Freescale Semiconductor

Merging applications using Kinetis Design Studio

By: Carlos Mendoza / Technical Information Center



About this document

Usually a user application can be developed independently, that is without the bootloader, and it can be loaded into the microcontroller and debugged directly. However, for production purposes it is worth merging the user application and bootloader together, so it can be downloaded into the microcontroller all at once as a single file and reduce manufacture time and cost.

This document shows two methods of merging the applications.

- 1. Merging applications using linker commands.
- 2. Merging applications using the P&E Advanced Flash Programming options.

The steps described in the document were done using the MK64FN1M0VLL12 MCU like the one in the FRDM-K64F board, but the same principles are applicable to any Kinetis MCU.

Software versions

The steps described in this document are valid for the following versions of the software tools:

KDS v3.0.0

Contents

1. Glossary	3
2. Overview and concepts	4
2.1 Linker File (.ld)	4
2.1.1 Memory Segment	4
2.1.2 Sections Segment	4
3. Merging two applications	5
3.1 Creating applications	5
3.2 Merging applications using linker commands	10
3.3 Merging applications using the P&E Advanced Flash Programming options	14
Appendix A – Binary file generation in KDS	22
Appendix B - References	24

1. Glossary

- **KDS** *Kinetis Design Studio*: Integrated Development Environment (IDE) software for Kinetis MCUs.
- **KSDK** *Kinetis Software Development Kit*: Set of peripheral drivers, stacks and middleware layers for Kinetis microcontrollers.

2. Overview and concepts

2.1 Linker File (.ld)

The Linker file (.ld) combines a number of object and archive files, relocates their data and ties up symbol references. Usually the last step in compiling a program is to run ld.

2.1.1 Memory Segment

The memory segment is used to divide the microcontroller memory into segments. Each segment can have read, write and execute attributes. The address and the length of each segment are defined as well. An example is shown in listing 1.

MEMORY		
{		
<pre>m_interrupts</pre>	(RX)	: ORIGIN = 0x00000000, LENGTH = 0x00000400
<pre>m_flash_config</pre>	(RX)	: ORIGIN = 0x00000400, LENGTH = 0x00000010
m_text	(RX)	: ORIGIN = 0x00000410, LENGTH = 0x000FFBF0
m_data	(RW)	: ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
m_data_2	(RW)	: ORIGIN = 0x20000000, LENGTH = 0x00030000
}		

Listing 1 – K64 Memory segment

2.1.2 Sections Segment

In sections segment are defined the contents of target-memory sections. In other words, a section indicates which parts of your application will be allocated in each memory segment. Main sections are '.text' which contains all the code and the constants of an application, '.data' which contains all initialized data, and '.bss' which contains all non-initialized data.

Below you can see section '.text' of an application using K64. As you can notice it is contained in segment 'm_text'.

```
.text :
{
  . = ALIGN(4);
  *(.text)
                           /* .text sections (code) */
  *(.text*)
                           /* .text* sections (code) */
                          /* .rodata sections (constants, strings, etc.) */
  *(.rodata)
                          /* .rodata* sections (constants, strings, etc.) */
  *(.rodata*)
                          /* glue arm to thumb code */
  *(.glue 7)
                          /* glue thumb to arm code */
  *(.glue_7t)
  *(.eh frame)
 KEEP (*(.init))
 KEEP (*(.fini))
  = ALIGN(4);
} > m_text
```

Listing 2 - K64 .text section

3. Merging two applications

3.1 Creating applications

- Before we start merging the projects we need to create 2 new baremetal or KSDK projects (No Processor Expert) using KDS, select MK64FN1M0xxx1 as device. Let's call them Application1 and Application2.
- Application1 is supposed to be the Bootloader, as example it will just toggle the blue LED 5 times. It will be linked to the default memory address which is 0x0000 and will end at 0xA000 (the flash space reserved for the bootloader will depend on the size of your bootloader).
- Application2 is supposed to be the user Application, as example it will toggle the red LED forever. It will be linked to the address 0xA000 and will end at 0x0010_0000.
- Here is the Flash memory layout of how the bootloader and application will be programmed:

m_interrupts -[0x0000_0000 m_flash_config -[0x0000_0400 0x0000_0410 m_text _	Application1
m_interrupts0x000A_0000 m_flash_config0x000A_0400 m_text 0x000A_0410	Application2

- In a default project the memory segments of MK64FN1M0xxx1 are defined in the linker file as next:

MEMORY		
{		
<pre>m_interrupts</pre>	(RX)	: ORIGIN = 0x00000000, LENGTH = 0x00000400
<pre>m_flash_config</pre>	(RX)	: ORIGIN = 0x00000400, LENGTH = 0x00000010
m_text	(RX)	: ORIGIN = 0x00000410, LENGTH = 0x000FFBF0
m_data	(RW)	: ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
m_data_2	(RW)	: ORIGIN = 0x20000000, LENGTH = 0x00030000
}		

Listing 3 – Default Memory segments

First we need to reduce the memory size in Application1 to leave some space and create a new memory segment for Application2 which will start in address 0xA000. Open MK64FN1M0xxx12_flash.ld located in "\${ProjDirPath}/Project_Settings/Linker_Files", the MEMORY segment must look as below after being edited.

MEMORY {		
m_interrupts	(RX)	: ORIGIN = 0x00000000, LENGTH = 0x00000400
m_flash_config	(RX)	: ORIGIN = 0x00000400, LENGTH = 0x00000010
m_text	(RX)	: ORIGIN = 0x00000410, LENGTH = 0x00009BF0
app2_text	(RX)	: ORIGIN = 0x0000A000, LENGTH = 0x000F6000
m_data	(RW)	: ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
m_data_2	(RW)	: ORIGIN = 0x20000000, LENGTH = 0x00030000
}		

Listing 4 – Application1 modified Memory segments

[m_text]Bootloader space[app2_text]Application space

As it was mentioned before, the Application2 must be linked in address 0xA000, therefore we must add an offset of 0xA000 to all the flash segments. Open MK64FN1M0xxx12_flash.ld located in "\${ProjDirPath}/Project_Settings/Linker_Files", the MEMORY segment must be edited as follows:

MEMORY {			
<pre>m_interrupts</pre>	(RX)	: ORIGIN = 0x0000A000, LENGTH = 0x00000400	
m_flash_config	(RX)	: ORIGIN = 0x0000A400, LENGTH = 0x00000010	
m_text	(RX)	: ORIGIN = 0x0000A410, LENGTH = 0x000F5BF0	
m_data	(RW)	: ORIGIN = 0x1FFF0000, LENGTH = 0x00010000	
m_data_2	(RW)	: ORIGIN = 0x20000000, LENGTH = 0x00030000	
}			

Listing 5 – Application2 modified Memory segments

- Build **Application2**, then look for Application2.map which you will find "\${ProjDirPath}/Debug":



- Open the Application2.map file and search for the Reset_Handler which is the application entry point address. As you can see in this case it is 0x0000a4d8:

Merging applications using Kinetis Design Studio

Listing 6 – Application2.map Entry Point

- Now go back to **Application1**. We will make this application to toggle the onboard blue LED 5 times and then jump to the entry code of **Application2** (Reset_Handler). You can use the following code:

```
/* include peripheral declarations "fsl_device_registers.h" if it is a KSDK project or
"MK64F12.h" if it is a baremetal project */
#define GPIO_PIN_MASK
                                   0x1Fu
#define GPIO PIN(x)
                                   (((1)<<(x & GPIO PIN MASK)))
void delay();
int main(void){
       int i;
       /* Turn on all port clocks */
       SIM_SCGC5 = SIM_SCGC5_PORTA_MASK | SIM_SCGC5_PORTB_MASK |
SIM_SCGC5_PORTC_MASK | SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;
       /*Set PTB21 (connected to BLUE LED) for GPIO functionality*/
       PORTB PCR21=(0|PORT PCR MUX(1));
       /*Change PTB21 to output*/
       GPIOB_PDDR=GPIO_PDDR_PDD(GPI0_PIN(21));
       for(i = 0; i < 10; i++){</pre>
              /*Toggle the blue LED on PTB21*/
              GPIOB_PTOR | =GPIO_PDOR_PDO(GPIO_PIN(21));
              delay();
       }
       __asm("bl 0x0000a4d8"); //Jump to Application2 entry point
       return 0;
}
void delay(){
  unsigned int i, n;
  for(i=0;i<10000;i++){</pre>
         for(n=0;n<200;n++){
                __asm("nop");
         }
  }
}
```

Listing 7 – Application1 main.c

The next step is to make **Application2** toggle the Red LED forever. You can use the following code:

```
/* include peripheral declarations "fsl_device_registers.h" if it is a KSDK project or
"MK64F12.h" if it is a baremetal project */
#define GPI0_PIN_MASK
                                   0x1Fu
#define GPIO_PIN(x)
                                   (((1)<<(x & GPIO_PIN_MASK)))
void delay();
int main(void){
       /* Turn on all port clocks */
       SIM SCGC5 = SIM SCGC5 PORTA MASK | SIM SCGC5 PORTB MASK |
SIM_SCGC5_PORTC_MASK | SIM_SCGC5_PORTD_MASK | SIM_SCGC5_PORTE_MASK;
       /*Set PTB22 (connected to RED LED) for GPIO functionality*/
      PORTB_PCR22=(0|PORT_PCR_MUX(1));
       /*Change PTB22 to output*/
      GPIOB_PDDR=GPIO_PDDR_PDD(GPI0_PIN(22));
      while(1){
              /*Toggle the RED LED on PTB22*/
             GPIOB PTOR = GPIO PDOR PDO(GPIO PIN(22));
              delay();
       }
      return 0;
}
void delay(){
  unsigned int i, n;
  for(i=0;i<10000;i++){</pre>
         for(n=0;n<200;n++){
                __asm("nop");
         }
  }
}
```

Listing 8 – Application2 main.c

3.2 Merging applications using linker commands

- Now that the two applications have been created we will merge them using linker commands, first we need to generate a binary file of the **Application2** to insert it in the **Application1**, to do this follow the steps described in the Appendix A. After this, copy the binary file into the 'Sources' folder in **Application1**.



 The next step is to tell the Application1 linker to include the Application2 binary file. First you need to use TARGET, INPUT and OUTPUT_FORMAT commands. You can do this just after MEMORY segment and before SECTIONS segment:

```
/* Specify the memory areas */
MEMORY
{
  m_interrupts
                          (RX) : ORIGIN = 0x00000000, LENGTH = 0x00000400
  m_flash_config
                          (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
                          (RX) : ORIGIN = 0x00000410, LENGTH = 0x00009BF0
  m_text
                         (RX) : ORIGIN = 0x0000A000, LENGTH = 0x000F6000
  my_text
                         (RW) : ORIGIN = 0x1FFF0000, LENGTH = 0x00010000
  m data
                         (RW) : ORIGIN = 0x20000000, LENGTH = 0x00030000
  m_data_2
}
                                 /* specify the file format of binary file */
TARGET(binary)
INPUT (Application2.bin)
OUTPUT_FORMAT(default)
                                /* provide the file name */
/* restore the out file format */
/* Define output sections */
SECTIONS
{
•••
```

Listing 9 – Application1 modified linker file

- Then add a new section inside SECTIONS segment to tell the linker where to allocate this binary file. You can call this section '.app2' and put it just before section '.data'. Notice this section is contained in segment 'my_text'.

Listing 10 – Application1 new app2 section

Merging applications using Kinetis Design Studio

Finally go to menu Project > Properties > C/C++ Build > Settings > ARM Ltd Windows GCC C Linker > Libraries and add under 'Library search path (-L)' the Sources folder path

"\${workspace_loc:/\${ProjName}/Sources}", this way the linker will be able to find the **Application2** binary file.



_

Now you can build and program the application using 'Flash from file...' option. Just click 'Flash from file...' icon to get the Flash Configurations menu, choose your connection and the .elf file generated by Application1 and click on the 'Flash' button.



Flash Configurations						
Create, manage, and run configurations		4				
	Name: Application1_Debug_OpenOCD					
type filter text	📄 Main 🕸 Debugger 🕨 Startup 🧤 S	Source 🔲 Common				
GDB OpenOCD Debugging GOB OpenOCD Debug OpenOCD	Project:					
C Application2_Debug_OpenOCD	Application1	Browse				
C GDB PEMicro Interface Debugging C GDB SEGGER J-Link Debugging	C/C++ Application:					
	Debug/Application1.elf					
	Build (if required) before launching	Variables Search Project Browse				
	Build configuration: Use Active					
	Enable auto build	🔘 Disable auto build				
	Ose workspace settings	Configure Workspace Settings				
Filter matched 9 of 9 items		Apply Revert				
?		Flash Close				

 Reset your board and you should see the blue LED toggle 5 times which indicates the Application1 is being executed after this the red LED will start to toggle indicating that the program jumped to the Application2.

3.3 Merging applications using the P&E Advanced Flash Programming options

- After creating the two applications following the steps described in the section 3.1 we will now
 proceed to merge them using the P&E Advanced Flash Programming options flash the Application2
 to the MCU.
- For this section we will need our board to have the P&E OpenSDA firmware, you can find more information on how to load this firmware to your board on this link: <u>https://community.freescale.com/docs/DOC-105199</u>
- Build and program the Application2 using 'Flash from file...' option. Just click 'Flash from file...' icon to get the Flash Configurations menu, choose the "GDB PEMicro Interface Debugging" connection, and select the .elf file generated by **Application2**.



Flash Configurations		—
Create, manage, and run configurations	:	5
Image: Second system type filter text □ GDB OpenOCD Debugging □ GDB PEMicro Interface Debugging	Name: Application2_Debug_PNE Main % Debuga Startup Source Common Project: Common Source Common	
C Application1_Debug_PNE C Application2_Debug_PNE GDB SEGGER J-Link Debugging	Application2 C/C++ Application: Debug/Application2.elf	Browse
	Variables Search Project Build (if required) before launching Build configuration: Use Active	Browse
	 Enable auto build Disable auto build Use workspace settings Configure Workspace Settings 	
Filter matched 9 of 9 items	Apply	Revert
?	Flash	Close

 Configure the PEMicro interface settings on the "Debugger" tab, click on Apply then on the 'Flash' button.

Flash Configurations	
Create, manage, and run configurations	4
Create, manage, and run configurations	Name: Application2_Debug_PNE Main Debugger Startup PEMicro Interface Settings Interface: OpenSDA Embedded Debug - USB Port Interface: OpenSDA Embedded Debug - USB Port Refresh Port: 1 USB1 - OpenSDA (43704E48) Refresh Select Device Vendor: Freescale Family: Specify IP Specify Network Card IP E Additional Options Mass erase on connect Use SWD protocol Advanced Options Hardware Interface Power Control (Voltage> Power-Out Jack) ms Provide power to target Regulator Output Voltage Power Up Delay ms Power off target upon software exit 2V Power Up Delay ms
	Debug Shift Freq (KHz) [®] 5000 Delay after Reset and before communicating to target for 0 ms Apply Revert
	4 Flash Close

- The next step is to flash the **Application1** to the board making sure that the **Application2** is not erased, to do this we use the Advanced Flash Programming options from P&E.
- Click on the 'Flash from file...' icon to get the Flash Configurations menu, choose the "GDB PEMicro Interface Debugging" connection and select the .elf file generated by **Application1**.

<u>H</u> elp	
X X 3 - 1	n : {
Flash from file	

Merging applications using Kinetis Design Studio

- · · · · ·		
Create, manage, and run configuration	5	
🗈 🗎 🗶 📄 🎲 🕶	Name: Application1_Debug_PNE	
type filter text	📄 Main 🅸 Debugg 🏷 2 rtup 🦆 Source 🔲 Common	
GDB OpenOCD Debugging	Project:	
C Application1_Debug_PNE	Application1	Browse
C Application2_Debug_PNE	C/C++ Application:	
GDB SEGGER J-Link Debugging	Debug/Application1.elf	
	Variables Search Project	t Browse
	Build (if required) before launching	
	Build configuration: Use Active	•
	© Enable auto build © Disable auto build	
	Use workspace settings <u>Configure Workspace Setting</u>	ngs
		_
		1
Filter matched 9 of 9 items	Apply	Revert
(?)	Flash	Close

- Configure the PEMicro interface settings on the "Debugger" tab and open the "Advanced Options".

Seash Configurations		x
Create, manage, and run configurations		5
Image: Second	Name: Application1_Debug_PNE Main	
Filter matched 9 of 9 items	Apply Revert	
0	Flash	

 On the Advanced options window we will preserve the memory range where the Application2 is located.

Flash Algorithm Selection	Non-Vo	latile Mem	ory Prese	ervation	
Use the following flash algorithm when programming flash data:	Data tha memory erase/pr	Data that reside in a preserved range of memory will be maintained through error (program cycles, Values will be			
freescale_k64fn1m0m12_1x32x256k_pflash.arp	masked to match the row size of the memory.				
Use Alternative Algorithm Browse	V Pres	Preserve this range (Memory Range 0)			
	From:	A000	To:	FFFFF	
JTAG Daisy Chain Settings	Pres	erve this ra	inge (Me	mory Range 1)	
	From:	0	To:	3	
Tap Number: 0 Pre-IR Bits: 0	Preserve this range (Memory Range 2)				
	From:	0	To:	3	
Program Trim Registers					
Program Trim Registers Calculate Trim and Program the Non-Volatile Trim Register Calculate Trim reference frequency is: 32768.0 Hz (V	alid Range: 3	1250.0 to 3	9062.5 H	z)	
Program Trim Registers Calculate Trim and Program the Non-Volatile Trim Register Default trim reference frequency is: 32768.0 Hz Use custom trim reference frequency: 0.0 Hz	alid Range: 3	1250.0 to 3	9062.5 H	z)	
Program Trim Registers Calculate Trim and Program the Non-Volatile Trim Register Default trim reference frequency is: 32768.0 Hz (V Use custom trim reference frequency: 0.0 Hz Program Partition	alid Range: 3	1250.0 to 3	9062.5 H	z)	
Program Trim Registers Calculate Trim and Program the Non-Volatile Trim Register Default trim reference frequency is: 32768.0 Hz (V Use custom trim reference frequency: 0.0 Hz Program Partition Enable Partitioning for the device 0 Partition	alid Range: 3 [13:12] = EEE 7:4] =4'b000	S1250.0 to 3 SPLIT, Part 0, Partition	9062.5 H tition[11: J[3:0] = D	z) 8] = EEESIZE EPART	
Program Trim Registers Calculate Trim and Program the Non-Volatile Trim Register Default trim reference frequency is: 32768.0 Hz (V. Use custom trim reference frequency: 0.0 Hz Program Partition Enable Partitioning for the device 0 Partition Preserve Partitioning for the device 0 EEPROM	alid Range: 3 [13:12] = EEF 7:4] =4'b000 data set size	1250.0 to 3 SPLIT, Part 0, Partition must be w	9062.5 H tition[11: (3:0] = D vithin 0x0	z) 8] = EEESIZE EPART 00x00	

- Click on OK, then Apply and finally on Flash.
- Reset your board and you should see the blue LED toggle 5 times which indicates the Application1 is being executed after this the red LED will start to toggle indicating that the program jumped to the Application2.
- Now that the MCU has both applications flashed we need to generate a single binary file that contains both of the applications so it can be used for factory programming, this can be done by performing a memory dump of the MCU.

- Click on the arrow next to the Debug icon and select the "Debug Configurations..." option.



 On the Debug Configurations window choose the "GDB PEMicro Interface Debugging" connection, go to the "Startup" tab and enable the "Attach to Running Target" option, click on Apply then on Debug.

Ebug Configurations	
Create, manage, and run configurations	- The second sec
Create, manage, and run comgurations	Name: Application1_Debug_PNE Main Debugger Startup Semihosting Settings Console routed to: Center Enable semihosting Console routed to: Telnet GDB client Enable Telnet console Telnet Port: S1794 Load Symbols and Executable Use project binary: Application1.elf Workspace File System Symbols offset (hex): Symbols offset (hex): File System File System Variable offset (hex): Vorkspace File System Workspace File System System Symbols offset (hex): Vorkspace File System Workspace File System System Symbols offset (hex): Vorkspace File System Workspace File System System Ster DC (hex): Verkspace File System GDB run commands: Verkspace Verkspace Yee
Filter matched 15 of 15 items	Apply Revert
?	3 Debug Close

- The Debug perspective will open, suspend the execution of the program.



- Go to the Memory view (Window > Show View > Memory) and add a new address to monitor:

📮 Console	Tasks	🖹 Problems	Executables	Memor	y 🔊	
Monitors					1	£
					Add Men	nory Monitor
1						
		🈹 Monitor Me	emory	— ×	3	
		Enter address	or expression to	monitor:		
		0x0		•		
		?	ок	Cancel		

- The memory information will show up:

📃 Console 🧔 Tasks 🚦	🛛 Problems 🛛 🔘 Execut	ables 📋 Merr	iory 🛛 📑 🛛	1019 1010 🗏		5
Monitors 🕂 🕂	🗶 🦹 0x0 : 0x0 <he< th=""><th>x> 🛛 🕂 Nev</th><th>Renderings</th><th></th><th></th><th></th></he<>	x> 🛛 🕂 Nev	Renderings			
♦ 0x0	Address	0 - 3	4 - 7	8 - B	C - F	
	0000000	00000320	D9040000	05050000	05050000	
	00000010	05050000	05050000	05050000	00000000	-
	00000020	0x10 900000	00000000	00000000	05050000	-
	0000036	0000000	00000000	05050000	05050000	
	00000040	05050000	05050000	05050000	05050000	
	00000050	05050000	05050000	05050000	05050000	
	0000060	05050000	05050000	05050000	05050000	
	00000070	05050000	05050000	05050000	05050000	
	00000080	05050000	05050000	05050000	05050000	
	00000090	05050000	05050000	05050000	05050000	
	000000A0	05050000	05050000	05050000	05050000	
	00000B0	05050000	05050000	05050000	05050000	

- Click on the "Export" button, set the start and end address of the memory, set a file name, select the format of the output file and click on OK:

No	1	Export Memory									—]
Vi		Format: SRecord	{	5			_				
		Start address: 0	_{×0} 2	E	nd address	0x100000	3	Length:	1048576		
		File nar 4 C:\U	Jsers\b382	85\Documents	APP1_APP	2_MERGE.sr	ec	Brow	se		
		SRecord format o	only suppo	rts 32-bit addr	ess spaces.				6		
		?						0	K R	Cancel	
	Cons	sole 🤕 Tasks 🚦	Problen	ns 🜔 Execut	ables 📋 I	Memory 🛛	🗋 🗖 🖞	010 1010		10 -	▽ □ [
Mon	itors	; 🔶	X %	0x0:0x0 <he< th=""><th>o 🛛 🕂</th><th>New Rende</th><th>rings</th><th></th><th></th><th></th><th></th></he<>	o 🛛 🕂	New Rende	rings				
	<u>م</u> (0x0		Address	0 - 3	4 - 3	7 8	- B	C - F		

- A window will show up indicating that the memory content is being exported:

🫞 Memory Export to S-Record Fi	le 🗖 🗖 💌
Transferring Data	
🕅 Always run in background	
	Run in Background Cancel Details >>

Merging applications using Kinetis Design Studio

- Finally you get a single file with both applications that can be used for production purposes:

Appendix A – Binary file generation in KDS

 Open the project in KDS, go to Project -> Properties -> C/C++ Build -> Settings -> Toolchains. Enable the checkbox for "Create flash image" and click on Apply:

Roperties for Application2		
type filter text	Settings	(□ ▼ ⊂) ▼ ▼
 ▷ Resource Builders ▲ C/C++ Build Build Variables Environment 	Configuration: Debu	ug [Active] Manage Configurations
Logging	🛞 Tool Settings 🔇	🖲 Toolchains 🕥 Build Steps 🙅 Build Artifact 🗟 Binary Parsers 😣 Error Parsers
Tool Chain Editor	Name:	GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)
Tools Paths	Architecture:	ARM (AArch32)
Linux Tools Path	Prefix:	arm-none-eabi-
Project References Run/Debug Settings	Suffix:	
Task Repository	C compiler:	gcc
WikiText	C++ compiler:	g++
	Archiver:	ar
	Hex/Bin converter:	objcopy
	Listing generator:	objdump
	Size command:	size
	Build command:	make
	Remove command:	rm
	Toolchain path:	\${eclipse home}//toolchain/bin
	1	(to change it use the <u>global</u> or <u>workspace</u> preferences pages or the <u>project</u> properties page)
	Create flash imag Create extended I	e isting 2
		Restore Defaults Apply
?		OK Cancel

 2) Go to Tool Settings -> Cross ARM GNU Create Flash Image -> General. In the "Output file format (-O) option select Raw binary. Click on Apply and then OK or close the Properties window.



3) Build the project. Once the build process is over, you should find the generated binary file (**.bin** extension) inside of the build folder called "Debug" by default:



Appendix B - References

- KDS webpage: www.freescale.com/kds
- Relocating Code and Data Using the KDS GCC Linker File (.ld) for Kinetis: <u>https://community.freescale.com/docs/DOC-104433</u>
- Kinetis Design Studio videos:
 - Installation of KDS and Kinetis SDK: <u>https://community.freescale.com/videos/3281</u>
 - Installation of OpenSDA Firmware: <u>https://community.freescale.com/videos/3282</u>
 - Debugging with KDS: <u>https://community.freescale.com/videos/3283</u>
 - Building the KSDK demo applications: <u>https://community.freescale.com/videos/3378</u>