AN14377 Continuous SRAM Address Usage on MCXA15x Rev. 1.0 — 26 July 2024

Application note

Document information

Information	Content
Keywords	AN14377, MCXA, continuous SRAM, remap
Abstract	This application note introduces how to configure and use the SRAM X0 Alias to form a continuous SRAM address.



1 Introduction

Configurable continuous SRAM address is a new and beneficial feature on the MCXA series. Continuous SRAM address offers the following benefits:

- Convenient for DMA operation leveraging the continuous SRAM address mapping.
- Suitable for applications that require a large continuous SRAM region, such as graphic display.

This application note considers MCXA156 as an example to demonstrate how to configure and use a continuous SRAM address.

2 Memory map and architecture

This section describes the memory map and memory architecture of MCXA156.

2.1 Memory map

<u>Table 1</u> shows the memory map of MCXA156, which can also be found in the attachment of the Reference Manual. The address of SRAM X0 Alias follows the end of SRAM B2, offering a continuous address space with SRAM A0, A1, A2, A3, B0, B1, and B2. As a result, there are a total of 128 KB SRAMs with continuous address.

Start address (hex)	End address (hex)	Size (KB)	Description			
Code bus memory						
0400000	04001FFF	8	SRAM X0 (Slave Port 0)			
04002000	04002FFF	4	SRAM X1 (Slave Port 0)			
		System RAM				
2000000	20001FFF	8	SRAM A0 (Slave Port 1)			
20002000	20005FFF	16	SRAM A1 (Slave Port 1)			
20006000	20007FFF	8	SRAM A2 (Slave Port 1)			
20008000	2000FFFF	32	SRAM A3 (Slave Port 1)			
20010000	20017FFF	32	SRAM B0 (Slave Port 3)			
20018000	2001BFFF	16	SRAM B1 (Slave Port 3)			
2001C000	2001DFFF	8	SRAM B2 (Slave Port 3)			
2001E000	2001FFFF	8	SRAM X0 Alias			

Table 1. Memory map

Note: The SRAM X0 Alias region is reserved by default.

2.2 Memory architecture

Figure 1 shows the memory architecture of MCXA156. To access the SRAM X0 Alias region address, enable the corresponding CPU0_SBUS, DMA0, and USB0 bits of the AHB matrix remap control (remap) register in the SYSCON module. As a result, the CM33 System Bus, DMA, and USB FS can access up to 128 KB continuous address. After enabling the bits of remap, both SRAM X0 (0x04000000-0x04001FFF) and SRAM X0 Alias (0x2001E000-0x2001FFFF) region can be accessed. When the SRAM X0 Alias address is accessed, the address gets translated to the corresponding SRAM X0 address. Then the user can access the SRAM X0.

NXP Semiconductors

AN14377

Continuous SRAM Address Usage on MCXA15x

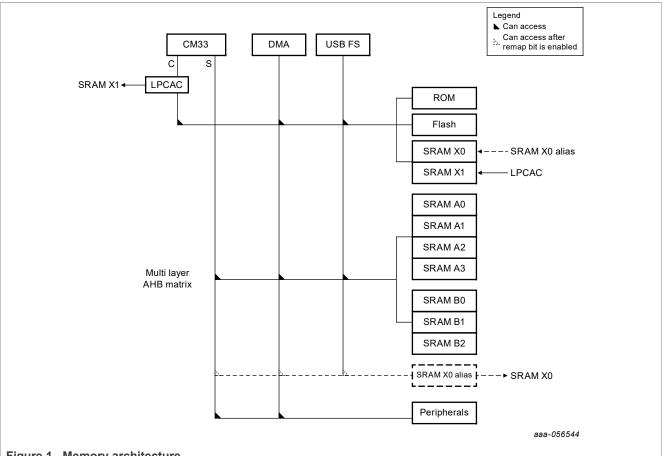


Figure 1. Memory architecture

Note: CM33 accesses SRAM X0 Alias region through the System Bus and accesses SRAM X0 region through the Code Bus.

3 SRAM X0 Alias usage limitations

This section lists the limitations on SRAM X0 Alias usage as follows:

- Enable the corresponding remap bit before accessing the SRAM X0 Alias region.
- Consider the value of the stack pointer (SP).

The user usually wants to define a continuous SRAM address in the linker file and allocate the stack region at the end of the SRAM. The stack pointer is 32-bit and the stack region must be 32-bit aligned, so the initial stack pointer address is usually the defined SRAM end address plus one.

If the user wants to use the SRAM X0 Alias region, the initial stack pointer has to be 0x20020000. The ROM checks the stack pointer before jumping to the extended bootloader.

For the MCXA156 extended bootloader, the valid stack pointer region is 0x20000000 to 0x2001FFFF and 0x04000000 to 0x04002FFF, so 0x20020000 is an invalid address. This is the stack pointer limitation of SRAM X0 Alias usage.

4 Workarounds

This section describes the two workarounds to configure continuous SRAM address.

AN14377 Application note

4.1 Workaround to allocate the STACK space to form continuous SRAM address

To allocate the STACK space at the end of a valid stack pointer region and enable remap bits, perform the following steps:

- 1. Set the initial stack pointer to 0x2001FFFC, which is a valid stack pointer address for ROM validation and is 32-bit aligned.
- 2. The SRAM space from 0x2001FFFD to 0x2001FFFF can be used for data to check the safety of a stack pointer.
- 3. Then, enable the corresponding remap bit before using the stack.

4.2 Workaround to cheat the ROM to validate the SP value successfully to form a continuous SRAM address

To cheat the ROM and form a continuous SRAM address, perform the following steps:

- 1. Set the first word of the vector table to a valid stack pointer value.
- 2. Then, enable the corresponding remap bit.
- 3. Set the SP register to the desired value before using the stack.

Note: The SP value in the vector table is only for ROM checking, not for the real SP value. Therefore, take care when using the SP value in the vector table in the application code.

The following sections describe the detailed steps of this workaround for different IDEs.

4.2.1 MCUXpresso IDE

To modify the project code to implement the continuous SRAM address in the MCUXpresso IDE, perform the following steps:

1. Modify the size of SRAM to include the SRAM X0 Alias region. Also, modify the location and size of SRAMX to reserve the SRAM X0 region to ensure data security, as shown in Figure 2.

Continuous SRAM Address Usage on MCXA15x

	MCU settings							⇔ ▼ ⇔ ▼ 8
 Resource Builders C/C++ Build 	Available	parts						
Build Variables	- SDK MCUs			×	- Preinstalled MCUs	:		
Environment	MCUs from installed	SDKs. Please click above or vi	sit <u>mcuxpresso.nxp.co</u>	om to	MCUs from preinstall	ed LPC and generic Cor	tex-M part support	
Logging	obtain additional SD)Ks.			Target			^
MCU settings	NXP MCXA156			^	> CTNxxx			
Settings	MCXA156				> LPC1102			
Tool Chain Edi	> MCXN2XX				> LPC112x			
> C/C++ General	> MCXN5XX				> LPC11Axx			
MCUXpresso Cor	> MCXN9XX				> LPC11E6x			
Project Natures	> QN909x				> LPC11Exx			
Project Reference:				~	> LPC11U6x			~
> Run/Debug Settir	Target architecture:				cortex-m33			
Task Tags	Preserve memory	configuration						
> Validation	Preserve project o							
		5						
	Memory details (M	ICXA156)*						
	Default LinkServer	Flash Driver						Browse
	Туре	Name	Alias	Lo	ocation	Size	Driver	
	Flash	PROGRAM_FLASH	Flash	0)	кO	0x100000	MCXA1xx.cfx	9 1
	RAM	SRAM	RAM	0)	×2000000	0x20000		×
	RAM	SRAMX	RAM2	0)	x4002000	0x1000		
	Add Flash Add R		Cality Island Dalata		1	Manage Frank	Caracte	
	Add Hash Add H	AM	Split Join Delete		Import.	Merge Export	Generate	
							Refresh	MCU Cache
							Nerresi	rinco cucile
< >							Restore <u>D</u> efaults	Apply
0						/	Apply and Close	Cancel
Figure 2. SR	AM settings in	MCUXpresso IDE						

2. Set the initial stack pointer to a valid value for the ROM, as shown in Figure 3.

<pre>attribute ((used, section(".isr void (* const g_pfnVectors[])(void)</pre>	
// Core Level - CM33	
&_vStackBase,	// Set initial stack pointer to a valid value for ROM
ResetISR,	// The reset handler
NMI_Handler,	// NMI Handler
HardFault_Handler,	// Hard Fault Handler

Figure 3. Set the first word of the vector table in startup_mcxa156.c

3. Enable the corresponding remap bit and set the SP register to the desired value before using the stack, as shown in Figure 4.

<pre>attribute ((naked, section(".after_ve void ResetISR(void) {</pre>	ctors.reset")))
<pre>// Disable interrupts</pre>	
<pre>asm volatile ("cpsid i");</pre>	
// <u>Config</u> VTOR & MSPLIM register	
asm volatile ("LDR R0, =0xE000ED08	\n"
"STR %0, [R0]	\n"
"LDR R1, [%0]	\n"
"MOVS R0, #21	\n"// Enable Remap
"LDR R1, =0×40091200	\n"
"STR R0, [R1]	\n"
"MSR MSP, %2	\n"//Configure SP register to actual value of stack region
"MSR MSPLIM, %1	\n"
:	
: "r"(g pfnVectors),	"r"(_vStackBase), "r"(_vStackTop)
: "r0", "r1", "r2");	
Figure 4. Enable remap and configure the SP rec	gister in startup mcxa156.c

4.2.2 IAR IDE

To modify the project code to implement the continuous SRAM address in IAR IDE, perform the following steps:

1. Modify the m_data_end to include the SRAM X0 Alias region. Also, modify the DATA_region to reserve the SRAM X0 region to ensure data security, as shown in Figure 5.

			= 0x00000000; = 0x000001FF;
			= 0x00000200; = 0x000FFFFF;
			= 0x20000000; = 0x2001FFFF;
			= 0x04000000; = 0x04001FFF;
c	define memo	ory mem with size = 4G;	
			om m_interrupts_start to m_interrupts_end] om m_text_start to m_text_end]; art to m_data_endsize_cstack];
c	define reg	ion CSTACK_region = mem:[from m_data_	endsize_cstack+1 to m_data_end];

Figure 5. SRAM settings in linker file

V

2. Set the initial stack pointer to a valid value for the ROM, as shown in Figure 6.

ector_table	-Sh (CCTACK)	
DCD	sfb(CSTACK)	
DCD	Reset_Handler	
DCD	NMI_Handler	
DCD	HardFault_Handler	
DCD	MemManage_Handler	
DCD	BusFault_Handler	
DCD	UsageFault_Handler	

;Set initial stack pointer to a valid value for ROM

;NMI Handler ;Hard Fault Handler ;MPU Fault Handler ;Bus Fault Handler ;Usage Fault Handler

Figure 6. Set the first word of the vector table in startup_MCXA156.s

3. Enable the corresponding remap bit and set the SP register to the desired value before using the stack, as shown in Figure 7.

R	eset_Handler	
	CPSID	I ; Mask interrupts
	LDR	R0, =0xE000ED08
	LDR	R1, =vector_table
	STR	R1, [R0]
	MOVS	R0, #21 ;Enable Remap
	LDR	R1, =0x40091200
	STR	R0, [R1]
	LDR	R0, =sfe(CSTACK) ;Configure SP register to actual value of stack region
	MSR	MSP, RØ
	LDR	R0, =sfb(CSTACK)
	MSR	MSPLIM, RØ
	CPSIE	I ; Unmask interrupts
	LDR	R0, =SystemInit
	BLX	RØ
	LDR	R0, =iar_program_start
	BX	R0

Figure 7. Enable remap and configure SP register in startup_MCXA156.s

4.2.3 Keil IDE

To modify the project code to implement the continuous SRAM address in the Keil IDE, perform the following steps:

1. Modify the ${\tt m_data_size}$ to include the SRAM X0 Alias region, as shown in Figure 8.

<pre>#define m_interrupts_start 0x0000000 #define m_interrupts_size 0x0000200 #define m_text_start 0x0000200 #define m_text_size 0x0000FE00 #define m_data_start 0x2000000 #define m_data_start 0x2000000 #define m_data_size 0x0000EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE</pre>				
<pre>#define m_text_start #define m_text_start #def</pre>		#defi	ine m_interrupts_start	0x0000000
<pre>#define m_text_size 0x000FFE00 #define m_data_start #define m_data_start #define m_data_start #define m_data_start #define m_data_size igure 8. SRAM settings in linker file . Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. /* Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. /* Set initial stack pointer to a valid value for ROM */ /* Reset Handler */ /* NMI Handler */ /* MMI Handler*/ /* MMI Handler*/ /* Reset Handler */ /* Reset Handler */ /* Reset Handler */ /* Mark Interrupts */ /* Reset Handler: /* Mask interrupts */ /* Mask interrupts */ /* Configure SP register to actual value of stack region */ mar map.r2 /* Unmask interrupts */ idt r r0,=main /* Unmask interrupts */ /* Unmask interrup</pre>		#defi	ine m_interrupts_size	0x0000200
<pre>#define m_text_size 0x000FFE00 #define m_data_start #define m_data_start #define m_data_size igure 8. SRAM settings in linker file . Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. /* Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. /* Set initial stack pointer to a valid value for ROM */ /* Reset Handler */ /* NMI Handler */ /* Mard Fault Handler*/ /* Unmask interrupts */ /* Configure SP register to actual value of stack region */ /* Timer map.r2 /* Unmask interrupts */ /* Configure SP register to actual value of stack region */ /* Timer map.im. */ /* Unmask interrupts */ /* Configure SP register to actual value of stack region */ /* Timer map.im. */ /* Unmask interrupts */ /* Unmask int</pre>		La fi	ing a bout shout	000000000
<pre>idefine m_data_start</pre>				
<pre>idefine m_data_size igure 8. SRAM settings in linker file . Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. Vectors:</pre>		#dell	ING M_CEXC_SIZE	OKOOFFEOO
<pre>idefine m_data_size igure 8. SRAM settings in linker file . Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. Vectors:</pre>		#defi	ine m data start	0x20000000
Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. Vectors: .long Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base /* Set initial stack pointer to a valid value for ROM */ .long NEW_Handler /* Reset Handler */ /* Reset Handler */ .long HardFault_Handler /* NMI Handler*/ .long Reset_Handler /* NMI Handler*/ .long Kest the first word of the vector table in startup_MCXA156.S Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: /* Mask interrupts */ .equ VTOR, OMEOOEDOS .dt r1, = Vectors .st r1, [r0] movs r0, \$21 .dt r1, = 0x40091200 .st r1, [r1] .dt r2, =Image\$ARM_LIB_STACK\$\$ZI\$\$Base mst msplim, r0 .dt r0, =Image\$ARM_LIB_STACK\$\$ZI\$\$Base mst msplim, r0 .dt r0, =Sys				0x00020000
Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. Vectors: .long Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base /* Set initial stack pointer to a valid value for ROM */ .long NEW_Handler /* Reset Handler */ /* Reset Handler */ .long HardFault_Handler /* NMI Handler*/ .long Reset_Handler /* NMI Handler*/ .long Reset_Handler /* NMI Handler*/ igure 9. Set the first word of the vector table in startup_MCXA156.S Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: /* Mask interrupts */ .equ .cqu VTOR, OMEOOEDOS .equ .dt r0, =VTOR .equ .dt r1, = Vectors .equ .equ /* Enable Remap */ .equ .dt r1, =0x40091200 .equ .st r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base .equ .ms msplim, r0 .eqp				
Set the initial stack pointer to a valid value for the ROM, as shown in Figure 9. Vectors: .long Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base /* Set initial stack pointer to a valid value for ROM */ .long NEW_Handler /* Reset Handler */ /* Reset Handler */ .long HardFault_Handler /* NMI Handler*/ .long Reset_Handler /* NMI Handler*/ .long Kest the first word of the vector table in startup_MCXA156.S Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: /* Mask interrupts */ .equ VTOR, OMEOOEDOS .dt r1, = Vectors .st r1, [r0] movs r0, \$21 .dt r1, = 0x40091200 .st r1, [r1] .dt r2, =Image\$ARM_LIB_STACK\$\$ZI\$\$Base mst msplim, r0 .dt r0, =Image\$ARM_LIB_STACK\$\$ZI\$\$Base mst msplim, r0 .dt r0, =Sys	iaure 8. SRA	M settings in linker file		
<pre>Vectors: .long Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base .long Reset_Handler /* Set initial stack pointer to a valid value for ROM */ .long MMI_Handler /* NMI Handler*/ .long HardFault_Handler /* NMI Handler*/ igure 9. Set the first word of the vector table in startup_MCXA156.S . Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: cpsid i /* Mask interrupts */ .equ VTOR, 0xED00ED08 ldr r0, =VTOR ldr r1, =_Vectors str r1, [r0] movs r0, \$21 /* Enable Remap */ ldr r1, =Ox40091200 str r0, [r1] ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msplm, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	-		alid value for the DO	A as shown in Figure 0
<pre>.long Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base .long Reset_Handler .long NMI_Handler .long MATHFAULT_Handler .long HardFault_Handler .long HardFault_Handle</pre>	. Set the init	ai stack pointer to a va	and value for the ROI	vi, as shown in <u>Figure 9</u> .
<pre>.long Rest Handler /* Rest Handler */ .long NMT_Handler /* Rest Handler */ .long HardFault_Handler /* NMT Handler*/ igure 9. Set the first word of the vector table in startup_MCXA156.S . Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10.</pre> Reset Handler: cpsid i /* Mask interrupts */ .equ VTOR, 0xE000ED08 ldr r0, =VTOR ldr r1, =_Vectors str r1, [r0] movs r0, f21 /* Enable Remap */ ldr r1, =0x40091200 str r0, [r1] ldr r2, =Tmage\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp. r2 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main	Vectors:			
<pre>.long NMI_Handler /* NMI Handler*/ .long HardFault_Handler /* Hard Fault Handler*/ igure 9. Set the first word of the vector table in startup_MCXA156.S . Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: cpsid i</pre>	.long I	mage\$\$ARM_LIB_STACK\$\$ZI\$\$	Base	/* Set initial stack pointer to a valid value for ROM *
<pre>.long HardFault_Handler /* Hard Fault Handler*/ igure 9. Set the first word of the vector table in startup_MCXA156.S . Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: cpsid i</pre>	.long R	eset_Handler		/* Reset Handler */
<pre>igure 9. Set the first word of the vector table in startup_MCXA156.S . Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: cpsid i</pre>	.long N	MI_Handler		/* NMI Handler*/
<pre>. Enable the corresponding remap bit and set the SP register to the desired value before using the stack, a shown in Figure 10. Reset_Handler: cpsid i</pre>	.long H	ardFault_Handler		/* Hard Fault Handler*/
<pre>Reset_Handler: cpsid i</pre>				
<pre>cpsid i</pre>			bit and set the SP re	gister to the desired value before using the stack,
<pre>.equ VTOR, 0xE000ED08 ldr r0, =VTOR ldr r1, =_Vectors str r1, [r0] movs r0, #21 /* Enable Remap */ ldr r1, =0x40091200 str r0, [r1] ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>			bit and set the SP re	gister to the desired value before using the stack,
<pre>ldr r0, =VTOR ldr r1, =_Vectors str r1, [r0] movs r0, #21 /* Enable Remap */ ldr r1, =0x40091200 str r0, [r1] ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in <mark>F</mark>	igure 10.		gister to the desired value before using the stack,
<pre>ldr r1, =_Vectors str r1, [r0] movs r0, #21</pre>	shown in F	igure 10. dler: i /* Mas		gister to the desired value before using the stack,
<pre>str rl, [r0] movs r0, #21 /* Enable Remap */ ldr rl, =0x40091200 str r0, [r1] ldr r2, =Tmage\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i</pre>	shown in <u>F</u>	i <mark>gure 10</mark> . dler: i /* Mas VTOR, OxE000ED08		gister to the desired value before using the stack,
<pre>movs r0, #21 /* Enable Remap */ ldr r1, =0x40091200 str r0, [r1] ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i</pre>	shown in F Reset_Hand cpsid .equ ldr	igure 10. dler: i /* Mas VTOR, 0xE000ED08 r0, =VTOR		gister to the desired value before using the stack,
<pre>ldr rl, =0x40091200 str r0, [r1] ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in <u>F</u> Reset_Hand cpsid .equ ldr ldr	igure 10. iler: i /* Mas VTOR, 0xE000ED08 r0, =VTOR r1, =_Vectors		gister to the desired value before using the stack,
<pre>str r0, [r1] ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in F Reset_Han cpsid .equ ldr ldr str	<pre>igure 10. dler:</pre>	k interrupts */	gister to the desired value before using the stack,
<pre>ldr r2, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Limit /* Configure SP register to actual value of stack region */ msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in F Reset_Han. cpsid .equ ldr ldr str movs	<pre>igure 10. dler: i</pre>	k interrupts */	gister to the desired value before using the stack,
<pre>msr msp, r2 ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in F Reset_Hand copsid ldr ldr str movs ldr	igure 10. dler: i /* Mas VTOR, 0xE000ED08 r0, =VTOR r1, = Vectors r1, [r0] r0, #21 /* En r1, =0x40091200	k interrupts */	gister to the desired value before using the stack,
<pre>ldr r0, =Image\$\$ARM_LIB_STACK\$\$ZI\$\$Base msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in F Reset_Hand opsid .equ ldr ldr str	igure 10. dler: i /* Mas VTOR, 0xE000ED08 r0, =VTOR r1, =_Vectors r1, [r0] r0, \$21 /* En r1, =0x40091200 r0, [r1]	ak interrupts */	
<pre>msr msplim, r0 ldr r0,=SystemInit blx r0 cpsie i</pre>	shown in F Reset_Hand opsid .equ ldr ldr str ldr str ldr	<pre>igure 10. dler: i</pre>	ak interrupts */	
<pre>ldr r0,=SystemInit blx r0 cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in F Reset_Hand cpsid .equ ldr ldr str movs ldr str ldr str ldr msr	<pre>igure 10. dler:</pre>	able Remap */ ACK\$\$ZI\$\$Limit /* Conf:	
blx r0 cpsie i /* Unmask interrupts */ ldr r0,=main	shown in F Reset_Ham cpsid .equ ldr str tr ldr str ldr str ldr str ldr	<pre>igure 10. dler:</pre>	able Remap */ ACK\$\$ZI\$\$Limit /* Conf:	
<pre>cpsie i /* Unmask interrupts */ ldr r0,=_main</pre>	shown in F Reset_Hand opsid ldr ldr str ldr str ldr str ldr msr ldr msr	<pre>igure 10. dler:</pre>	able Remap */ ACK\$\$ZI\$\$Limit /* Conf:	
ldr r0,=_main	shown in F Reset_Hand opsid .equ ldr ldr str ldr str ldr msr ldr msr ldr	<pre>igure 10. dler: i</pre>	able Remap */ ACK\$\$ZI\$\$Limit /* Conf:	
	shown in F Reset_Hand opsid .equ ldr ldr str ldr str ldr msr ldr msr ldr blx	<pre>igure 10. dler: i</pre>	k interrupts */ able Remap */ ACK\$\$ZI\$\$Limit /* Conf: ACK\$\$ZI\$\$Base	
	shown in F Reset_Ham cpsid .equ ldr str dr str ldr str ldr msr ldr msr ldr blx cpsie	<pre>igure 10. dler:</pre>	k interrupts */ able Remap */ ACK\$\$ZI\$\$Limit /* Conf: ACK\$\$ZI\$\$Base	

Figure 10. Enable Remap and configure SP register in startup_MCXA156.S

5 Demo validation

This section provides a demo for validating SP register, read and write for boundary unaligned address, and DMA access to continuous SRAM address.

5.1 Hardware and software requirements

Table 2 describes the hardware and software requirements.

Category	Description
Hardware	Board: FRDM-MCXA156Cable: One Type-C USB
Software	IDEs supported: • MCUXpresso 11.9.0 or later • IAR embedded Workbench 9.50.1 or later • Keil MDK 5.38 or later

5.2 Set up

To set up this demo, perform the following steps:

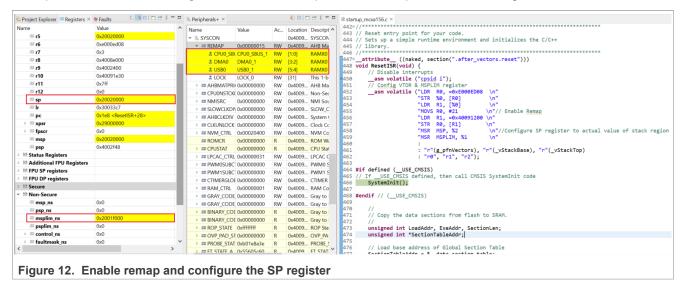
- 1. Use a Type-C USB cable to connect J21 of the FRDM-MCXA156 board and the USB port of the PC.
- 2. Download the project here: https://github.com/nxp-appcodehub/an-continuous-sram-address-mcxa15x or refer to the software attached to this application note.
- 3. Build and download the project to the FRDM-MCXA156 board.

5.3 SP register validation

The first word in the vector table, the initial stack pointer, has a value of 0x2001f000, which is a valid value for ROM, as shown in <u>Figure 11</u>.

Name	Value	Description	435 // created by the finker via the code ked managed finker script mechanism. It 434 // contains the load address, execution address and length of each RW data
MCXA156 (cortex-m33)		frdmmcxa156_hello_world.axf	435 // section and the execution and length of each BSS (zero initialized) section.
···· r0	0x2001f000	Argument/Scratch Register 1	436 //***********************************
²⁰⁰ r1	0x1cd	Argument/Scratch Register 2	<pre>437 extern unsigned intdata_section_table; 438 extern unsigned intdata_section_table_end;</pre>
²⁰⁰ r2	0x2	Argument/Scratch Register 3	439 extern unsigned intbss_section_table;
200 r3	Oxff	Argument/Scratch Register 4	440 extern unsigned int _bss_section_table_end;
²⁸³⁷ r4	0x0	Variable Register 1	441
²⁰⁰ r5	0x4002000	Variable Register 2	4420 //***********************************
²²²¹ r6	0xe000ed08	Variable Register 3	443 // Reset entry point for your code. 444 // Sets up a simple runtime environment and initializes the C/C++
2000 r7	0x3	Variable Register 4	445 // library.
³⁴³¹ r8	0x4008e000	Variable Register 5	446 //**********************************
200 r9	0x4002400	Variable Register 6	447°attribute ((naked, section(".after_vectors.reset")))
2001 r10	0x40091e30	Variable Register 7	448 void ResetISR(void) { 449 // Disable interrupts
²⁰⁰ r11	0x7ff	Variable Register 8	449 // Disable interrupts 8450 asm volatile ("cpsid i");
2001 r12	0x0	Intra-Procedure-Call Scratch Reg	451 // Config VTOR & MSPLIM register
2229 sp	0x2001f000	Stack Pointer (r13)	452asm volatile ("LDR R0, =0xE000ED08 \n"
2007 br	0x30033c7	Link Register (r14)	453 "STR %0, [R0] \n"
2003 pc	Ox1cc <resetisr></resetisr>	Program Counter (r15)	454 "LDR R1, [%0] \n" 455 "MOVS R0, #21 \n"/ Enable Remap
> 2001 xpsr	0x69000000	Program Status Register	456 "LDR RI. =0x40091200 \n"
> IIII fpscr	0x0	Floating Point Status Control Reg	457 "STR R0, [R1] \n"
iiii msp	0x2001f000	Main Stack Pointer	458 "MSR MSP, %2 \n"//Configure SP register to actual value of stack region
¹¹¹¹ psp	0x4002f48	Process Stack Pointer	459 "MSR MSPLIM, %1 \n"
🗥 🛗 Status Registers		Status Registers for cortex-m:	460 : 461 : "r"(g_pfnVectors), "r"(_vStackBase), "r"(_vStackTop)
> IIII apsr	nZCvQ	Application Program Status Regi	462 : "r@", "r1", "r2");
> IIII ipsr	no fault	Interrupt Program Status Registe	463
> IIII epsr	Т	Execution Program Status Regist	464 #if defined (USE_CMSIS)
Additional FPU Registers		Additional FPU Registers for c	465 // IfUSE_CMSIS defined, then call CMSIS SystemInit code
> IIII fpccr	0xc0000000	Floating Point Context Control R 🛩	466 SystemInit();

The corresponding remap bits are enabled, and the value of the SP register is the end address of SRAM X0 Alias plus one, as shown in <u>Figure 12</u>. The above operation is completed before using the stack.



5.4 Read and write test for boundary unaligned address

Figure 13 shows the read and write test for boundary unaligned address as follows:

- The 32-bit *p_test point address is 0x2001DFFF (the last byte of SRAM B2) and the initial value of *p_test is 0x0.
- Then, write OxFFFFFFF to *p test.
- Finally, the value of *p test is read as 0xfffffff.

The 32-bit $*p_test$ address is not 4 bytes aligned and spans SRAM B2 and SRAM X0 Alias, the read and write works fine.

********Read and write test for contiguous SRAM address boundary unaligned address*******
32bit *p_test point address : 0x2001DFFF
Initial *p_test = 0x0
Write 0xFFFFFFF to *p_test
Updated *p_test = 0xfffffff

Figure 13. Access to boundary unaligned address

5.5 DMA access to continuous SRAM address test

This demo also tests the DMA access to this continuous SRAM address in Active and Low power mode, and it works fine.

The destination buffer spans between RAM B2 and RAM X0 Alias, and DMA can transport normally as shown from the log information in <u>Figure 14</u>. In Power Down mode, the SYSCON register is in retention state, and can still use DMA partial wakeup to transport data on continuous SRAM address.

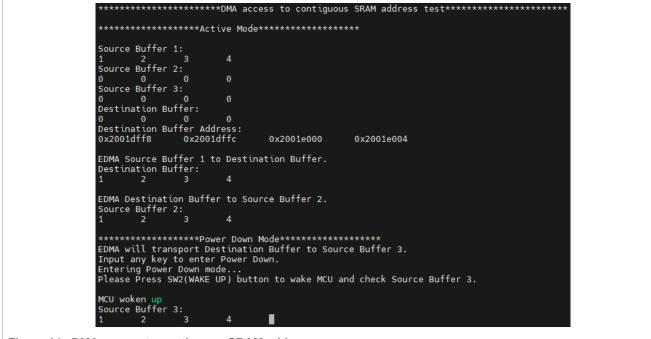


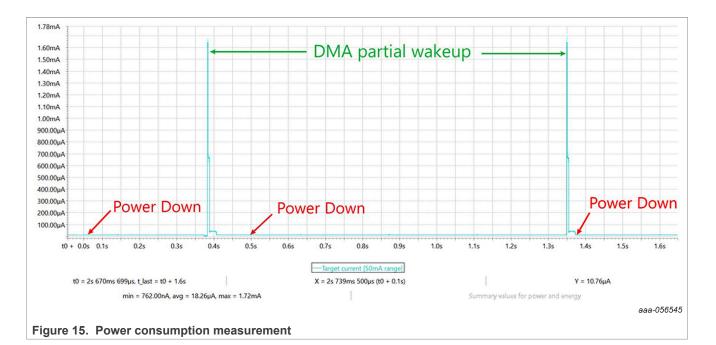
Figure 14. DMA access to continuous SRAM address

Figure 15 shows that the MCU is in Power Down mode and uses DMA partial wakeup to transport data on continuous SRAM address.

NXP Semiconductors

AN14377

Continuous SRAM Address Usage on MCXA15x



6 Summary

This application note introduces how to configure and use the SRAM X0 Alias to form a continuous SRAM address, and provides a demo to validate the feasibility of continuous SRAM address.

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
- 3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

8 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14377 v.1	26 July 2024	Initial public release

Continuous SRAM Address Usage on MCXA15x

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect. Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at <u>PSIRT@nxp.com</u>) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

 $\ensuremath{\mathsf{NXP}}\xspace$ B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners. **NXP** — wordmark and logo are trademarks of NXP B.V.

Continuous SRAM Address Usage on MCXA15x

IAR — is a trademark of IAR Systems AB.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Continuous SRAM Address Usage on MCXA15x

Contents

1	Introduction	2
2	Memory map and architecture	2
2.1	Memory map	
2.2	Memory architecture	
3	SRAM X0 Alias usage limitations	3
4	Workarounds	
4.1	Workaround to allocate the STACK space	
	to form continuous SRAM address	4
4.2	Workaround to cheat the ROM to validate	
	the SP value successfully to form a	
	continuous SRAM address	4
4.2.1	MCUXpresso IDE	4
4.2.2	IAR IDE	
4.2.3	Keil IDE	
5	Demo validation	7
5.1	Hardware and software requirements	7
5.2	Set up	
5.3	SP register validation	8
5.4	Read and write test for boundary unaligned	
	address	8
5.5	DMA access to continuous SRAM address	
	test	9
6	Summary	10
7	Note about the source code in the	
	document	10
8	Revision history	
	Legal information	12

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2024 NXP B.V.

All rights reserved.

For more information, please visit: https://www.nxp.com

Document feedback Date of release: 26 July 2024 Document identifier: AN14377