EM BEACON FIRMWARE

PORTING THE CODE TO GCC4.4 COMPILER

Scope

Your EM Beacon code has been developed under a GCC toolchain based on Ride7 IDE.
The GCC toolchain was based on GCC3.2 compiler. It now comes with the next release GCC4.4.
For different reasons it might be useful to use GCC4.4 instead of GCC3.2.

1. GCC3.2 is not supported by Windows 10 and next coming Windows operation system
2. GCC3.2 maintenance is frozen.
3. All bug fixes and new features will only appear in upgrades of GCC version 4.4 (4.x, 5.x, etc.).
4. GCC4.4 offers some better optimization results (typically from 10 to 15% less code footprint produced)

Next releases of the Beacon firmware will be offered for GCC4.4 and upgraded version of GCC4.4.
During this interim period or if you are working on an older Beacon firmware version, porting your Beacon firmware might be required.
This technical note describes the few modifications you need to implement in your project to comply with GCC4.4 (i.e. it describes the list of required modifications to port your Beacon firmware from GCC3.2 to GCC4.4).
Some aspects of the modifications are not deeply explained in this document. To get a deeper understanding, refer to the "Raisonance tools for C816 family – Getting Started" document [GettingStartedC816_Ride7.pdf file] located under <RideInstallDir>*Raisonance \Ride\Doc\C816 folder.*

This Technical Note comes with a special release of the Beacon firmware 3.0.0 prepared for GCC4.4 toolchain: this release has already been updated according to the description enclosed in this document. It can be used as an example you may refer to if needed.

Enable your GCC4.4 toolchain

To do so, go to the Ride\GNU directory and run the setGCC4.bat batch file (some systems require that you are signed in as Administrator). After this, Ride7 will compile using GCC 4.4.

After switching to GCC4, there are a few things that you will need to do to compile your Beacon project (originally developed under GCC3.2).
You must follow the instructions below carefully.

Modification of the linker script ctr0.ld

The file is located under your Beacon source (src) folder <BeaconDir>*EMBCxx_beacon\src*

1. Add sub-sections in all sections to allow code elimination and avoid linking error. It is required for all sections (text, data, etc …)

```
Example:
 .rodata :              /* the constant values go after the code */
 /*--------------------------------------------------------------------*/
  {
    . = ALIGN(32);  /* align on row beginning */
   _srodata = .;
     *(.rodata)
     *(.rodata* )  /* required with GCC4 dead code removal optis */
     *(.rodata.* ) /* required with GCC4 dead code removal optis */
   _erodata = .;
  } > data_rom
```

Note : This change is backwards compatible with GCC3.2. Therefore you don't need to change anything in case you use this new version of crt0.ld under GCC3.2 toolchain.

2. The other crt0.ld modification concerns page0 allocation and linking section order.
To avoid allocation problems by the linker with page0, you must modify your linker script and move all ".page0*" (.page0_bss, .page0_data, etc.) sections to the first position in the SECTIONS part of the file, before all non-page0 data (.bss, .data, etc.).

Note : This change is backwards compatible with GCC3.2. Therefore you don't need to change anything in case you use this new version of crt0.ld under GCC3.2 toolchain.

Compiler options & settings

1. Do not use GCC standard headers

Make sure this Compiler Dialect option is deactivated when using GCC4 – as shown on Figure 1: Compiler dialect settings for GCC4.4

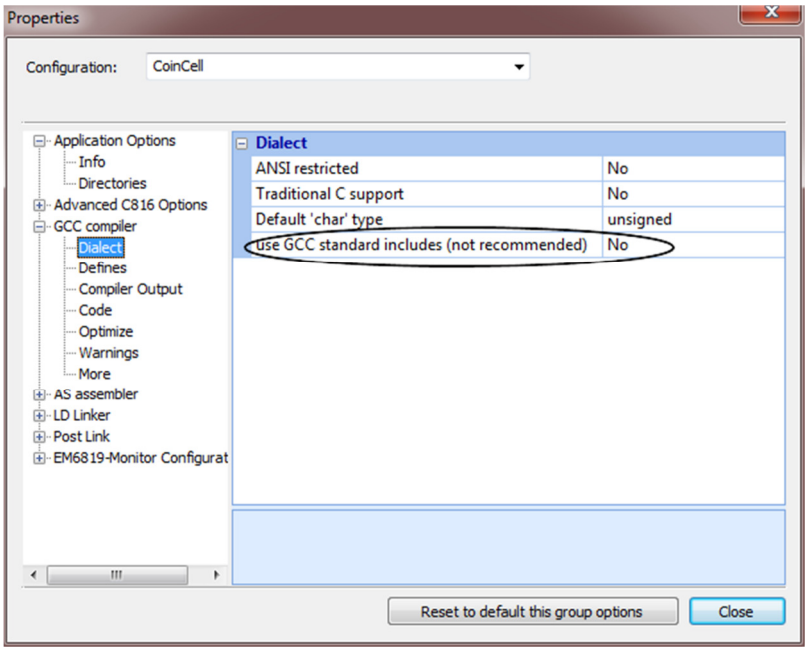

Figure 1: Compiler dialect settings for GCC4.4

Note : This change is not backwards compatible with GCC3.2. Therefore under GCC3.2 toolchain the "use GCC standard includes (not recommended)" Dialect settings must be set to Yes.

2. Use –Os Optimization level
   Make sure this level is set when using GCC4 – as shown on Figure 2: Compiler Optimize settings for GCC4.4
3. Enable the Ignore Inline option
   Make sure this level is enabled when using GCC4 – as shown on Figure 2: Compiler Optimize settings for GCC4.4
4. Enable the per function section option
   Make sure this level is enabled when using GCC4 – as shown on Figure 2: Compiler Optimize settings for GCC4.4

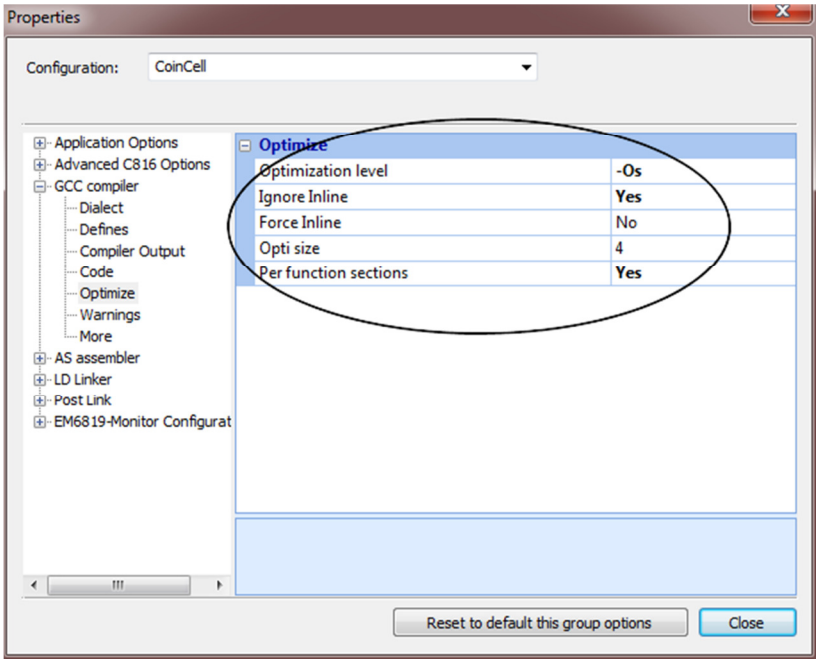

Figure 2: Compiler Optimize settings for GCC4.4

Note : These changes are not backwards compatible with GCC3.2. Therefore under GCC3.2 toolchain the "Compiler Optimize settings" must be set as shown on Figure 3: !!! Compiler Optimize settings for GCC3.2 !!!

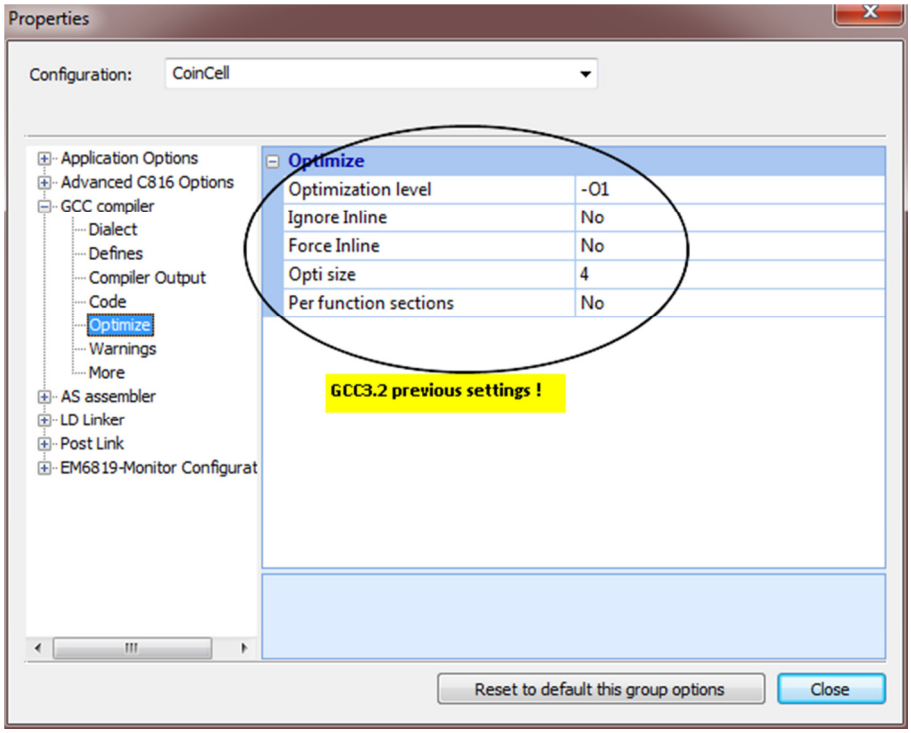

Figure 3: !!! Compiler Optimize settings for GCC3.2 !!!

Linker options & settings

1. Disable Data Init feature
   Make sure this option is disabled when using GCC4 – as shown on Figure 4: Linker General settings for GCC4.4
2. Remove unused sections
   Make sure this option is enabled when using GCC4 – as shown on Figure 4: Linker General settings for GCC4.4
3. Remove Program Headers Alignment – as shown on Figure 1: Compiler dialect settings for GCC4.4
   Make sure this option is deactivated when using GCC4 – as shown on Figure 4: Linker General settings for GCC4.4

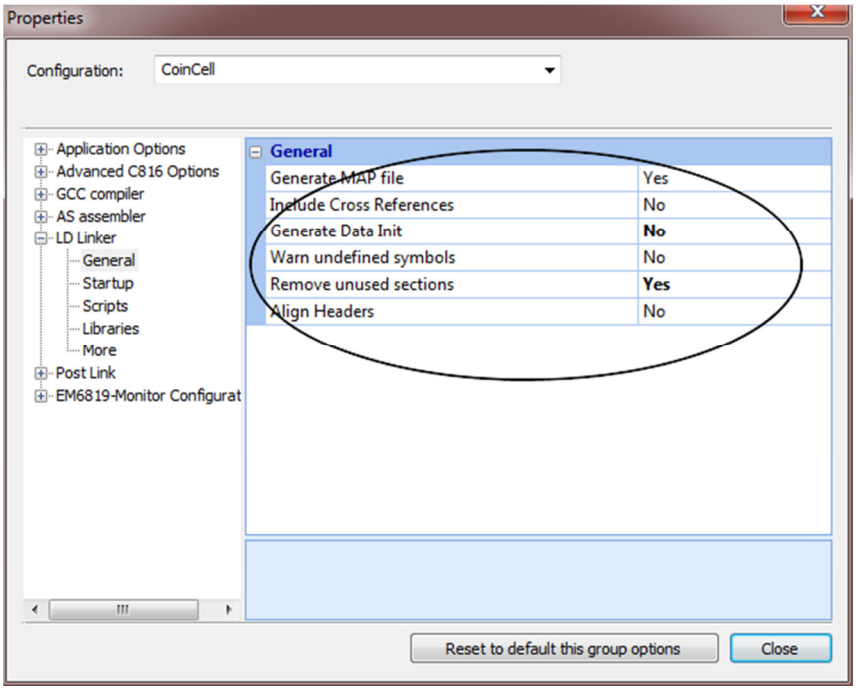

Figure 4: Linker General settings for GCC4.4

Note : This change is not backwards compatible with GCC3.2. Therefore under GCC3.2 toolchain the "General linker settings" must be set as shown on Figure 5: !!! Linker General settings for GCC3.2 !!!

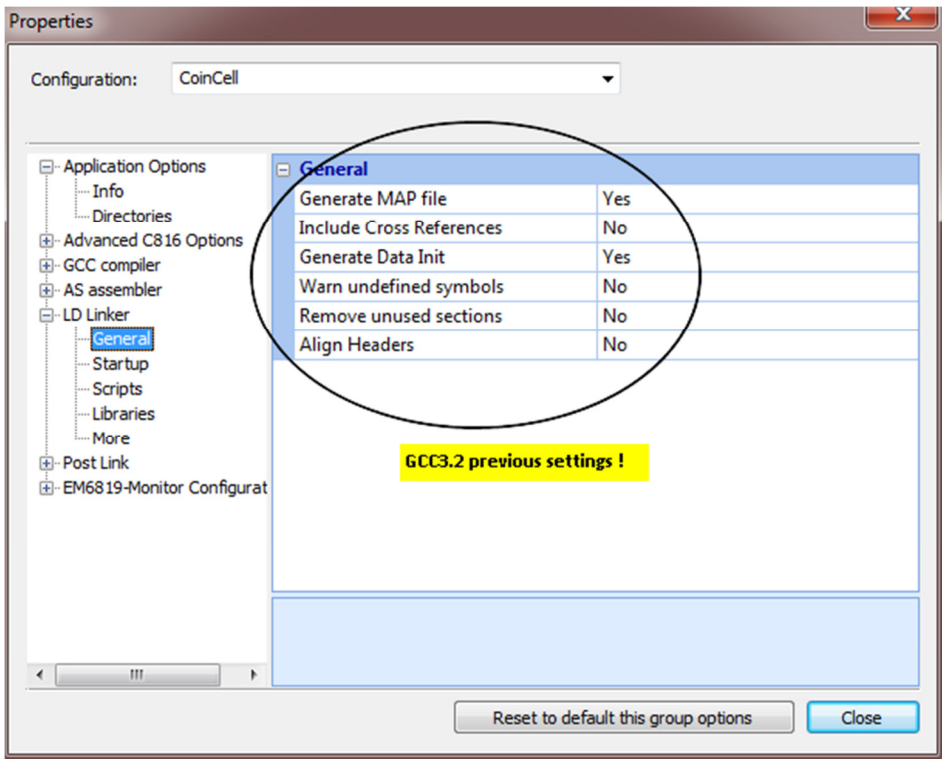

Figure 5: !!! Linker General settings for GCC3.2 !!!

Post link settings

Disable the Post Link option.
Make sure this option is deactivated when using GCC4 – as shown on Figure 6: Post Link setting

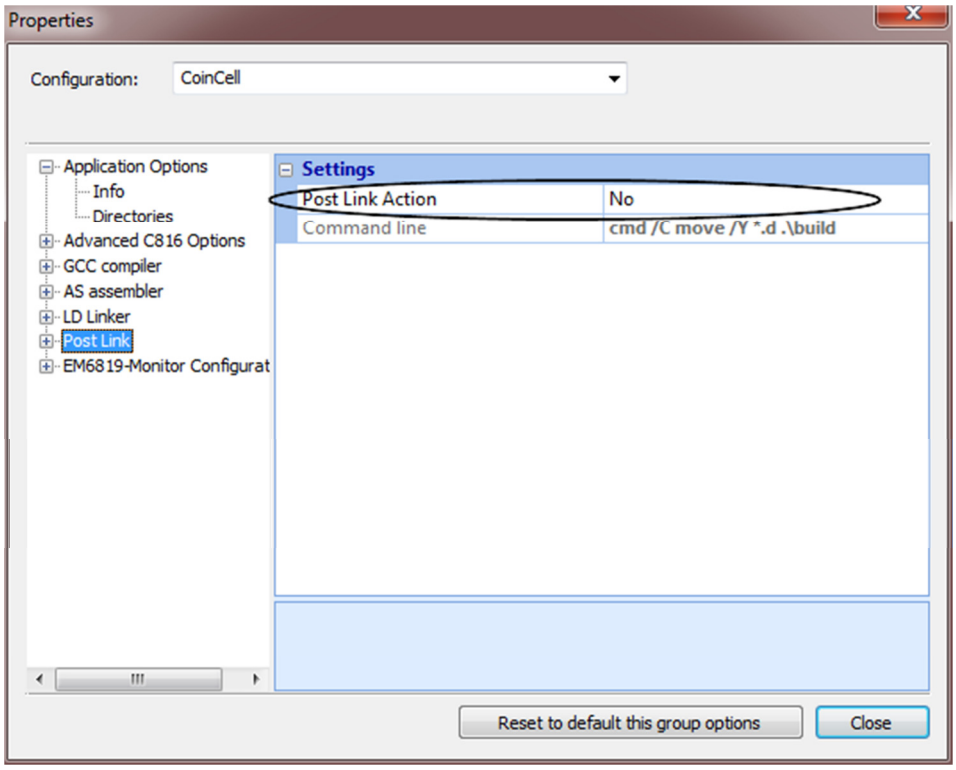

Figure 6: Post Link setting

Note : This change is not backwards compatible with GCC3.2. Therefore under GCC3.2 toolchain the "Post Link option" setting must be defined as shown on Figure 7: !!! Post Link settings for GCC3.2 !!!

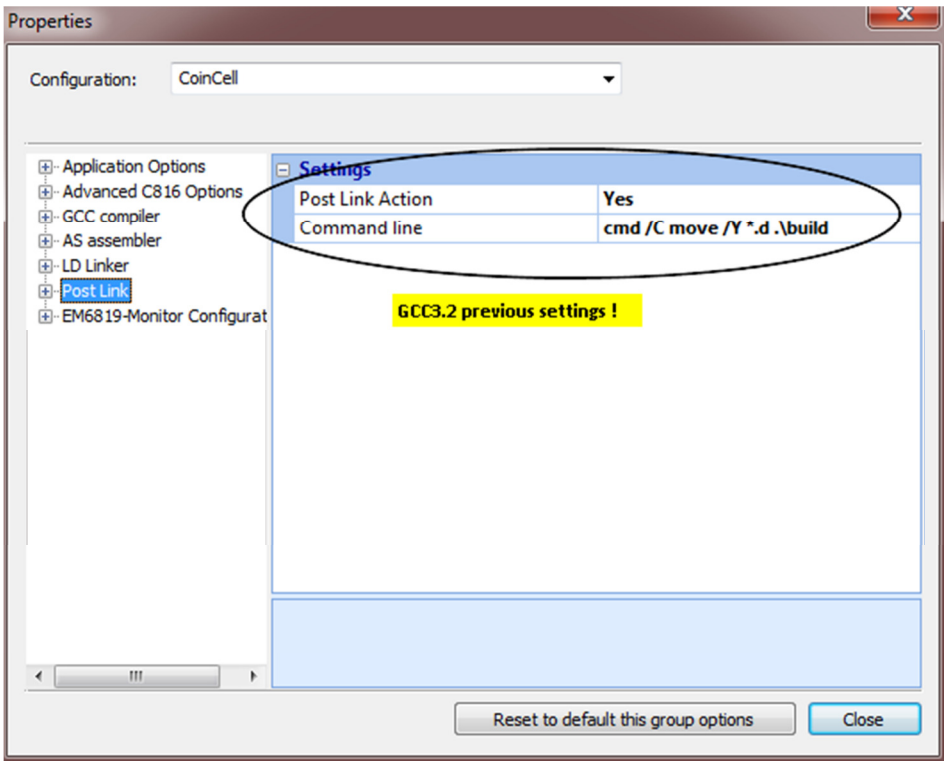

Figure 7: !!! Post Link settings for GCC3.2 !!!

Contact Information

Technical support:
http://www.emdeveloper.com
support@emdeveloper.com