse (installed in C:\Users\hso\eclipse)
- Atmel Studio 7 (installed in C:\Program Files (x86))
- HTerm 0.8.1 (installed in C:\hterm)

Board

- Atmel SAME70 X PLAINED
- Atmel PROTO1 X PLAINED PRO with three light diodes

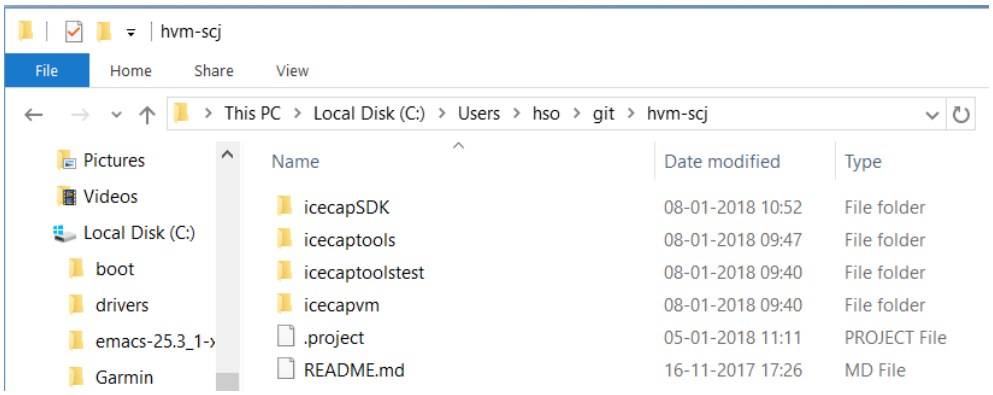
Software

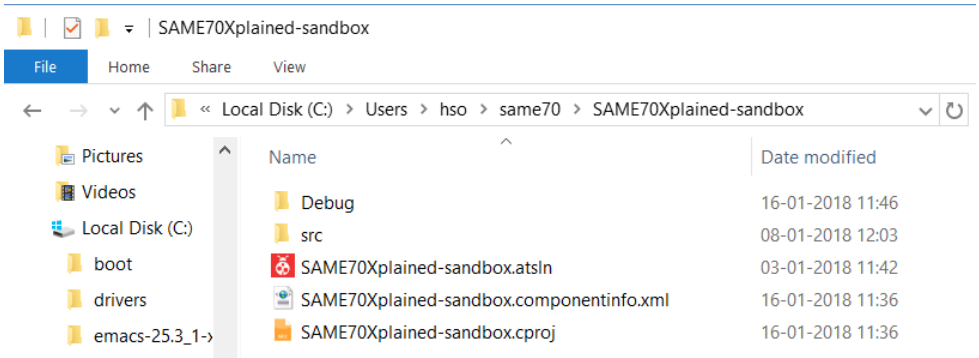
The user software is installed in sub-folders in the "*user home" directory*. In Java, this folder is accessed by the Java statement `System.getProperty("user.home")`. In Windows, the "user home" directory may look like the following `C:\Users\hso`. This "user home" folder is used in the following descriptions.

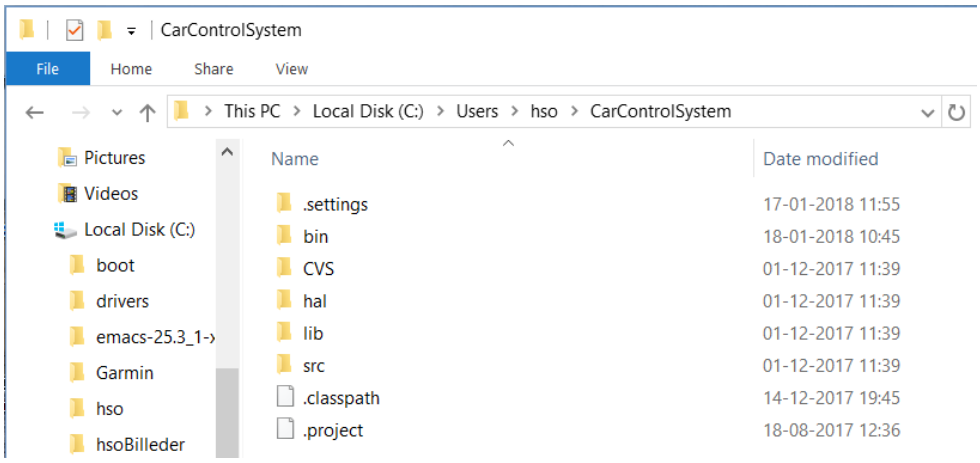
The user software is:

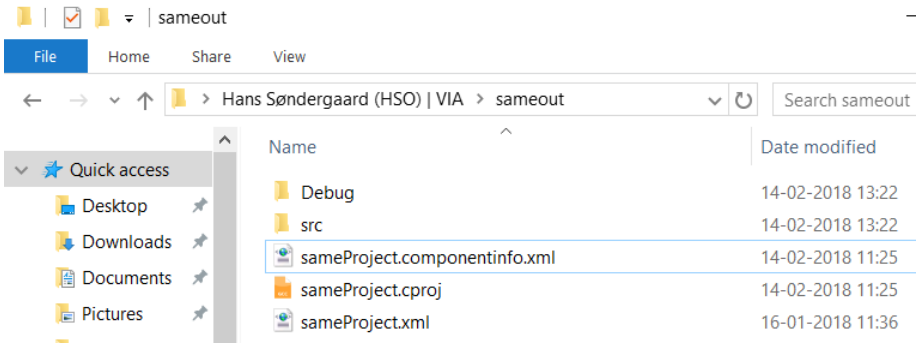
- HVM/SCJ: `C:\Users\hso\git\hvm-scj`
- SAME70Xplained-sandbox: `C:\Users\hso\same70\SAME70Xplained-sandbox`
- Class `HelloAtSAME`: `C:\Users\hso\CarControlSystem`, - or another project
- sameout: `C:\Users\hso\sameout`

This is shown in the following figures.









The example class, `HelloAtSAME`, is in Eclipse placed in `project CarControlSystem` in `package test.same70.examples`. In Section 2, it is demonstrated how to setup and execute this program on the SAME70 board.

1. Getting started

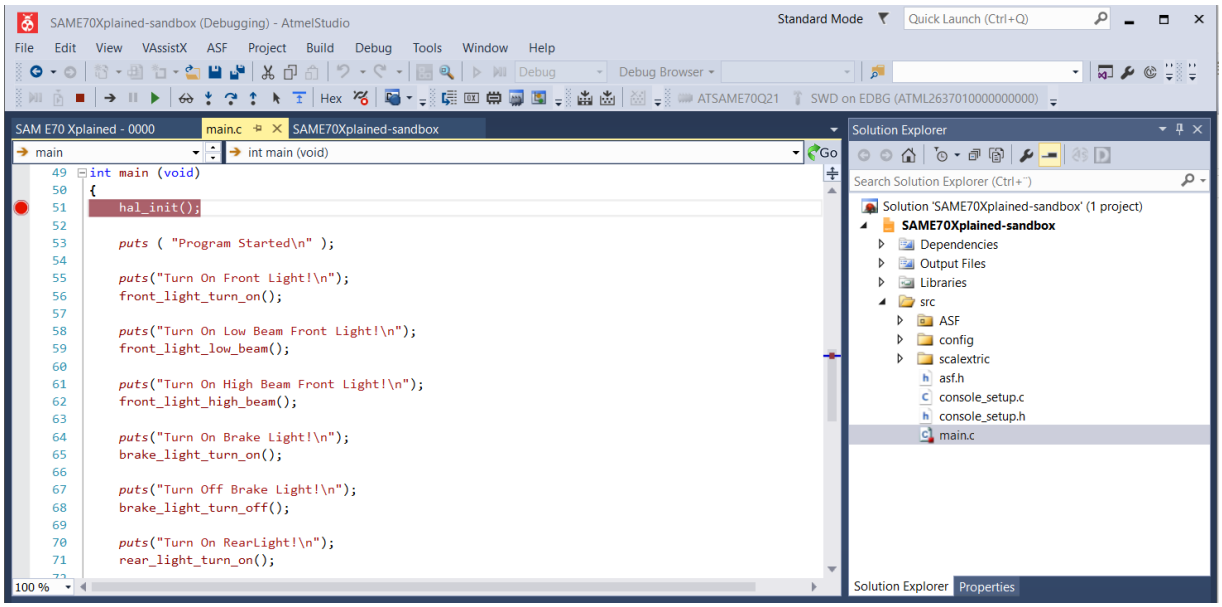
This first example explains how to setup, build and debug the SAME70Xplained-sandbox example.

a) Open the `SAME70Xplained-sandbox.cproj` in AtmelStudio, e.g. by double clicking on the file with this name.
b) Build `SAME70Xplained-sandbox`: Build => Build Solution
c) Board: USB connected from PC to *USB Debug* on Same70 board
d) HTerm started.
   - Connect to COM3 port
   - Baud: 115200
   - Newline at: LF
   - Mark Ascii
   - Send on enter: LF
   - DTR set ("true")

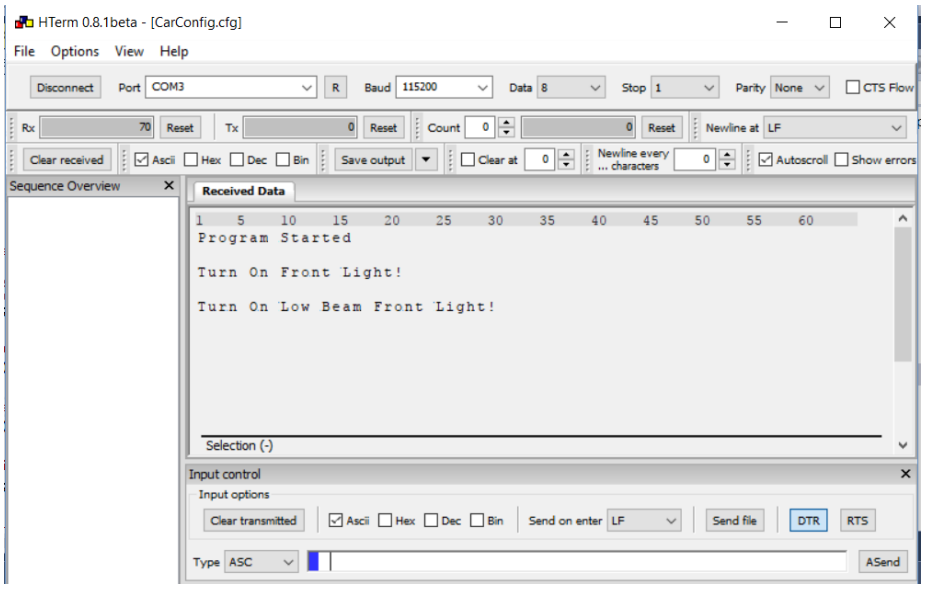
   When the HTerm configuration later on works correct, you can save the configuration for later use:

   File => Save config as ..., e.g. as CarConfig.cfg

   Next time HTerm is started, load the configuration: File => Load Config ...
e) Start Debugging in AtmelStudio:
   Put a breakpoint at the beginning of `main` and start debugging (Green arrows).



If HTerm is configured correctly, the strings from the `puts` statements are received and printed, and on the board, the light diodes are turned on and off.



2. HelloAtSAME

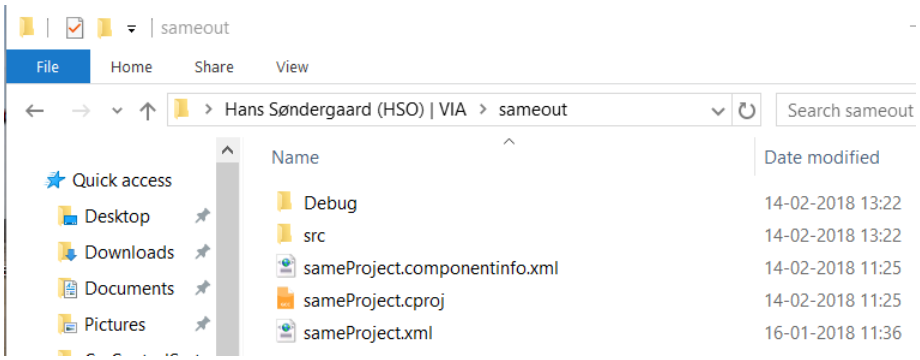
Next, we show how the `HelloAtSAME` example is setup, compiled, and executed. The `main` method looks as follows:

```
public static void main(String[] args) {
  init();  // initialize the board

  devices.Console.println("HelloAtSAME started"); // output to HTerm

  turnOnFrontLight();  // light diode lights up
  delay (10000);       // a short delay
  turnOffFrontLight(); // light diode goes out
```

```
        devices.Console.println("HelloAtSAME end");   // output to HTerm
    }
```

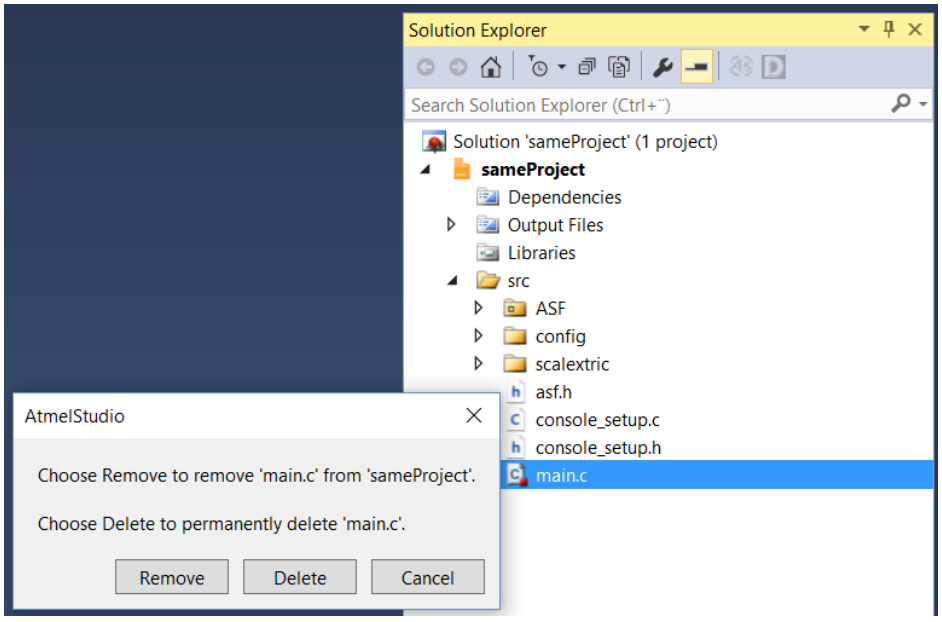
2.1
The `sameout` folder is created in user home: `C:\Users\hso\sameout`

2.2
Copy the content of the `SAME70Xplained-sandbox` folder to the `sameout` folder and rename the `.cproj` and the two `.xml` files to `sameProject` as shown below.



A double-click on `sameProject.cproj` starts the project in AtmelStudio.

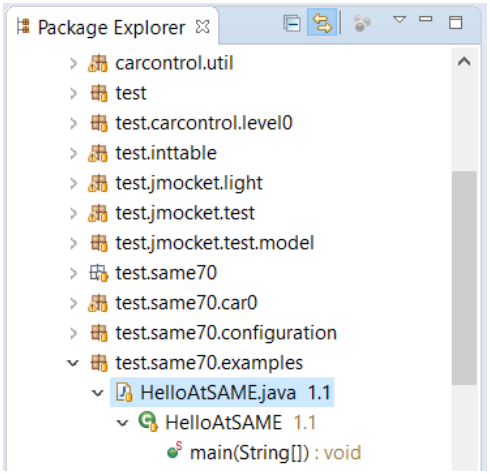
In `src`, `main.c` is removed:
right-click on `main.c` => Remove (but don't delete it (for later use, maybe)).



2.3
Go to Eclipse.

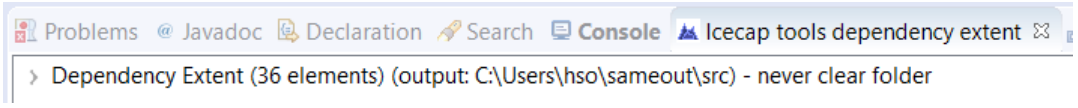
2.3.1
Convert to Icecap Application.

Now right-click on the `HelloAtSAME`'s `main` method in Package Explorer, see below.



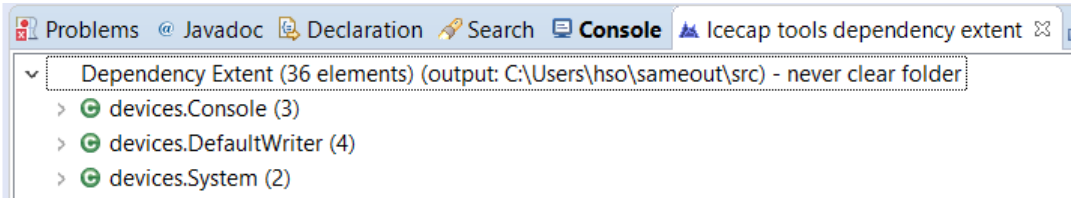
Here, find and start *Icecaptools > Convert to Icecap Application.*

The result is shown in Icecap tools dependency extent:

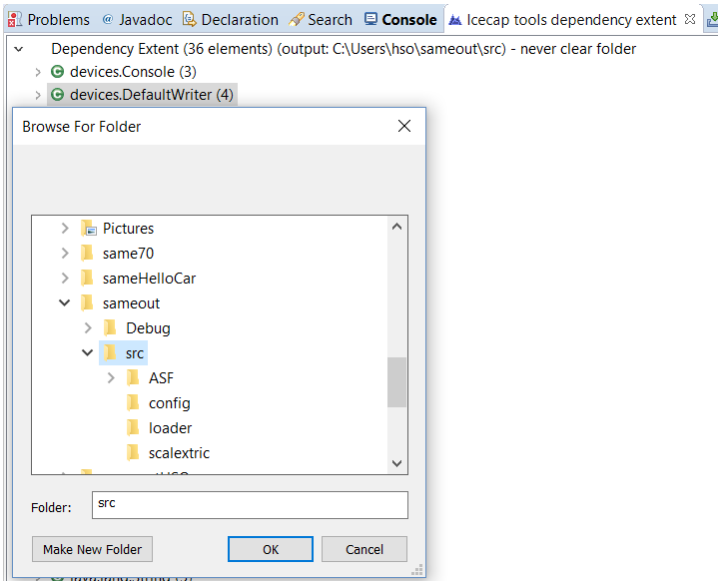


Notice if the output folder is correct and number of elements > 0.
The correct outputfolder is `src` in `sameout`: `C:\Users\hso\sameout\src`.

The dependency extent can be unfolded, as shown below.



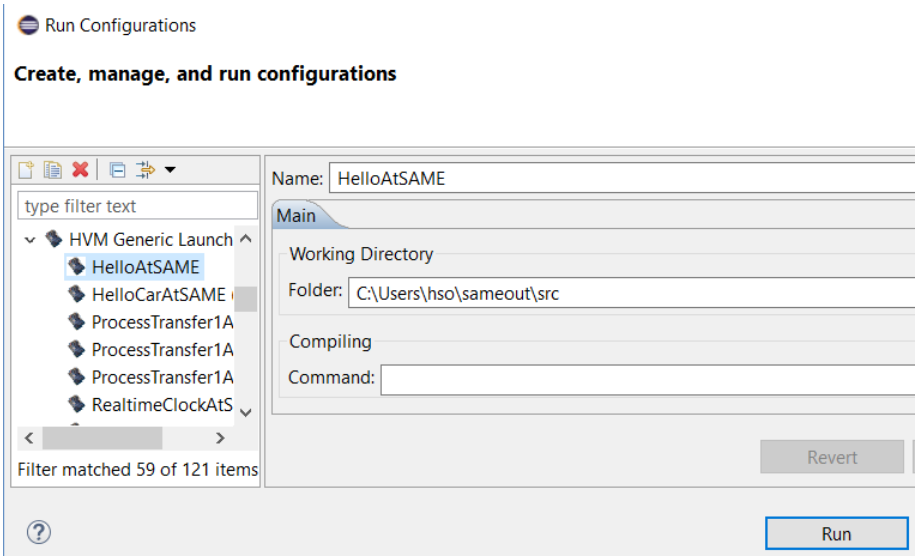
If the *output folder is not correct, or if zero elements, or if an error occurs*, right-click in the window, click on *"Set output folder"*, and browse for the correct output folder.



2.3.3

Run `HelloAtSAME.java` as as a *HVM Generic Launcher*.

In Package Explorer, right-click on `HelloAtSAME.java`:
Run As => Run Configurations ...
and setup a *HVM Generic Launcher* with working directory `C:\Users\hso\sameout\src` as shown below.

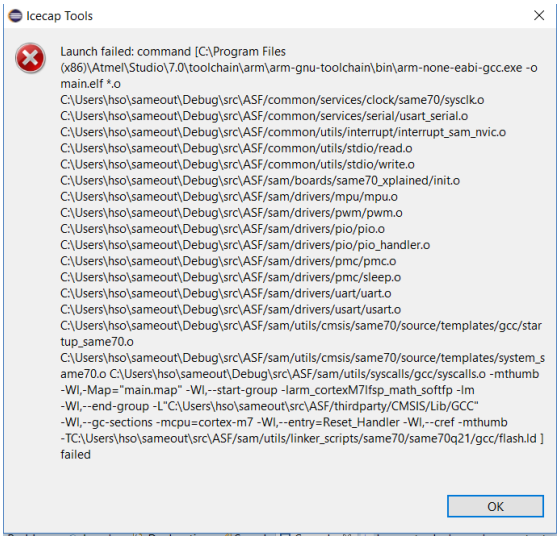


A successful compilation prints something like the following in Console:

```
...
C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\arm\arm-gnu-tool-
chain\bin\arm-none-eabi-gcc.exe ...
Compilation succeeded
C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\...
...
```

Notice that the compilation must be succeeded.
If the compilation is not successful, maybe something is missing (see section 2.4.1).

The rest of the output does not matter. It is some compilation for the board, but in our case, this is finished in AtmelStudio. Maybe an output dialog box pops up as the one shown below.

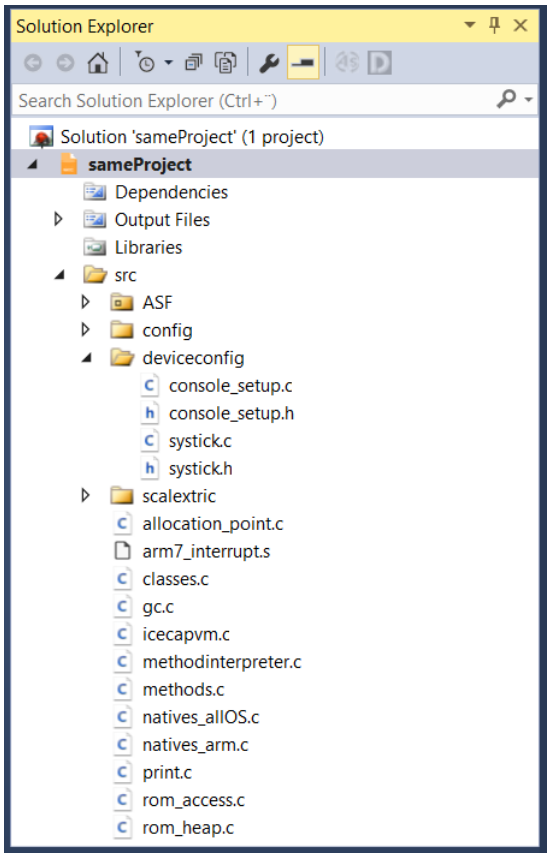


2.4
Switch to AtmelStudio.

2.4.1
The first time, you add the following eleven methods to `src` in `sameProject`:

```
allocation_point.c  classes.c  gc.c  icecapvm.c  methodinterpreter.c
methods.c  natives_allOS.c  natives_arm.c  print.c  rom_access.c
rom_heap.c
```

In Solution Explorer, right-click on `src`, then *Add => Existing Item* ..., find the generated files in `sameProject\src` and add them.
The result is shown in the figure of the Solution Explorer below.



Also notice the `deviceconfig` folder. It contains some user-defined files of which the `console_setup` files are required to be able to run the program.

How to *add a folder*, see

Tip: Add multiple files and folders to an AVR Studio project quickly,
https://avrstudio5.wordpress.com/2011/07/12/tip-add-existing-multiple-files-and-folders-to-an-avr-studio-project-quickly/

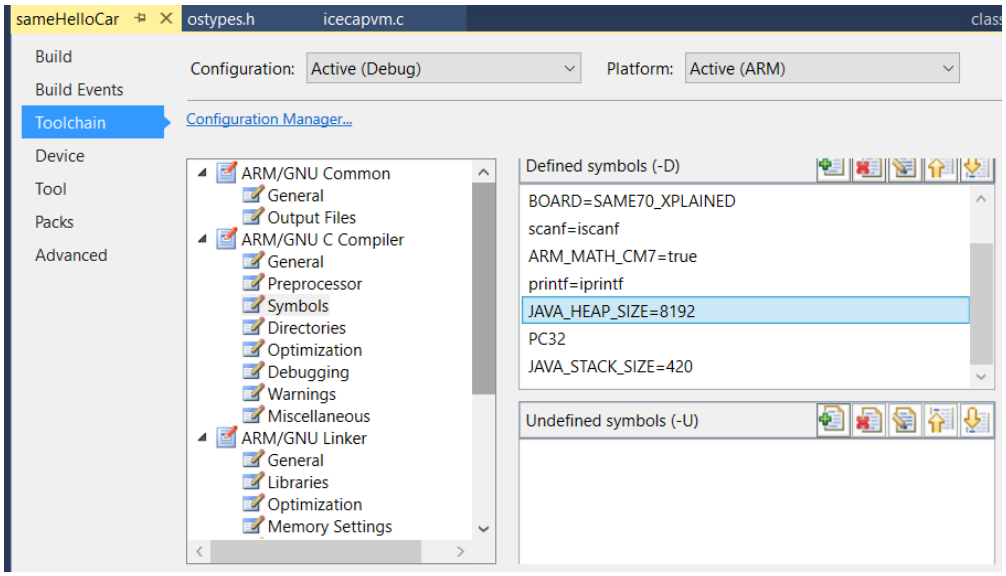
2.4.2

Moreover, the first time you must also add the following Java Define properties (-D properties)

```
JAVA_HEAP_SIZE=8192
PC32
JAVA_STACK_SIZE=420
```

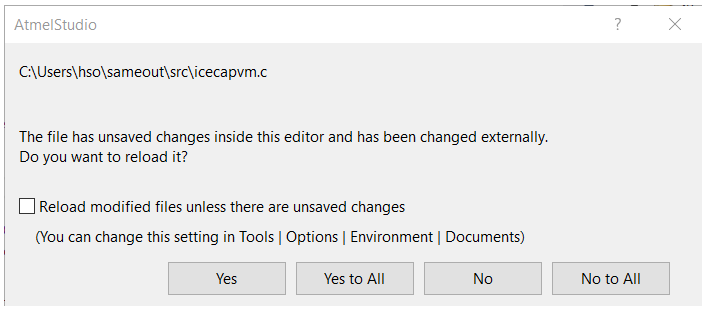
Those Java properties are defined in `package test.same70.configuration`:
`MinimalTargetConfigurationSAME` (or perhaps in `ConfigSAME`).

In Solution Explorer (in AtmelStudio), right-click on the *sameProject => Properties*.
The above properties are added in *Toolchain => ARM/GNU C Compiler => Symbols*, as shown below:



2.4.3

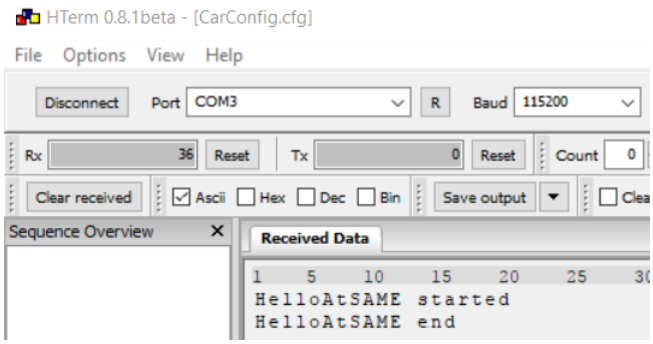
During the development and changing from Eclipse to AtmelStudio, sometimes a dialog box pops up telling that some files have changed. Press "*Yes to All*" to reload the new changes.



2.4.4

In AtmelStudio, continue.

Build and debug, cf. Section 1, b) – e).
The output on HTerm should be as expected, as shown in `main` at the beginning of this Section 2 and illustrated in the figure below; and on the board, the *green light diode lights up and goes out again a little later*. ☺



3. Other examples

In the above example, `HelloAtSAME`, the class extends `MinimalTargetConfigurationSAME`.

3.1

The following examples are subclasses of `TargetConfigurationSAME`. This class implements some *native system tick* methods:

```
void initSystemTick();
void handleSystemTick();
int msDelay(int ms);
int getTicks();
```

The methods call the corresponding C-functions which are specified in `systick.h` and implemented in `systick.c`. These files are in `sameProject` located in the `deviceconfig` folder as shown in the figure in section 2.4.1.

The native methods are used in `TargetConfigurationSAME` to implement

```
public static class ATSAMe70RealtimeClock;
protected static void delay(int i);
```

and in class `MachineFactorySAME` to implement

```
startMachineSpecificSystemTick();
```

In class `MachineFactorySAME`, the above realtime clock, `ATSAMe70RealtimeClock`, is used to implement `getRealtimeClock()`:

```
public RealtimeClock getRealtimeClock() {
  RealtimeClock clock =
    new TargetConfigurationSAME.ATSAMe70RealtimeClock();
  ...
}
```

The system tick methods and the realtime clock are tested in the classes

```
SystemTickAtSAME
RealtimeClockAtSAME.
```

Notice, that the heap size in `TargetConfigurationSAME` is `8192`. Remember to control and, if necessary, edit this Java property in AtmelStudio as illustrated in section 2.4.2.

3.2

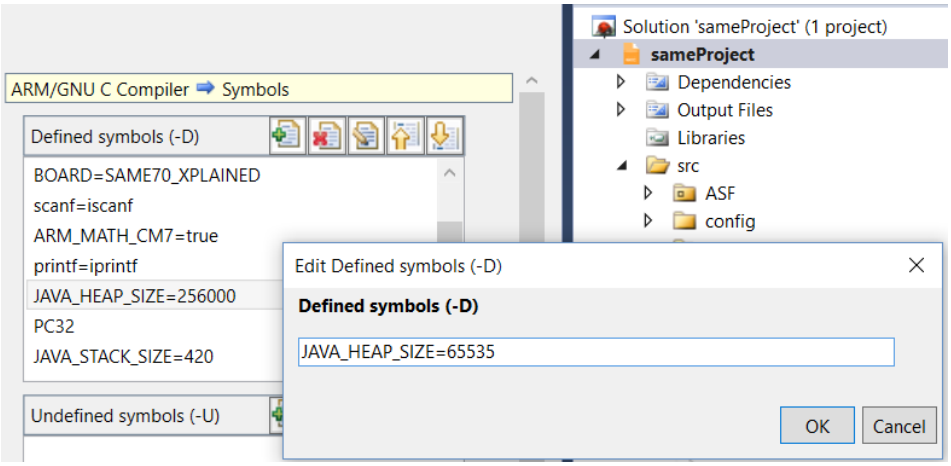
To test different SCJ specific properties, the following SCJ configuration class is used:

```
class SCJTargetConfigurationSAME extends TargetConfigurationSAME;
```

In the same way as above, remember to set the correct heap size in

```
sameProject -> Toolchain ->ARM/GNU C compiler -> Symbols
```

In `SCJTargetConfigurationSAME`, `getJavaHeapSize()` returns `65535`. Therefore, edit “Defined symbols” in `sameProject` and set the correct heap size as illustrated below.



The SCJ test classes are

```
ProcessTransfer1AtSAME
ProcessTransfer2AtSAME
ProcessScheduler1AtSAME
ExecuteWithStackAtSAME
```

They extend the `SCJTargetConfigurationSAME` class.

To be able to make process transfer in the `ProcessScheduler1AtSAME` test example, the `systemTick` variable

```
volatile uint8 systemTick;
```

defined in `icecapvm -> src -> natives_allOS.c`, must be declared and updated in `systick.c` in `sameProject`:

```
extern volatile uint8 systemTick;

void SysTick_Handler(void) {
     ...
     systemTick++;
}
```

3.3

`SCJLevel0AtSAME` is an *SCJ Level 0* test class, which tests a *safelet* implemented in class `SCJLevel0`. As explained in appendix A, at Level 0 the cyclic executive scheduling model is used.

In this example, the *Java property heap size is 300000 and stack size is 1024*. Remember to set those sizes in `sameProject -> Toolchain ->ARM/GNU C compiler -> Symbols`.

The different SCJ memory and stack sizes are defined in `SCJLevel0AtSAME.main`.
The sum of those sizes must be smaller than the defined Java property heap size (300000).

A memory and stack analysis is included in this test example so that it is possible to see how much memory is actually used. An exercise might be to try to optimize (that is to say minimize) the memory usage.

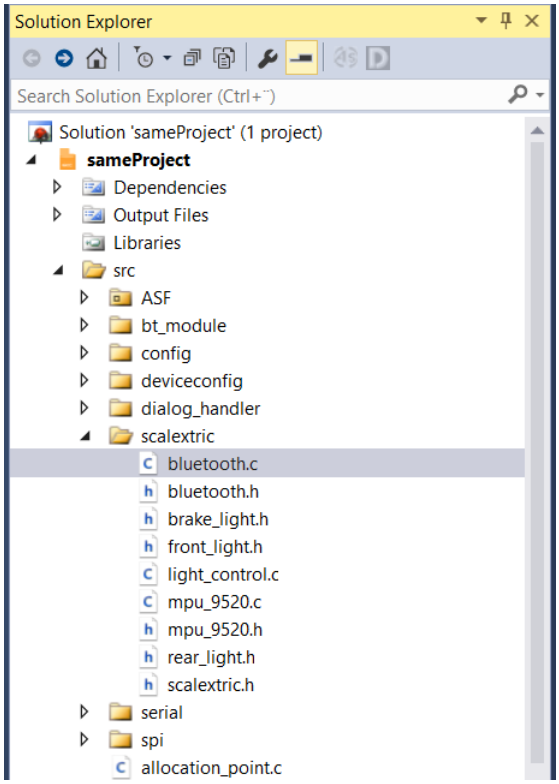
3.4

`CarLevel0_BT_AtSAME` is an *SCJ Level 0* test class with *Bluetooth* connection.
The SCJ setup is the same as above in `SCJLevel0AtSAME` (Section 3.3).

In `class CarConfiguration` the port is instantiated as

```
port = new Port(new BluetoothCommunicationDeviceImpl());
```

In `sameProject` are added the folders: `bt_module`, `dialog_handler`, and `serial`, and the files `bluetooth.h` and `bluetooth.c`



3.5

`SCJLevel1AtSAME` is an *SCJ Level 1* test class, which tests the *safelet* implemented in class `SCJLevel1`. At Level 1, a fixed-priority pre-emptive scheduling model is used.

In this example, the *Java property heap size is 300000*. Remember to set this heap size in `sameProject`.

The different SCJ memory and stack sizes are defined in `SCJLevel1AtSAME.main`, just as in the SCJ Level 0 example in section 3.3.

3.6

`AccelerometerAtSAME extends TargetConfigurationSAME`.

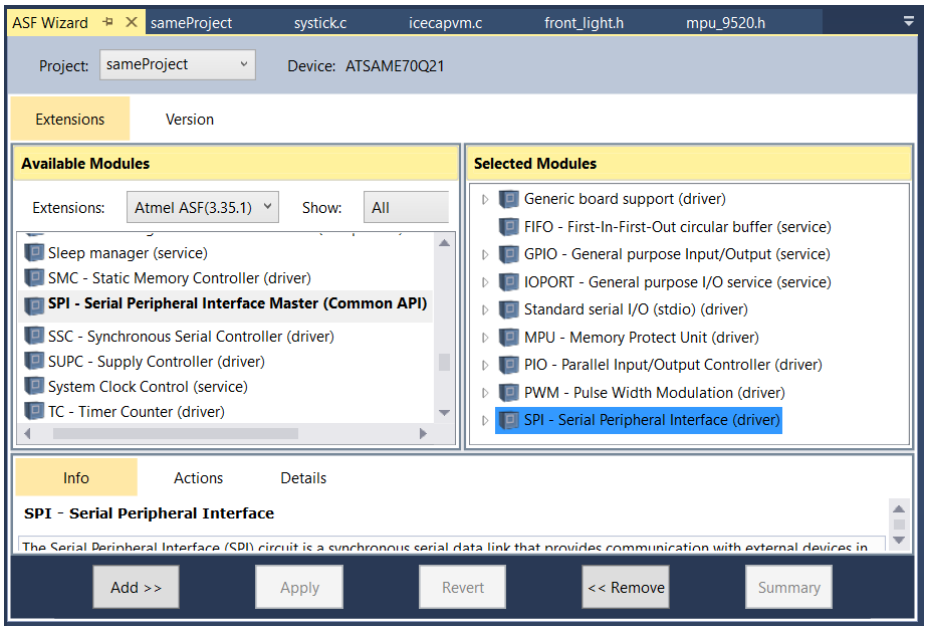
Heap size in `TargetConfigurationSAME` is 8192.

In AtmelStudio:

ASF -> ASF Wizard:

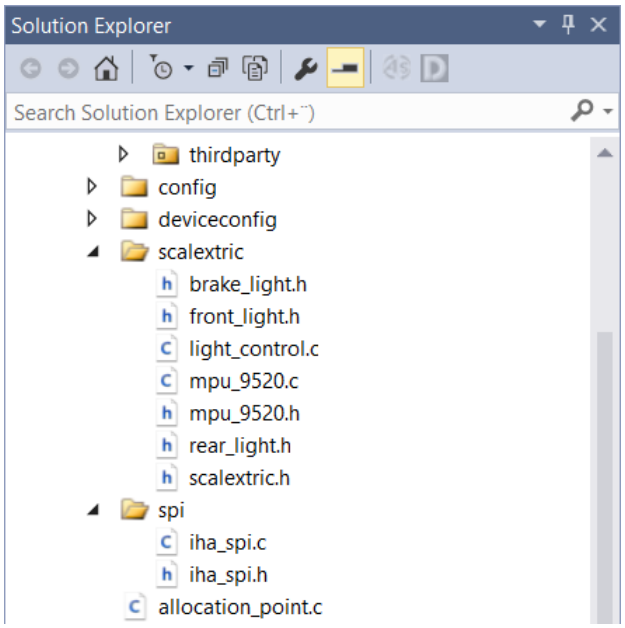
SPI – Serial Peripheral Interface (driver) -> Add>>
FIFI – First-In_First_out circular buffer (service) -> Add>>

Finally: Apply-> OK (overwrite: YES).



New C files:

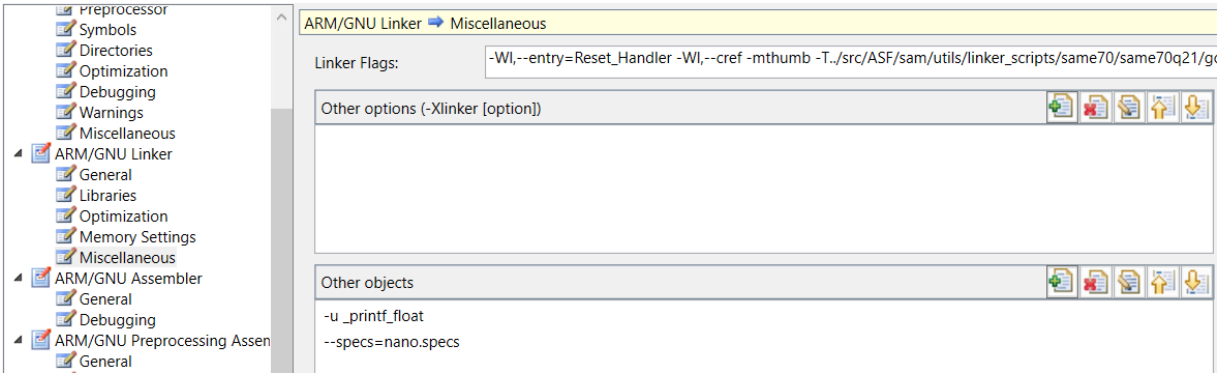
`mpu_9520.h` and `mpu_9520.c`
`iha_spi.h` and `iha_spi.c`



In ARM/GNU Linker -> Miscellaneous, add the following link options:

```
--specs=nano.specs
-u _printf_float
```

This is shown in the figure below. For further explanation, see [12].


Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
ARM/GNU Linker
General
Libraries
Optimization
Memory Settings
Miscellaneous
ARM/GNU Assembler
General
Debugging
ARM/GNU Preprocessing Assen
General
ARM/GNU Linker ➡ Miscellaneous
Linker Flags:
-Wl,--entry=Reset_Handler -Wl,--cref -mthumb -T../src/ASF/sam/utils/linker_scripts/same70/same70q21/g
Other options (-Xlinker [option])
Other objects
-u _printf_float
--specs=nano.specs