This section will cover the Programmed Segmentation Control and Extended Virtual Addressing.

Introduction

- Programmed Segmentation Control (PSC)
  - MIPS32® Release 3
  - Flexible memory segments
- Extended Virtual Addressing (EVA)
  - Extend Kernel Spaces
  - Easier for Kernel to Access User Spaces
  - Fully programmable Exception Vector
  - Full programmable BEV using Boot Exception Vector Overlay

Segmentation Control and Extended Virtual Addressing have been added to MIPS32® in Release 3 of the Architecture. The MIPS Architecture is changing with the times, and today many products are demanding larger memory segments. In previous releases of the architecture, the memory scheme is static in the allocation of memory and the assignment of attributes for the different memory segments.

In Release 3, the Programmed Segmentation Control provides more flexible memory segments. An OS such as Linux can use Programmed Segmentation Control to extend kernel and User spaces without losing performance. This is called Extended Virtual Address or EVA.

There are also new instructions that facilitate EVA which allow the Kernel to access User space as if it were executing as a process in user mode. In addition it adds the ability of changing both the boot exception vector and the exception vector virtual and physical address.

Introduction

- What are Segments and how to program them
- How a core is setup for Legacy Compatibility Setting
- How Segments are set up to use 3GB of Ram
- Boot Overlays
- Example of Segmentation Control and Linux using the Malta Evaluation Board

The topics covered in this section are

What are Segments and how to program them

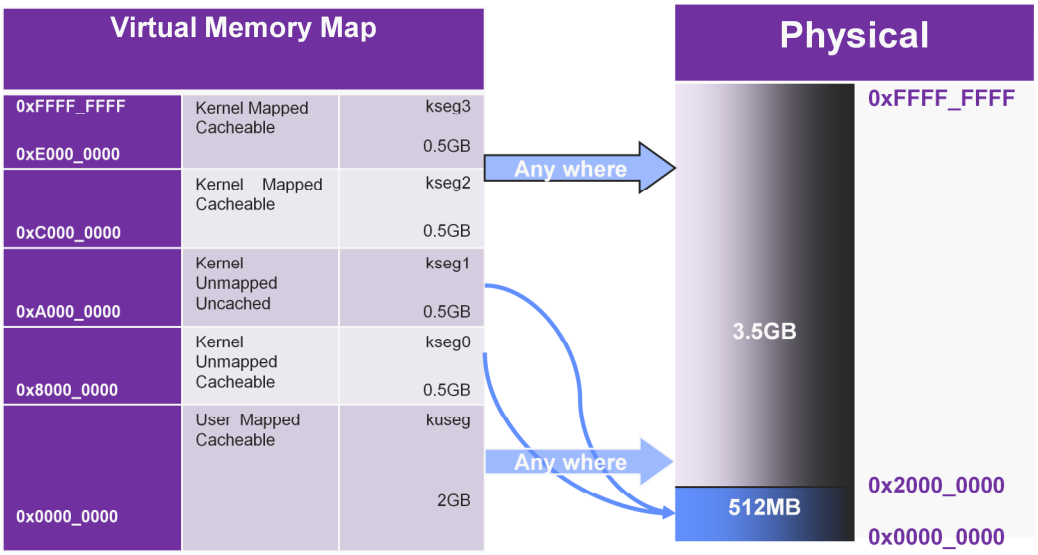
How a core is setup for Legacy Compatibility Setting

How Segments are set up to use 3GB of Ram

Boot Overlays which change the boot exception vector physical address

And an Example of how Linux uses Segmentation Control on our Malta Evaluation Board

Legacy Memory Map With TLB



Prior to Programmed Segmentation Control, each segment of the Address Space was classified as “Mapped” or “Unmapped”.

+ A “Mapped” address is one that is translated through the TLB. KUSEG, KSEG2 and KSEG3 are the Mapped segments.

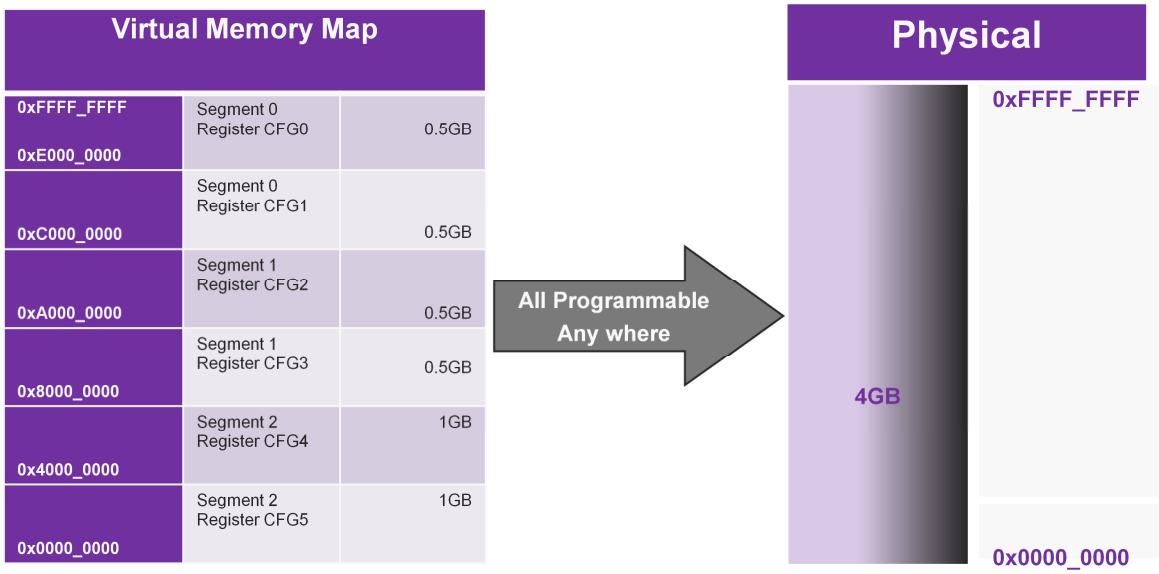
+ An “Unmapped” address is one that is not translated through the TLB. KSEG0 and KSEG1 are unmapped segments and that provide a 512MB window into the lowest portion of the physical address space, starting at physical address zero.

Each segment of an Address Space is associated with one or more of the three processor’s operating modes User, Supervisor, or Kernel. A segment associated with a particular mode is accessible if the processor is running in that mode or a more privileged mode. The Kernel mode is the most privileged mode and has access to the full 4 GBs of virtual and physical address space. The user mode is the least privileged mode and has access to the lowest 2 GBs of virtual address space when a TLB is used.

The thing to take away from this is that up until now each segment was defined with specific and unchangeable attributes.

This mapping is now referred to as Legacy mode. This is also the default segment mapping when the core is built for legacy mode.

Programmed Segmentation Control



Programmed Segmentation Control divides the Virtual memory map into 6 fixed-size virtual segments whose characteristics are fully programmable.

Each segment can be set for mapped or unmapped, Kernel, Supervisor or User mode access, and set for any Cache Attribute. The only characteristic that is fixed is the Virtual Address range of each segment.

Notice that the segment size is different for segments 4 and 5.

Configuration Registers for PSC

- There are three CP0 registers that are used to configure the characteristics of the segments.

Segment number	CP0 Register	Segment Bits	Size	Starting Virtual Address	Ending Virtual Address
0 (CFG0)	SegCtl0 5,2	0 - 15	.5 GB	0xE000 0000	0xFFFF FFFF
1 (CFG1)		16 - 31	.5 GB	0xC000 0000	0xDFFF FFFF
2 (CFG2)	SegCtl1 5,3	0 - 15	.5 GB	0xA000 0000	0xBFFF FFFF
3 (CFG3)		16 - 31	.5 GB	0x8000 0000	0x9FFF FFFF
4 (CFG4)	SegCtl2 5,4	0 - 15	1 GB	0x4000 0000	0x7FFF FFFF
5 (CFG5)		16 - 31	1 GB	0x0000 0000	0x3FFF FFFF

Segments are programmed using three CP0 registers that are used to configure the characteristics of the segments. Each register is divided into two 16-bit sections; each section is used to configure a segment.

Shown is an expanded Segment Address Table with the CP0 registers and the bit field that correspond to each segment.

The size column is the size of the segment and the address columns are the starting and ending virtual address for each segment.

Segmentation Register

Register Fields		(SegCtl0-3 Register Format (CP0 Register 5, Select 2-5))	Reset State
Name	Bits		
PA Upper 16 PA Lower 16	31:25 15:9	Physical address bits 31:29 for segment 1. For use when unmapped. Bits 27:25 correspond to physical address bits 31:29. Bits 31:28 are reserved for future expansion.	Configuration Dependent
AM Upper 16 AM Lower 16	22:20 6:4	Configuration 1 access control mode	Configuration Dependent
EU Upper 16 EU Lower 16	19 3	Error condition behavior. Configuration segment 1 becomes unmapped and uncached when StatusERL = 1.	Configuration Dependent
C Upper 16 C Lower 16	18:16 2:0	Cache coherency attribute for segment 1, for use when unmapped. As defined by the base architecture.	Configuration Dependent

As mentioned in the previous slide there are 3 CP0 segmentation registers. Each of them is split into 2 16 bit parts. Each part is used to configure an address segment. This slide shows the configuration fields for a segment in these registers.

In the name column you can see the field name followed by Upper or Lower bit positions for the 2 segments in each register. There are 4 fields for each segment:

The Physical Address field which will map the segment to a physical address when the segment is configured to be unmapped.

The Access Mode which is the User, Kernel and Supervisor mode and cache state.

The error condition behavior, which when set changes the segment access to uncached and unmapped.

And last the cache coherency of the segment when it is configured unmapped.

The default reset state of these fields depends on the Use Legacy mode selection at Core build time. The settings will be covered in later slides.

Segmentation Register PA Fields

- PA (Physical Address Field)

Bits	Description	Reset
11:9 and 27:25	Physical address bits 31:29 for segment. These bits are used when the virtual address space is configured as kernel unmapped or EU is set and ERL =1, to select the segment in memory to be accessed.	Preset at Core Build

PA	Physical Address
000	0x0000 0000
001	0x2000 0000
010	0x4000 0000
011	0x6000 0000
100	0x8000 0000
101	0xA000 0000
110	0xC000 0000
111	0xE000 0000

When the segment is unmapped, either by the Access Mode or Error Condition, it will use the PA field to select the segment of physical memory that corresponds to the virtual segment. Physical addresses can only be set on a 512 Megabyte boundary so that means only the top 3 bits of the 32 bit physical address can be set and the remaining bits will be 0. The table shows the Physical address that corresponds to the PA settings.

Configuration Register AM Fields

- AM (Address Mode) Bits 6:4 and 22:20 (Preset at Core Build)

Description	Mode Name	Encoding
Unmapped Kernel Segment	UK	000
Mapped Kernel Segment	MK	001
Mapped Supervisor and Kernel Segment	MSK	010
Mapped User, Supervisor and Kernel Segment	MUSK	011
Mapped User and Supervisor and Unmapped Kernel	MUSUK	100
Unmapped Supervisor and Kernel Segment	USK	101
Unrestricted Unmapped Segment	UUSK	111

The Am field contains the three address mode bits. These are the various combinations of address modes a segment can be configured for, mode name and the field encoding.

Configuration Register EU Fields

- EU (Error Condition Behavior) bits 3 and 19 (Preset at Core Build)
  - If set, Configured segment becomes unmapped and uncached when Status.ERL = 1 (reset, NMI or cache error) regardless of any other settings.

When the EU field is set, the segment becomes unmapped and uncached to the physical address encoded in the PA field An error condition is one of reset, NMI or cache error.

Configuration Register C Fields

- C (Cache Coherency Attribute) Bits 2:0 and 18:16 (Preset at Core Build)

Description	Mode Name	Value of bits
Uncached, non-coherent	UC	010
Writeback, write-allocate, non-coherent	WB	011
Writeback, write-allocate, coherent, exclusive	CWBE	100
Writeback, write-allocate, coherent, exclusive on write	CWB	101
Uncached accelerated, non-coherent	UCA	111

The C field sets the cache coherency of the segment if its Address Mode is set to unmapped. The table shows the cache modes with their corresponding bit encodings.

Configuration for MIPS Legacy Memory Map

CP0 Register	Segment	PA Bits 31:29 15:9	AM (Access mode) 6:4	EU 3	C (Cache Coherency Attribute) 2:0	VAR (Virtual Address Range)	LM (Legacy Mode Segment)
SegCtl0 (5,2)	CFG0 Bits 15-0	na	MK - 001 (Mapped Kernel)	0	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3
	CFG1 Bits 31-16	na	MSK - 010 (Mapped Supervisor/Kernel)	0	na (TLB)	0xDFFF FFFF 0XC000 0000	KSEG2 KSSEG
SegCtl1 (5,3)	CFG2 Bits 15-0	000	UK - 000 (Unmapped Kernel)	0	010 (uncached)	0xBFFF FFFF 0xA000 0000	KSEG1
	CFG3 Bits 31-16	000	UK - 000 (Unmapped Kernel)	0	Config0 K0 bit determines CCA	0x9FFF FFFF 0x8000 0000	KSEG0
SegCtl2 (5,4)	CFG4 Bits 15-0	010	MUSK - 011 (Mapped User/Supervisor/Kernel)	1	na (TLB)	0x7FFF FFFF 0x4000 0000	KUSEG
	CFG5 Bits 31-16	000	MUSK - 011 (Mapped User/Supervisor/Kernel)	1	na (TLB)	0x3FFF FFFF 0x0000 0000	

As an example of how to configure all the segments, let's look at how Programmed Segmentation Control is used to configure a core to look just like the legacy memory mapping prior to Programmed Segmentation Control. The slide shows the complete configuration setting for a legacy memory map. The following slides will go through the setting of each segment in detail.

Programming Segments for Legacy KUSEG

- KUSEG Virtual address 0 – 0x7FFF FFFF

1	2	3	4	5	6	7	8
CP0 Register	Segment	PA Bits	AM (Access mode)	EU	C (Cache Coherency Attribute)	VAR (Virtual Address Range)	LM (Legacy Segment)
SegCtl2 (5,4)	CFG4 Bits 15-0	010	MUSK - 011 (Mapped User/Supervisor/Kernel)	1	na (TLB)	0x7FFF FFFF 0x4000 0000	KUSEG
	CFG5 Bits 31-16	000	MUSK - 011 (Mapped User/Supervisor/Kernel)	1	na (TLB)	0x3FFF FFFF 0x0000 0000	

The legacy KUSEG segment was accessible in Kernel, Supervisor, or User Mode with an address range that covers the lowest 2 GB of virtual memory and could be mapped through the TLB to any Physical address. When in error state, KUSEG becomes direct-mapped and uncached to the lowest 2 GB of Physical memory.

In the table SegCtl2 which is CP0 register 5, select 4 is used to program segments CFG4 and CFG5 which correspond to the legacy KUSEG virtual address range. The access mode is 0x011 or Mapped, User, Supervisor and Kernel, which means they will be accessible through the TLB in all modes for non-error conditions.

The Error Condition field, EU is set to 1 to indicate that these segments will become unmapped and uncached when the CPU is in an error state. The translation in this state is configured by the PA bits. Segment CFG5 sets PA to 0, causing a mapping of the segment to start at physical address 0, which will cover the first GB of physical memory. Segment CFG4 sets the PA to 010. This will map the segment to second lowest GB of physical memory.

The C field, the coherency attribute does not apply because the segment is mapped through the TLB.

Programming Segment for Legacy KSEG0

- KSEG0 Virtual address 0x8000 0000 – 0x9FFF FFFF

1	2	3	4	5	6	7	8
CP0 Register	Segment	PA Bits	AM (Access mode)	EU	C (Cache Coherency Attribute)	VAR (Virtual Address Range)	LM (Legacy Mode Segment)
SegCtl1 (5,3)	CFG3 Bits 31-16	000	UK - 000 (Unmapped Kernel)	0	Config0 K0 bit determines CCA	0x9FFF FFFF 0x8000 0000	KSEG0

The legacy KSEG0 was a segment accessible in Kernel mode. It had a 512 MB virtual address range starting at 0x8000 0000. It was cacheable and direct mapped to physical address 0.

In the table, SegCtl1 which is CP0 register 5, select 3 is used to program CFG3. This segment covers the 512 MB virtual address range starting at 0x8000 0000 . The Access Mode is unmapped, Kernel encoded with 000. The PA bits are encoded with to 0x000, so this segment will be directly map to physical address 0. The cache coherency attribute will be determined by the K0 field in the CP0 config0 register, more on this later.

The Error Condition field, EU is set to 0 to indicate that this section will retain these same characteristics when the CPU is in the error state.

Programming Segment for Legacy KSEG1

- KSEG1 Virtual address 0xA000 0000 – 0xBFFF FFFF

1	2	3	4	5	6	7	8
CP0 Register	Segment	PA Bits	AM (Access mode)	EU	C (Cache Coherency Attribute)	VAR (Virtual Address Range)	LM (Legacy Segment)
SegCtl1 (5,3)	CFG2 Bits 15-0	000	UK - 000 (Unmapped Kernel)	0	010 (uncached)	0xBFFF FFFF 0xA000 0000	KSEG1

The legacy KSEG1 was a segment accessible in Kernel mode. It had a 512 MB virtual address range starting at 0xA000 0000. It was uncached and direct mapped to physical address 0..

In the table, the SegCtl1 which is CP0 register 5, select 3 is used to program CFG2. This segment covers the 512 MB virtual address range starting at 0xA000 0000. The access mode encoding is 0x000 or unmapped, Kernel. The PA bits are set to 0x000, so this segment will directly map to physical address 0 covering the first 512 MB of physical memory. The cache coherency attribute is uncached so all memory accesses to this segment will go directly to memory.

The Error Condition field, EU is set to 0 to indicate that this section will retain these same characteristics when the CPU is in the error state.

Programming Segments setting for Legacy , KSEG2/3 or a combination of the two called KSSEG

- KSEG2/3 Virtual address 0xC000 0000 – 0xFFFF FFFF

1	2	3	4	5	6	7	8
CP0 Register	Segment	PA Bits 31:29	AM (Access mode)	EU	C (Cache Coherency Attribute)	VAR (Virtual Address Range)	LM (Legacy Mode Segment)
SegCtl0 (5,2)	CFG0 Bits 15-0	na	MK - 001 (Mapped Kernel)	0	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3
	CFG1 Bits 31-16	na	MSK - 010 (Mapped Supervisor/Kernel)	0	na (TLB)	0xDFFF FFFF 0XC000 0000	KSEG2 KSSEG

KSEG2 and KSEG3 cover 2 512 MB virtual address segments, starting at 0xC000 0000 and 0xE000 0000, respectively. These segments are always mapped, cacheable, and not accessible in User Mode.

In the table the SegCtl0 which is CP0 register 5, select 2 will be used to program these segments. This register controls the configuration for segments CFG1 and CFG0.

Segment CFG1 is a 512mb virtual address space starting at 0xC000 0000 that corresponds to KSEG2. The access mode is encoded 010 setting it to Mapped, which means that CFG1 will be accessible through the TLB in Supervisor and Kernel modes.

Segment CFG0 is a 512mb virtual address space starting at 0xE000 0000 that corresponds to KSEG3. The access mode is encoded 001 or Mapped, Kernel mode, which means CFG0 will be accessible through the TLB in Kernel mode.

Setting Legacy Compatibility Setting

- In Legacy Compatibility Setting the following values are set:
  - CP0 SegCtl0=0x0020 0010
  - CP0 SegCtl1=0x0003 0002
  - CP0 SegCtl2=0x0038 0438
  - CP0 Config5 register is set to 0x8000 0001
    - K=0 (K0 bit controls CCA setting for CFG3 - legacy KSGE0)
    - CV=0 (cache error vectored to 0xA000 0100)
  - CP0 Ebase is set to 0x8000 0000
    - Bits 31:30 Read only set to binary 10
  - GCR Core Local ResetExceptionBase Register=0xBFC0 0000
  - GCR Core-Local ResetExceptionExtendedBase Register=0x4000 0000

Just fixing the segmentation registers does not put the core into a Legacy compatibility Setting. Here is the total list of registers the need to be set for Legacy Compatibility Setting.

Note all of these reregisters will be preset to these values if the Legacy mode option was selected at core build time.

I will go into each of remaining registers in detail in the upcoming slides.

Config5 (CP0 Register 16, Select 5)

Name	Bits	Description	Read/Write	Reset state
K	30	0: Config K0 over rides segment configuration cache mode for CFG3 (Legacy KSEG0) Segment. 1: Config K0 disabled.		
		SI_EVAReset pin de-asserted at reset	RW	0
		SI_EVAReset pin asserted at reset,	R	1
CV	29	Cache error exception vector control. Disables logic forcing use of CFG2 (Legacy KSEG1) segment in the event of a Cache Error exception when StatusBEV = 0.	R/W	0
EVA	28	This bit is always a logic one to indicate support for enhanced virtual address (EVA).	R	1

There are 3 fields in the CP0 Config 5 register that are used to configure the core for Legacy compatibility.

The K bit controls the use of the K0 field in the CP0 Config register. For non-EVA legacy Cores the K0 field controls the cache coherency attribute of the KSEG0 virtual memory segment. To make an EVA core compatible with legacy software, the CFG3 segment that covers the same range of virtual address as did the KSEG0 segment can have its cacheability controlled by the K0 field. If the K field is 0 then the cacheability setting in the K0 bit will override the C field in the CFG3 segment configuration. If the K bit is 1 then the setting of the K0 field will have no effect on the cacheability of the CFG3 segment. The K bit is set by the SI_EVAReset pin on reset. If the pin is deserted then the reset state is 0 and the cacheability of CFG3 is controlled by the K0 field and the K bit is writable which enables the switching from Legacy compatibility to EVA Setting. If the pin is asserted the K0 field has no effect, the K field is set to 1 and is read only disallowing any change from EVA Setting to Legacy compatibility setting.

In addition to selecting the location of the cache coherency attributes for the CFG3 segment, the CONFIG5.K bit also causes hardware to generate two virtual boot exception overlay segments to be compatible with Legacy settings, of 0xBfC0 0000 and mirrored at 0x9FC0 0000. I will go into boot overlays in upcoming slides.

The CV field controls the cache error exception vector. For non EVA legacy Cores, the cache error exception is forced to virtual address 0xA000 0100 located in the uncached KSEG1 memory segment. To make an EVA core compatible with legacy software, when the CV bit is cleared bits 31:29 are

EBase Register

- CP0 EBase bit 11 “WG” Write Gate EBase Override

Register Fields		EBase Register (CP0 Register 15, Select 1)	Reset State
Name	Bits		
Exception Base	31:12	This field specifies the base address of the exception vectors when StatusBEV is zero.	0x80000
WG	11	Write gate. Bits 31..30 are unchanged on writes to Ebase when WG=0 in the value being written. The WG bit must be set true in the written value to change the values of bits 31..30.	0
CPUNum	9:0	This field specifies the number of the CPU in a multi-processor system	preset

The CP0 EBase register has three fields.

The Exception Base field sets bits 12 – 31 of the exception base address. This is defaulted to 0x8000 00 at reset.

The WG bit controls the writing of bits 31:39 of the Exception Base address. For non EVA legacy Cores, bits 31:30 of the Exception Base address are not writable and are set to a binary 10. This forces the Exception to a virtual address in the legacy KSEG0 and KSEG1 segments.

To make an EVA core compatible with legacy software, on reset bits 31:30 are set to a binary 10 and these bits are unchanged on writes to Ebase when WG=0 in the value being written. This forces the exception bass address into the CFG2 and CFG3 segments which correspond to the old KSEG0 and KSEG1 legacy segments.

If WG=1 in the written value then bits 31:30 are over written.

The CPUNum field is the CPU number of the processor executing the read of this register.

Boot Exception Vector Overlay

- Changing the Boot Exception Vector Addresses
  - Change the Virtual address.
    - Allows for more contiguous virtual memory
  - Change the Physical address
    - Allows for more contagious physical memory
  - Together allows for each core to use different boot code

All EVA Cores have a new feature called the Boot Exception Vector overlay. This overlay maps a virtual Boot Exception Vector to a Physical address overlaying any segment configuration. This is done to add more flexibility because both the virtual and physical address of the Boot Exception Vector can be changed and are no longer limited to 0xBFC0 0000 and physical address 0x1FC0 0000.

This is called the BEV Overlay because it overlays part of the configuration for the memory segment it is in. The BEV overlay is always present whether or not the core is in EVA Setting. The BEV Overlay is predefined at core build time so core 0 will always use the overlay mapping that was built into the core. Core 0 can make changes in the BEV overlay for the other cores before Core 0 powers up the other cores. That way they can run different boot code. The next 2 GCR registers are used to configure BEV Overlay.

GCR Core Local Reset Exception Base Register

- Setting Boot Exception vector
  - Value is a core build time setting
  - For Legacy Setting boot it must be built with 0xBFC0 0000

Register Fields		Core-Local Reset Exception Base Register (GCR_Cx_RESET_BASE Offset 0x0020)	Reset State
Name	**Bits**		
BEVExceptionBase	31:12	Bits [31:12] of the virtual address that the local core will use as the exception base in the boot environment (C0P0 StatusBEV=1).	0xBFC00

The Boot Exception Base field in the Core Local Reset Exception Base Register controls where the CPU will fetch the first instruction from on cold reset. For Legacy Setting, the start of the Boot Exception Vectors is located at the virtual address of 0xBFC0 0000 set as the default state for the GCR Core Local Reset Exception Base Register. If you want a Core to cold boot from a different address not legacy Setting then this register can be configured a IP configuration time for a different address.

If you wanted to have CPUs that used different boot code you could access this register through the core-other group from another CPU and set the boot address for this CPU. This register also depends on the settings in the next register the Core Local Reset Exception Extended Base Register.

Bits 12 through 28 will also be used for bits 12 through 28 of the Physical address for the boot exception vector.

Core Local Reset Exception Extended Base Register

Register Fields		Core-Local Reset Exception Extended Base Register (GCR_Cx_RESET_EXT_BASE Offset 0x0030)	Reset State
Name	Bits		
EVAReset	31	If set - Indication to the local core to not use the legacy reset (RW)	Build Option
LegacyUse ExceptionBase	30	If set - Indication to the core to not use the ExceptionBase Forces bits 31-30 of the exception Base Register to 10. (RW)	Build Option

To see what the BEV overlay is and to change the BEV overlay there is a Global Configuration Register called Core-Local Reset Exception Extended Base register. This register is an extension to the Core-Local Reset Exception Base Register. The value is used for placing the boot exception vectors within the physical address map during core boot-up time. This is a per core register so it is in the Core-Local section of the Global Configuration Registers.

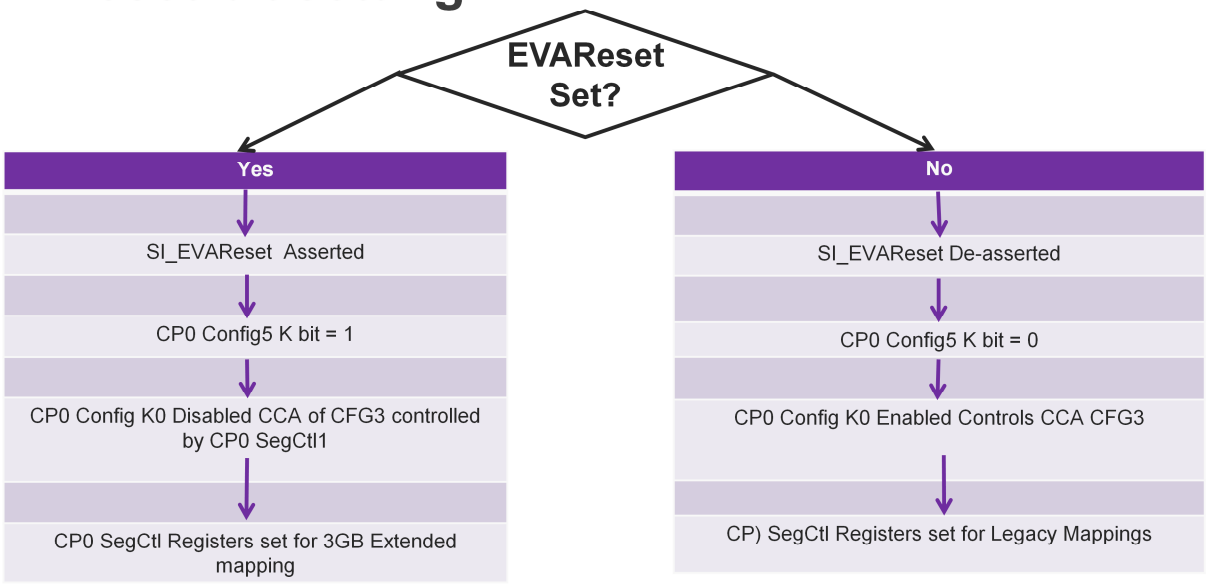
The initial value of EVAReset is set at IP configuration time. The EVAReset bit controls the SI_EVAReset pin. This pin is driven by the CM to the core.

If EVAReset is 0 then the SI_EVAReset pin is de-asserted which drives the K bit to 0. In this case the K0 field in the CP0 Config register will control the CCA of CFG3 and the CP0 segmentation control registers will reflect Legacy mappings for KUSEG, KSEG0, KSEG1 and KSEG2/3. In other words it will map and behave just like a Legacy core.

If the EVAReset is set then the SI_EVAReset pin will be asserted at boot and the K bit will be set and be unchangeable. This means that the state of the core is not Legacy compatible and can never be set for legacy compatibility. The k0 bit in the CP0 Config register is disabled and the CCA of CGF3 is controlled by the settings in the SegCtl register. The CP0 segmentation control registers will reflect an EVA mapping for a 3GB RAM region. The details of the mapping will be shown in a later slide.

The LegacyUseExceptionBase bit will force the Boot Exception vector address that is in the Core-Local Boot Exception Base Register to be located in segments CFG2 and CFG3 which correspond to the Legacy KSEG0 and KSEG1 by overriding bits 31:30 of the Boot Exception Vector address and forcing them to be 1:0.

EVAReset bit setting



Here is a summary chart of the effects of the setting of the EVAReset bit.

Core Local Reset Exception Extended Base Register

Register Fields		Core-Local Reset Exception Extended Base Register (GCR_Cx_RESET_EXT_BASE Offset 0x0030)	Reset State
Name	Bits		
BEVExceptionBaseMask	27:20	Bits [27:20] of the virtual address that the local core will use as the exception base in the boot environment (C0P0 StatusBEV = 1). (RW)	Build Option

Mask bits 27-20	Size in MB
0000 0000	1
0000 0001	2
0000 0011	4
0000 0111	8
0000 1111	16
0001 1111	32
0011 1111	64
0111 1111	128
1111 1111	256

continued on next slide

Overlay With 16MB Mask and Physical address of BEV of 0xBFC0 0000	
0xBFFF FFFF	End of Overlay Region
0xBFC0 0000	Boot Exception Vector
0xBF00 0000	Start of Overlay Region

Continuing with the Reset Exception Extended Base register; The BEVExceptionBaseMask determines the size of the Overlay region from 1 MB to 256 MB in powers of two. The initial value is set at core build time. The size also determines where the overlay starts. The overlay will start on a boundary that corresponds to the size of the overlay.

+ Here is a table that shows the mask bit encodings and the associated overlay size.

+ for example if the physical address of the Boot Exception vector were 0xBFC0 0000 and the Base Mask was set to 16MB then the start of the Overlay region would be 16MB boundary of 0xBF00 0000 and end at 0xBFFF FFFF.

Core Local Reset Exception Extended Base Register

Register Fields		Core-Local Reset Exception Extended Base Register (GCR_Cx_RESET_EXT_BASE Offset 0x0030)	Reset State
Name	Bits		
BEVException BasePA	7:1	Bits [35:29] of the physical address that the local core will use as the exception base in the boot environment (C0P0 StatusBEV = 1). (RW)	Build Option
PRESENT	0	Reads as 0x1. Writes ignored. (R)	1

Segment	31:29	28:0	Starting Hex Address
CFG0	111	0 0000 0000 0000 0000 0000 0000 0000	0xE000 0000
CFG1	110	0 0000 0000 0000 0000 0000 0000 0000	0xC000 0000
CFG2	101	0 0000 0000 0000 0000 0000 0000 0000	0xA000 0000
CFG3	100	0 0000 0000 0000 0000 0000 0000 0000	0x8000 0000
CFG4	010	0 0000 0000 0000 0000 0000 0000 0000	0x4000 0000
CFG5	000	0 0000 0000 0000 0000 0000 0000 0000	0x0000 0000

The Physical Base address is set using the BEV Exception Base PA. The Boot Exception Vector Base Physical address is a 7 bit field but the current cores only uses the first 3 bits. The remaining lower bits 28:0 of the address come from the Core Local Reset Base Register.

+ Having control of the top three bits allows the physical address of the boot exception vector to be placed in any Segment. This address and BEV Exception Base Mask then determines where the Overlay will be in memory as shown in the previous slide.

The Present bit is always set if this register is present. It is a read only bit.

Overlay for Legacy Compatibility Setting

- Place the Boot exception vector at 0xBFC0 0000 Virtual and Physical 0x1FC0 0000 with a size of 1MB.
  - Core-Local Reset Exception Base Register = 0xBFC0 0000
  - Core-Local Reset Exception Extended Base Register = 0x4000 0000
    - EVAReset bit 31 = 0 indicating to use Config K0 field to determine the CCA for CFG3 (Config 5 K bit defaults to 0).
    - LegacyUseExceptionBase bit 30 = 1 indicating to use the complete address in the Core-Local Reset Exception Base Register
    - BEVExceptionBaseMask bits 27:20 = 0000 0000 (0x00) to set the size of the address range to map to 1MB.
    - BEVExceptionBasePA bits 7:1 = 0000000 (0x0) combining this with bits 28:12 of the Core-Local Reset Exception Base Register sets the Physical address to 0x1FC0 0000

The Boot Exception Vector Overlay is always present. For a core that boots with legacy settings the Core-Local Reset Exception Base Register and the Core-Local Reset Exception Extended Base Register are preset at core build time with default values to match the Legacy virtual addresses.

Here are the legacy settings for a BEV overlay of 1MB.

+ The legacy virtual base address is set in the Core-Local Reset Exception Base Register.

+ The EVAReset bit is 0 to indicate that this is a legacy setting boot.

+ The LegacyUseExceptionBase is set to indicate the complete address of the Core-Local Reset Exception Base Register will be used for the virtual address.

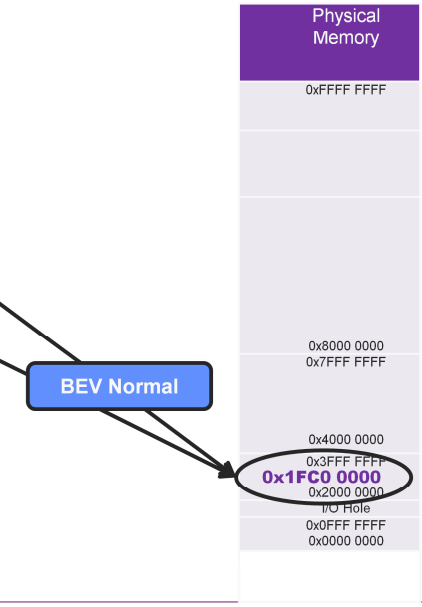
+ The BEVExceptionBaseMask is set to 0 to indicate a overlay size of 1MB.

+ And the BEVExceptionBasePA is set to 0 so the top three bit of the address are cleared. When combined with bits 0 – 28 of the BEV set in the Core-Local Reset Exception Base Register this set the physical address to 1FC0 0000 .

Note that the physical address can be set to a address other than 1FC0 0000 and the core still be considered legacy setting because it still uses the virtual boot addresses of 0xBFC0 0000.

Boot Exception Vector Overlay Legacy Compatibility (EVAReset = 0)

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	E U	C (cache coherency attribute)	VAR (Virtual Address Range)	LM (Legacy Mode)
SegCtl0 (5,4)	CFG 0	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3
	CFG1	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xDFFF FFFF 0XC000 0000	KSEG2
SegCtl1 (5,3)	CFG2	000	UK 000 (Unmapped Kernel)	1	010 (uncached)	0xBFFF FFFF 0xBFC0 0000 0xA000 0000	KSEG1
	CFG3	000	UK 000 (Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x9FFF FFFF 0x9FC0 0000 0x8000 0000	KSEG0
SegCtl2 (5,2)	CFG4	010	MUSK 011 (Mapped User, Supervisor and Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x7FFF FFFF 0x4000 0000	KUSEG
	CFG5	000	MUSK 011 (Mapped User, Supervisor and Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x3FFF FFFF 0x0000 0000	



Here is the table of the Segment Control registers combined with the boot exception Vector overlay when EVAReset = 0 putting the core into Legacy Compatibility Setting as discussed in the previous slide.

Overlay Example

- Place the Boot exception vector at 0xBFC0 0000 Virtual/Physical - size of 16MB.
  - Core-Local Reset Exception Base Register = 0xBFC0 0000
  - Core-Local Reset Exception Extended Base Register = 0x87F0 000B
    - EVAReset bit 31 = 1 indicating to use this Core-Local Reset Exception Extended Base Register and the Core-Local Reset Exception Base Register to determine the placement of the boot exception vector.
    - LegacyUseExceptionBase bit 30 = 0 indicating to use the complete address in the Core-Local Reset Exception Base Register
    - BEVExceptionBaseMask bits 27:20 = 0000 1111 (0x0F) to set the size of the address range to map to 16MB.
    - BEVExceptionBasePA bits 7:1 = 0000101 (0x5) combining this with bits 28:12 of the Core-Local Reset Exception Base Register sets the Physical address to 0xBFC0 0000

Overlay With 16MB Mask and address of BEV of 0xBFC0 0000		
Virtual	Physical	
0xBFFF FFFF	0xBFFF FFFF	End of Overlay Region
0xBFC0 0000	0xBFC0 0000	Boot Exception Vector
0xBF00 0000	0xBF00 0000	Start of Overlay Region

I'll go through what you would see in the Registers if the Core is set up at build time to boot in EVA mode with the Boot Exception Vector at 0xBFC0 0000 both for the virtual and Physical address, instead of physical address of 0x1FC0 0000, with a 16MB. Overlay size.

+The Boot Exception Base field in the Core Local Reset Exception Base Register will be set to 0XBFC0 0000 which sets the virtual address of the Boot Exception Vector.

+ The Core Local Reset Exception Extended Base Register would be set to 0x87F0 000B. Here is how that value is arrived at:

+ The EVAReset bit will be set indicating the addresses configured by the Core-Local Reset Exception Extended Base and the Core-Local Reset Exception Base Register will be used for the Virtual and Physical address of the boot exception vector.

+ The LegacyUseExceptionBase will be cleared so bits 31:30 will remain as they are in the Core-Local Reset Exception Base Register.

+ The BEVExceptionBaseMask bits 27:20 are set to 0x0F this sets the size of the address range to map to 16MB.

+ The BEVExceptionBasePA bits 7:1 are set to 0x5 combining these with bits 28:12 of the Core-Local Reset Exception Base Register sets the Physical address to 0xBFC0 0000

+ Here is the Map again showing the overlay region in relationship to the Boot Exception Vector.

The K bit in the CP0 Config 5 register would also be set which disables the K0 field in the CP0 Config 0 register so it will no longer control the Cache Attributes of the CFG3 region.

Boot Exception Vector Overlay

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	EU	C (cache coherency attribute)	VAR (Virtual Address Range)	LM (Legacy Mode)	Physical Memory
SegCtl0 (5,4)	CFG 0	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3	
	CFG1	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xDFFF FFFF 0xC000 0000	KSEG2	I/O registers and GCR Base
SegCtl1 (5,3)	CFG2	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	010 (uncached)	0xBFFF FFFF 0xBFC0 0000 0xBF00 0000		0xBFFF FFFF 0xBFC0 0000 0xBF00 0000
	CFG3	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0xA000 0000 0x9FFF FFFF 0x9FC0 0000 0x8000 0000	KSEG1 KSEG0	0xA000 0000 0x9FFF FFFF 0x8000 0000
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclueive on write)	0x7FFF FFFF 0x4000 0000	KUSEG	0x7FFF FFFF Contiguous RAM 0x4000 0000
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x3FFF FFFF 0x0000 0000		0x3FFF FFFF 0x0000 0000

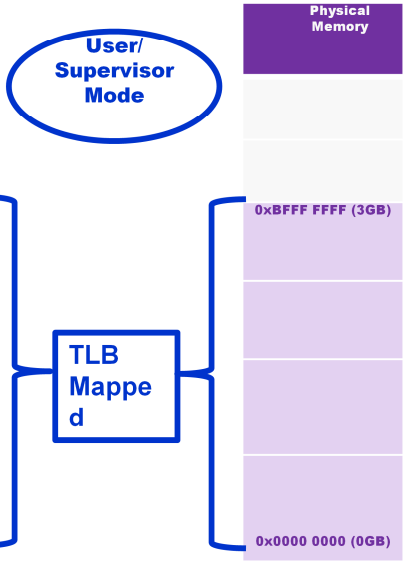
BEV OVERLAY

Here is an example of a 2GB physical Ram memory map when the boot exception vector overlay moves the physical address as setup in the previous slide. If the system is set up with an overlay that changes the physical address of the boot exception vector to the BFC0 0000 and if the I/O register placement was also placed at an address above the first 2 GB then there is no longer a hole preventing a contiguous block of RAM memory.

NOTE the Default value of the Global Configuration Base register is set to 0x1FBF 8000 this will also need to be changed to a address outside of the RAM memory space. The I/O registers and the GCRs can also be located with in the BEV overlay region which is preferable because it is guaranteed to be unmapped and uncached.

Expanding KUSEG/KEG0/1 to an Overlapping 3 Gigabyte Virtual Memory Segment

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	EU	C (cache coherency attribute)	VAR (Virtual Address Range)
SegCtl0 (5,4)	CFG0	NA	MK 001 (Mapped Kernel)	0	NA (TLB)	0xFFFF FFFF 0xE000 0000
	CFG1	NA	MK 001 (Mapped Kernel)	0	NA (TLB)	0xDFFF FFFF 0XC000 0000
SegCtl1 (5,3)	CFG2	101	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0xBFFF FFFF 0xA000 0000
	CFG3	100	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x9FFF FFFF 0x9FC0 0000 0x8000 0000
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x7FFF FFFF 0x4000 0000
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x3FFF FFFF 0x0000 0000



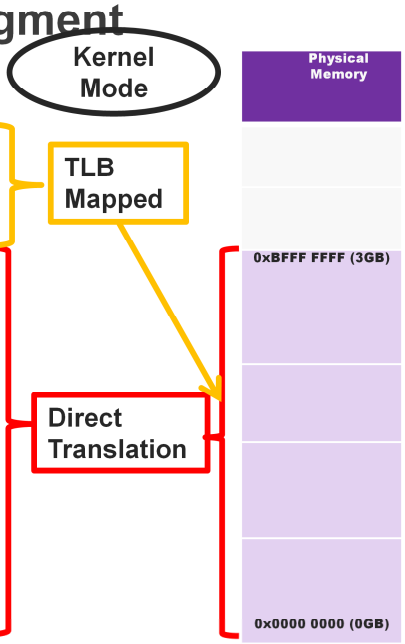
Next I will cover how to setup the segmentation register to expand the Kernel and user mapped virtual and physical address range to 3 gigabytes.

The main purpose of Programmed Segmentation Control is to expand the virtual and physical address space available in User mode, to expand the unmapped virtual and physical address space in Kernel mode, and to be able to overlap the two to make it easy for an OS Kernel to access the User address space.

+ This table shows just that. It configures a virtual memory from 0x0000 0000 to 0xBFFF FFFF as the first 3 GB of virtual memory, accessible in User/Supervisor mode as a mapped region.

Expanding KUSEG/KEG0/1 to an Overlapping 3 Gigabyte Virtual Memory Segment

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	EU	C (cache coherency attribute)	VAR (Virtual Address Range)	Physical Memory
SegCtl0 (5,4)	CFG 0	NA	MK 001 (Mapped Kernel)	0	NA (TLB)	0xFFFF FFFF 0xE000 0000	
	CFG1	NA	MK 001 (Mapped Kernel)	0	NA (TLB)	0xDFFF FFFF 0xC000 0000	
SegCtl1 (5,3)	CFG2	101	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (uncached)	0xBFFF FFFF 0xA000 0000	0xBFFF FFFF (3GB)
	CFG3	100	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x9FFF FFFF 0x9FC0 0000 0x8000 0000	
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x7FFF FFFF 0x4000 0000	
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x3FFF FFFF 0x0000 0000	0x0000 0000 (0GB)



For that same virtual range, it sets up Kernel mode access as unmapped, directly translated to the lower 3 GB of physical memory. It does this by using segments CFG2 through CFG5.

+ The top two virtual memory segments CFG 0 and 1 are set up as mapped Kernel mode only.

Expanding KUSEG/KEG0/1 to an Overlapping 3 Gigabyte Virtual Memory Segment

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	EU	C (cache coherency attribute)	VAR (Virtual Address Range)	Physical Memory
SegCtl0 (5,4)	CFG 0	NA	MK 001 (Mapped Kernel)	0	NA (TLB)	0xFFFF FFFF 0xE000 0000	
	CFG1	NA	MK 001 (Mapped Kernel)	0	NA (TLB)	0xDFFF FFFF 0XC000 0000	
SegCtl1 (5,3)	CFG2	101	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0xBFFF FFFF 0xA000 0000	0xBFFF FFFF (3GB)
	CFG3	100	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x9FFF FFFF 0x9FC0 0000 0x8000 0000	
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x7FFF FFFF 0x4000 0000	
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	011 (Writeback)	0x3FFF FFFF 0x0000 0000	0x0000 0000 (0GB)

Error Mode

Address Error

Direct Translated

In error mode the lower 3GB of virtual address become directly translated and uncached

New Load and Store Instructions to support EVA

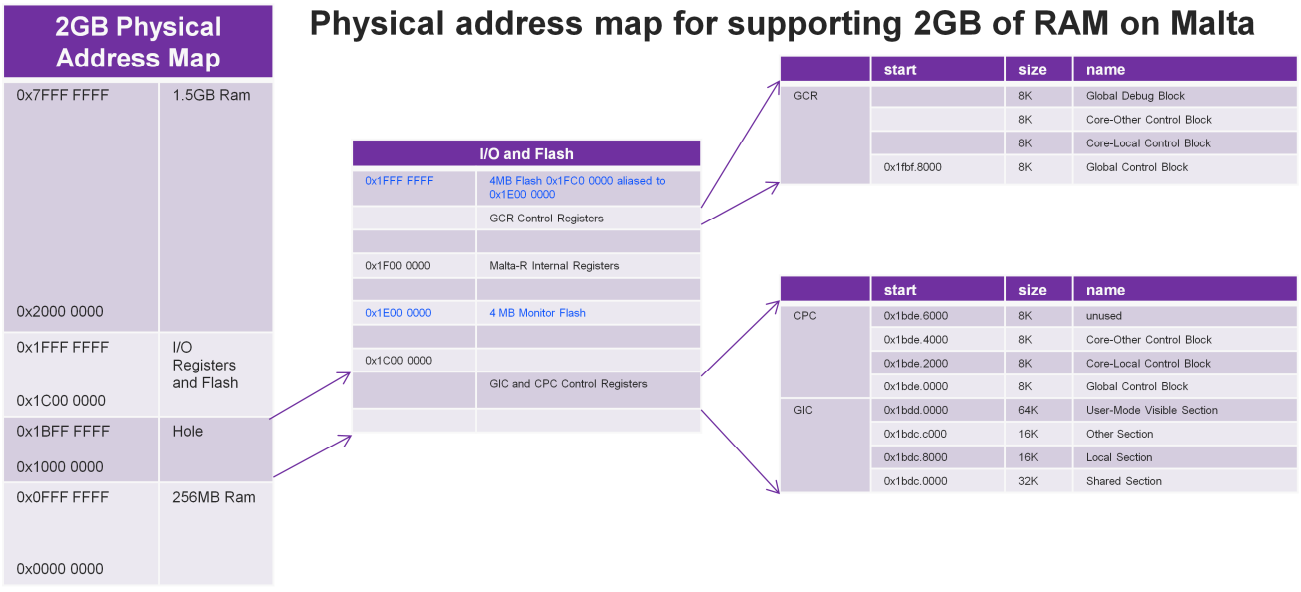
- Allow processor executing in kernel mode to access user address without the need to map to the user address.
  - ASID in the CP0 EntryHi register is used for the Kernel to access as if it were the user mode process if the user data is in a Segment configured with a AM of MUSK, MUSUK or UUSK. These instructions can only be used in Kernel mode.
- New Load 'E' Instructions are:
  - LBE, LBUE, LHE, LHUE, LLE, LWE, LWLE, and LWRE
- New Store 'E' Instructions are:
  - SBE, SCE, SHE, SWE, SWLE, and SWRE

The MIPS32 R3 architecture includes load and store instructions that allow the Kernel to access User space as if it were the process whose ASID is currently set in the CP0 EntryHi register.

The E instructions function in exactly the same fashion as there counterparts, except that address translation is performed using the user mode virtual address space mapping in the TLB. The memory segment must be configured to use the Mapped User Supervisor and Unmapped Kernel access mode, Unmapped User, Supervisor and Kernel or Mapped User or Supervisor and Kernel access mode.

The names for these instructions just have an E appended to the normal non-EVA load/store instruction name. All of these instructions have the same meaning as their non-EVA counterparts, except that they are Kernel-mode instructions that use the User-mode translation of the address for the load or store, based on the current EntryHi ASID value.

Linux Example using Programmed Segments



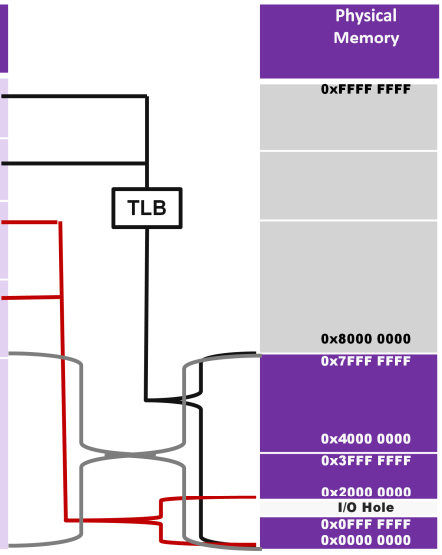
The next slides will show the use of Segmentation and EVA when used with Linux. This is example uses a MIPS Malta Evaluation Board for a test platform. The Evaluation Board is limited to a maximum addressable memory of 1.75 GB due to a limitation of the memory controller on the board.

The 1st RAM region is 256MB starting at physical 0. The 2nd RAM region occupies the region between 2 and 8 million hex, a 1.5GB region. There is an “IO hole” between the two RAM memory regions used for the boot flash, I/O device registers and the memory mapped GCRs.

Since IO address used is the same as a legacy core’s no device drivers needed to change.

EVA Memory Map For Linux on Malta FPGA - 2GB Ram Kernel Mode

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	EU	C (cache coherency attribute)	VAR (Virtual Address Range)	LM (Legacy Mode)
SegCtl0 (5,4)	CFG 0	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3
	CFG1	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xDFFF FFFF 0XC000 0000	KSEG2
SegCtl1 (5,3)	CFG2	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	010 (uncached)	0xBFFF FFFF 0xA000 0000	KSEG1
	CFG3	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x9FFF FFFF 0x8000 0000	KSEG0
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x7FFF FFFF 0x4000 0000	KUSEG
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x3FFF FFFF 0x0000 0000	



The next slides show the values programmed into the segmentation registers and how they pertain to the different processor modes.

This first memory map is of the Linux Kernel when running in kernel mode.

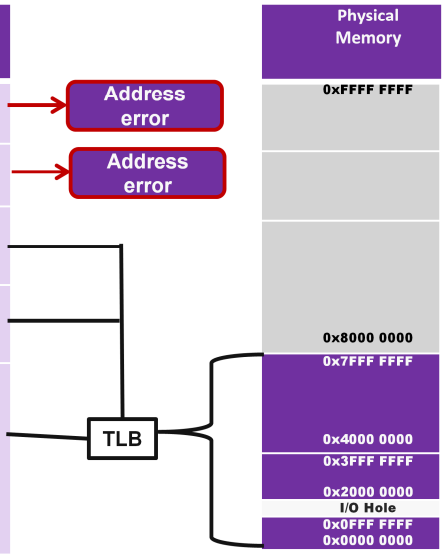
+ Segments CFG3 and CFG2 correspond to the old KSEG0 and KSEG1 and direct map to the lower 512 MB of physical address which encompasses the low 256MB of ram and the I0 Space.

+ Segments CFG5 and CFG4 correspond to the old KUSEG region are directly mapped to the lower 2GB of the physical address space encompassing the 2 RAM memory blocks and the I/O space.

+ Segments CFG1 and CFG0 correspond to the old KSEG2 and KSGE3 and are both mapped through the TLB.

EVA Memory Map For Linux on Malta FPGA - 2GB Ram User Mode

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	EU	C (cache coherency attribute)	VAR (Virtual Address Range)	LM (Legacy Mode)
SegCtl0 (5,4)	CFG 0	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3
	CFG1	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xDFFF FFFF 0XC000 0000	KSEG2
SegCtl1 (5,3)	CFG2	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	010 (uncached)	0xBFFF FFFF 0xA000 0000	KSEG1
	CFG3	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x9FFF FFFF 0x8000 0000	KSEG0
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x7FFF FFFF 0x4000 0000	KUSEG
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x3FFF FFFF 0x0000 0000	



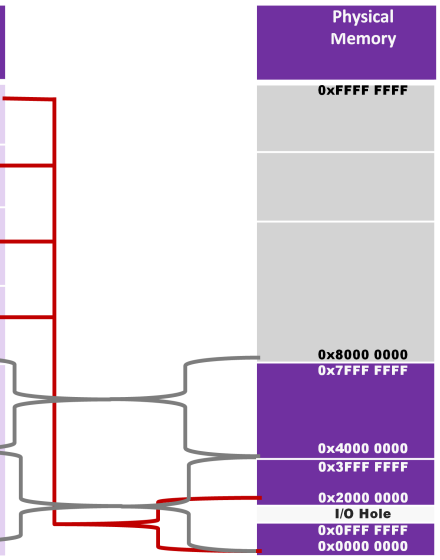
This is the memory map for the Linux Kernel running in user mode.

Segments CFG2, CFG3, CFG4 and CFG5 are all mapped through the TLB expanding the user virtual memory to 3GB.

+ Segments CFG0 and CFG1 are not accessible.

EVA Memory Map For Linux on Malta FPGA - 2GB Ram
EU - Status.ERL = 1 (reset, NMI or cache error)

CP0 Register	Segment	PA bits 31:29	AM (Access mode)	E U	C (cache coherency attribute)	VAR (Virtual Address Range)	LM (Legacy Mode)
SegCtl0 (5,4)	CFG 0	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xFFFF FFFF 0xE000 0000	KSEG3
	CFG1	000	MK 001 (Mapped Kernel)	1	na (TLB)	0xDFFF FFFF 0XC000 0000	KSEG2
SegCtl1 (5,3)	CFG2	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	010 (uncached)	0xBFFF FFFF 0xA000 0000	KSEG1
	CFG3	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x9FFF FFFF 0x8000 0000	KSEG0
SegCtl2 (5,2)	CFG4	010	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x7FFF FFFF 0x4000 0000	KUSEG
	CFG5	000	MUSUK 100 (Mapped User and Supervisor and Unmapped Kernel)	1	100 (Writeback, coherent, exclusive on write)	0x3FFF FFFF 0x0000 0000	



Here's the map for error conditions reset, NMI or cache errors.

+ Segments CFG0, CFG1, CFG2 and CFG3 correspond to the old KGES3, KSEG2, KSEG1 and KSEG0 respectively and all direct map to the lower 512 MB of physical address including the low 256MB of ram and the I0 Space.

+ Segment CFG5 is directly mapped to the lower 1GB of the physical address space which includes three quarters of a GB RAM divided into two parts with I/O space in-between.

+ Segment CFG4 is directly mapped to 1GB of physical ram stating at 0x4000 0000.

Adding the memory to Linux

- add_memory_region
  - Function call called in malta.init.c

```
Determined physical RAM map:
 memory: 00001000 @ 00000000 (reserved)
 memory: 000ef000 @ 00001000 (ROM data)
 memory: 00510000 @ 000f0000 (reserved)
 memory: 0fa00000 @ 00600000 (usable)
 memory: 60000000 @ 20000000 (usable)
```

- To check total available memory in the shell, we can use the free command to check how much memory is available to the system.

For the Linux kernel to use the extra memory the add_memory_region function is used to register the memory. See the arch/mips/mti-malta/malta-memory.c to see how this is done for the Malta Board.

Since the mappings and the cache attributes remain the same, existing device drivers and the root file system can be used without any modification.

Linux Files That Have Been Modify

File name	Description
arch/mips/kernel/cpu-probe.c	probe for interaptiv core
arch/mips/kernel/genex.S	add ftlb handler
arch/mips/kernel/r4k_switch.S	change _init_fpu
arch/mips/kernel/scall32-o32.S	setup stack argument in stackargs
arch/mips/kernel/segment.c	create proc entry for segment control
arch/mips/kernel/signal.c	add FPU context switch function call
arch/mips/kernel/smp-cmp.c	add segment check
arch/mips/kernel/spram.c	add interaAptiv spram support
arch/mips/kernel/traps.c	where ftlb handle implemented, add eva trap support
arch/mips/kernel/unaligned.c	emulate lhe, lwe, etc opcode
arch/mips/kernel/mips_ksyms.c	export strncpyxx, copy_from_user, etc
arch/mips/lib/memcpy-inatomic.S	add atomic copy from user, etc function
arch/mips/lib/memcpy.S	__copy_fromuser, __copy_touser, __copy_inuser,
arch/mips/lib/memset.S	add __bzero_user
arch/mips/lib/strlen_user.S	add __strlen_kernel_asm
arch/mips/lib/strncpy_user.S	add __strncpy_from_kernel_asm, __strncpy_from_kernel_nocheck_asm, __strncpy_from_user_asm
arch/mips/lib/strnlen_user.S	add __strnlen_kernel_asm,__strnlen_kernel_nocheck_asm,
arch/mips/mm/cache.c	modify __flush_cache_vmap, __flush_cache_vunmap, export mips_flush_data_cache_range
arch/mips/mm/c-r4k.c	adding D$ flush functions
arch/mips/mm/init.c	modify copy_to_user_page
arch/mips/mm/tlbex.c	add proAptiv support to tlbw function
arch/mips/mm/tlb-r4k.c	use tlbinv to invalidate tlb entry in local_flush_tlb_all()
arch/mips/mti-malta/malta-init.c	program PCI controlller to access 2GB memory
arch/mips/mti-malta/malta-memory.c	add prom_getevamdesc() to setup memory descriptor from bootloader to Linux Kernel
arch/mips/mti-malta/malta-pci.c	shift PCI devices to upper 2GB, to prevent PCI bridges loop
arch/mips/mti-malta/malta-setup.c	define new function plat_eva_setup() to setup segctl register

The changes to the Linux Kernel can be classified into the following;

Files marked in grey are for generic CPU detection, FPU initialization, and instruction emulation.

Files marked in purple contain functions that need to copy to user space or from kernel space. These functions have been changed to use the EVA Load/Store instructions. The reason for this is there is no KUSEG segment anymore and segmentation control register are not programmed to be Kernel mapped. In an old legacy core, when data was copied from address space in user mode to address space in kernel mode, lw/sw could be used because both USEG and KUSEG were mapped.

Files marked in green have been changed to use the new TLB instructions to invalidate TLB entries.

Files marked in white are needed for Malta to configure EVA and use the larger ram memory. This is platform specific. The key functions for these files setup the memory descriptor for Linux to register the appropriate memory that has been detected, register the memory with the add_memory_region function, and setup the segmentation control registers.

Linux Changes

- Use of CKSEG1ADDR for uncache memory access
  - Macro is in arch/mips/include/asm/addrspace.h
  - Depend on the file arch/mips/include/asm/mach-malta/spaces.h

Another thing that has been change in Linux kernel, for the Malta board, is the code that accesses the IO registers. The macro CKSEG1ADDR in addrspace.h has been changed depending on settings in the malta specific spaces.h include file. This macro should be used when a device driver needs access to an IO control register, via an uncached address. The macro supports a 3GB Ram space. It is needed because the kernel normally used the KSEG3 region as uncached for this purpose and when the core is programmed for 3GB of ram the corresponding CFG0 is configured for use as a DMA zone. It is recommended that you use a similar approach when you need to support 3GB of Ram.

Linux Segment Configuration

- Use CFG0 for SW IO Coherence when kernel is configured to support 3GB memory
  - - DMA_ZONE This segment is mapped to the lower area in the physical memory address unmapped and uncached. Device driver uses this zone to maintain software IO coherence.
  - For example, Ethernet driver will allocate the token ring buffer here. The Ethernet driver will use uncached access to these data structure to maintain SW IO coherence with the device.

When the kernel is configured to support 3GB of memory it will use the CFG0 segment as a DMA zone. This segment will be configured by the kernel as unmapped and uncached configured to the lower area in physical memory.

Linux Segment Configuration

- Use CFG2 for SW IO Coherence when kernel is configured less than 3 GB memory
  - - Linux kernel uses the segment the same as CFG0 except it's use when kernel is not configured to support 3GB memory.
- Use CFG1 for vmalloc
  - - To allocate large chuck of memory or loading large kernel module, it's more effective to allocate a large continuous memory to access the memory.
  - - This region needs to be configured as mapped. Whenever, CPU accesses this region, it will use TLB to access the memory.

The kernel will configure the CFG2 segment for I/O coherence when there is less than 3 GB of RAM memory the same as it configures the CFG0 segment when there is 3GB of memory or more .

In all cases the CFG1 segment is configured by the kernel as a mapped segment used for vmalloc and loading kernel modules.