SMP Linux Bring up on a MIPS32® Coherent Processing System

Document Number: MD00721
Revision 01.01
February 7, 2013

MIPS and MIPS’ affiliates do not assume any liability arising out of the application or use of this information, or of any error or omission in such information. Any warranties, whether express, statutory, implied or otherwise, including but not limited to the implied warranties of merchantability or fitness for a particular purpose, are excluded. Except as expressly provided in any written license agreement from MIPS or an authorized third party, the furnishing of this document does not give recipient any license to any intellectual property rights, including any patent rights, that cover the information in this document.

The information contained in this document shall not be exported, reexported, transferred, or released, directly or indirectly, in violation of the law of any country or international law, regulation, treaty, Executive Order, statute, amendments or supplements thereto. Should a conflict arise regarding the export, reexport, transfer, or release of the information contained in this document, the laws of the United States of America shall be the governing law.

The information contained in this document constitutes one or more of the following: commercial computer software, commercial computer software documentation or other commercial items. If the user of this information, or any related documentation of any kind, including related technical data or manuals, is an agency, department, or other entity of the United States government ("Government"), the use, duplication, reproduction, release, modification, disclosure, or transfer of this information, or any related documentation of any kind, is restricted in accordance with Federal Acquisition Regulation 12.212 for civilian agencies and Defense Federal Acquisition Regulation Supplement 227.7202 for military agencies. The use of this information by the Government is further restricted in accordance with the terms of the license agreement(s) and/or applicable contract terms and conditions covering this information from MIPS Technologies or an authorized third party.

1 Introduction

This application note describes how to boot up SMP Linux on a MIPS32® Coherent Processing System.

The application was developed using 1004K™ bit files executing on a Malta development platform.

- Linux kernel:
  http://www.linux-mips.org/pub/linux/mips/mti-stable/v2.6/linux-mti-2.6.35.9-2.tar.gz

- Yamon source:
  http://www.mips.com/secure-download/index.dot?product_name=/auth/yamon-src-02.21.tar.gz

- Bit files:
  A00205-1004Kc-1_3c_0-2i-64ID-64TLB-noL2_ITU-REF00693 fl
  A00206-1004Kc-1_3c_0-2i-64ID-64TLB-noL2_ITU-REF00693 fl

- Malta™ Development Board with 256 MB RAM

2 YAMON and Linux Boot Up Flows

2.1 YAMON Boot Up Flow

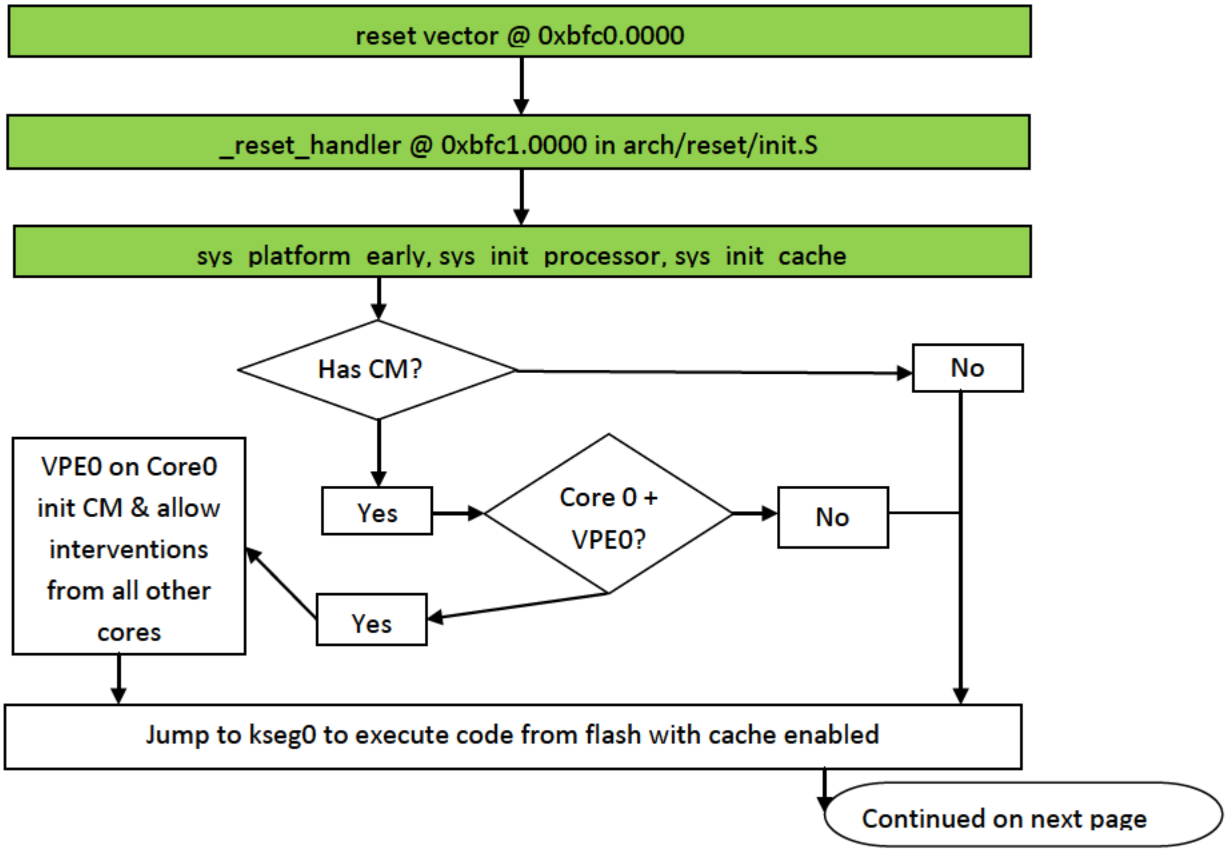
The Malta bit file is hardwired so that on cold reset, only Core0 in a multi-core cluster will power-up and begin executing code from the reset vector at address 0xbfc0.0000. Only VPE0 on Core0 will execute the code sequence in the flow chart and enter the shell. No other VPE/Core will execute YAMON code until VPE0 on Core0 powers-up the other cores in the `shell_cpu_init()` function.

YAMON is carefully coded such that only VPE0 on Core0 initializes the Coherence Manager (CM), Cluster Power Controller (CPC), and relocates code and data.

The *EBase.CPUNum* bit is used to identify which core is executing the code. In this case, VPE0 on Core0 has *EBase.CPUNum*=0. The CPU number is checked in the `sys_platform_early()` function. As a result, the value of *EBase.CPUNum* is stored in the V1 general-purpose register.

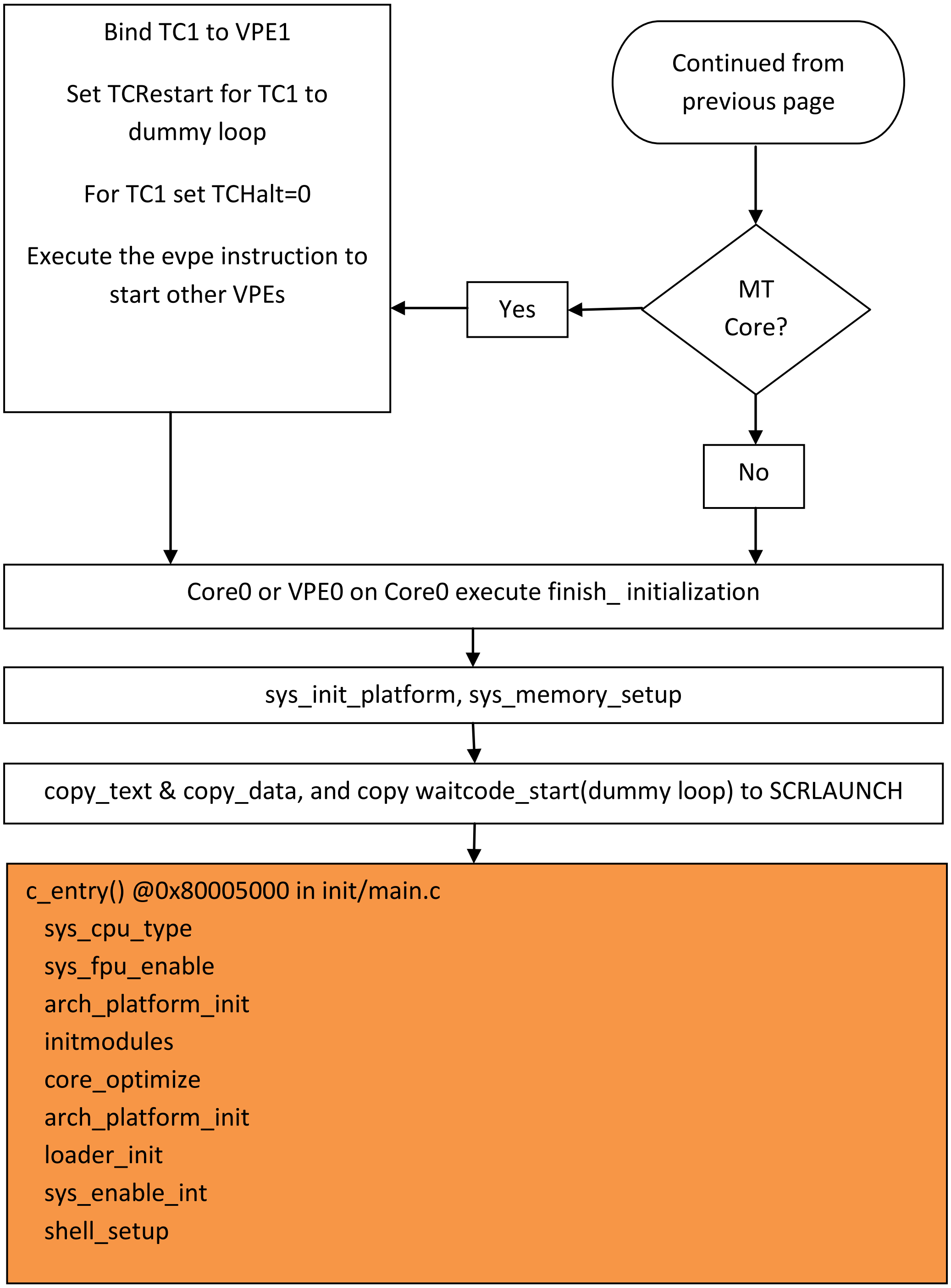
A flow chart of YAMON boot from the reset vector to the boot loader's shell is shown below. In the figure:

- A green background indicates that code is executed from flash with cache disabled.
- A white background indicates that code is executed from flash with cache enabled.
- An orange background indicates that code is executed from RAM with cache enabled.

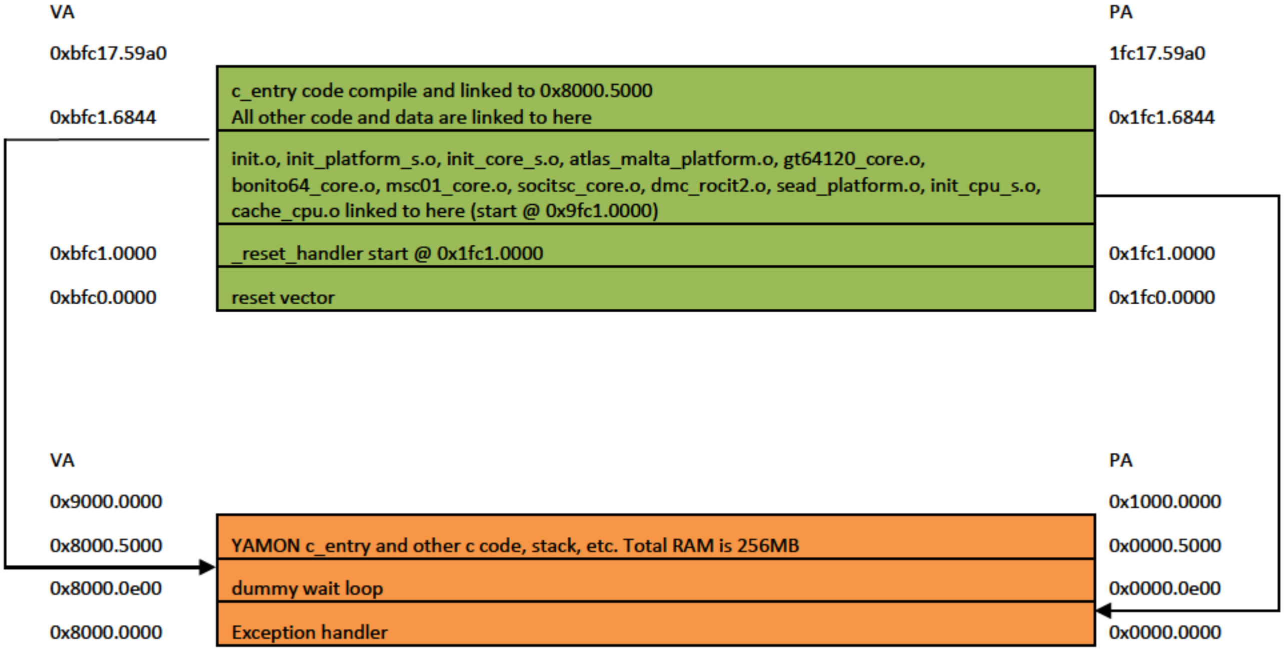

reset vector @ 0xbfc0.0000
_reset_handler @ 0xbfc1.0000 in arch/reset/init.S
sys_platform_early, sys_init_processor, sys_init_cache
Has CM?
No
Yes
Core 0 +
VPE0?
No
Yes
VPE0 on Core0 init CM & allow interventions from all other cores
Jump to kseg0 to execute code from flash with cache enabled
Continued on next page

After CM has been initialized, YAMON initializes multi-threading (MT) by binding TC1 to VPE1.

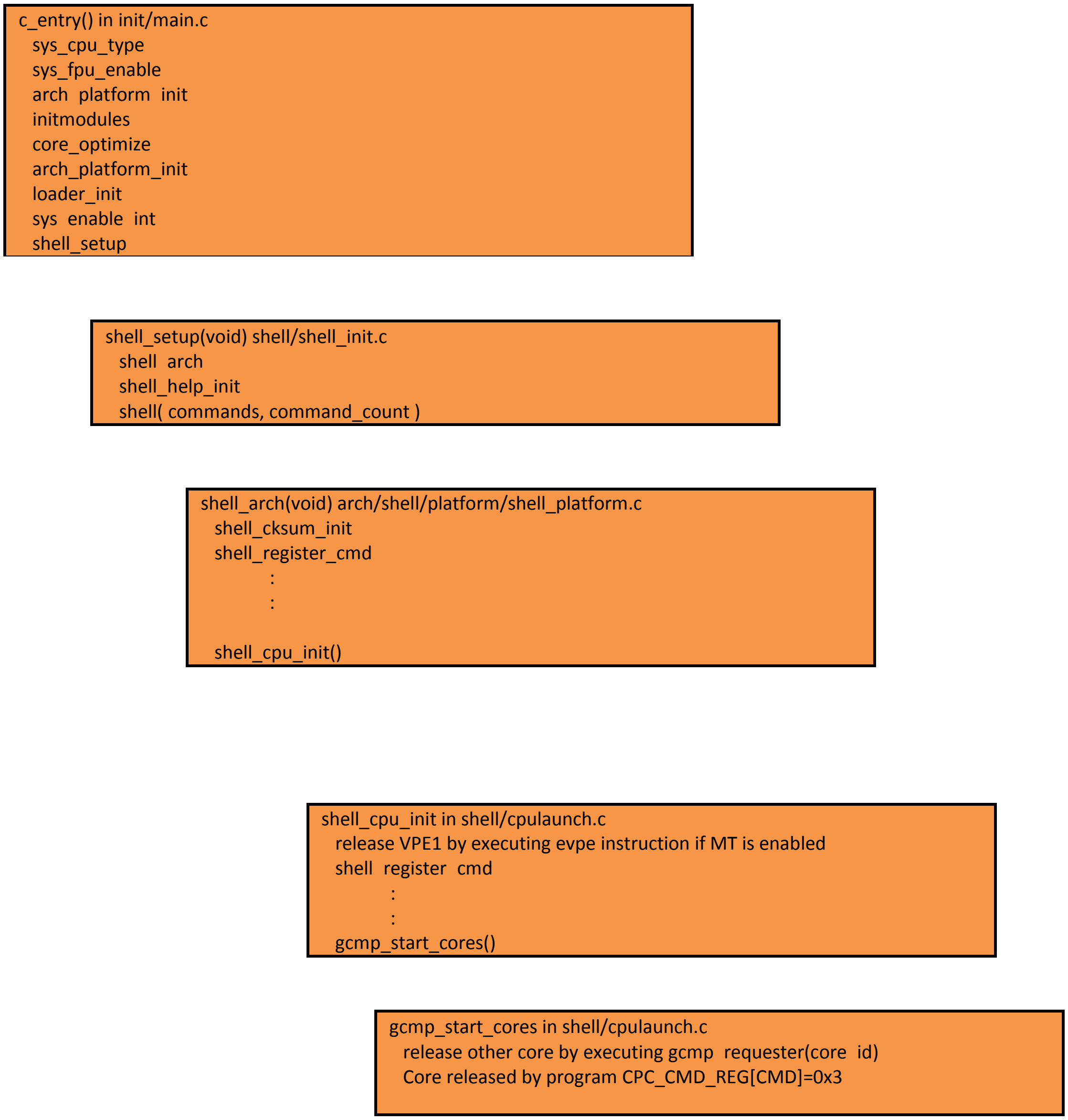
VPE1 on Core0 does not begin executing code after the Coherent Processing system has been initialized by VPE0; VPE1 will be released later in `shell_cpu_init`(), which is required because the RAM has not been initialized, and the code has not been copied to the RAM. After multi-threading has been initialized, VPE0/TC0 will copy code and data to the RAM. For a non-MT core such as the 1074K™, the MT initialization mechanism is bypassed.


Continued from previous page
MT Core?
Yes
No
Bind TC1 to VPE1
Set TCRestart for TC1 to dummy loop
For TC1 set TCHalt=0
Execute the evpe instruction to start other VPEs
Core0 or VPE0 on Core0 execute finish_ initialization
sys_init_platform, sys_memory_setup
copy_text & copy_data, and copy waitcode_start(dummy loop) to SCRLAUNCH
c_entry() @0x80005000 in init/main.c
sys_cpu_type
sys_fpu_enable
arch_platform_init
initmodules
core_optimize
arch_platform_init
loader_init
sys_enable_int
shell_setup

The high-level view of the memory map after the code relocation is shown below.


VA
PA
0xbfc17.59a0
1fc17.59a0
c_entry code compile and linked to 0x8000.5000
All other code and data are linked to here
0xbfc1.6844
0x1fc1.6844
init.o, init_platform_s.o, init_core_s.o, atlas_malta_platform.o, gt64120_core.o, bonito64_core.o, msc01_core.o, socitsc_core.o, dmc_rocit2.o, sead_platform.o, init_cpu_s.o, cache_cpu.o linked to here (start @ 0x9fc1.0000)
0xbfc1.0000
_reset_handler start @ 0x1fc1.0000
0x1fc1.0000
0xbfc0.0000
reset vector
0x1fc0.0000
VA
PA
0x9000.0000
0x1000.0000
0x8000.5000
YAMON c_entry and other c code, stack, etc. Total RAM is 256MB
0x0000.5000
0x8000.0e00
dummy wait loop
0x0000.0e00
0x8000.0000
Exception handler
0x0000.0000

After the code relocation, YAMON will continue performing system-level initialization. Before entering the shell, VPE0 on Core0 will release other cores in the `gcmp_start_cores()` function. The calling flow is shown below. To release a core, it is powered-up by setting *CPC_CMD_REG.CMD*=0x3.



```
c_entry() in init/main.c
  sys_cpu_type
  sys_fpu_enable
  arch  platform  init
  initmodules
  core_optimize
  arch_platform_init
  loader_init
  sys  enable  int
  shell_setup
```

```
shell_setup(void) shell/shell_init.c
  shell  arch
  shell_help_init
  shell( commands, command_count )
```

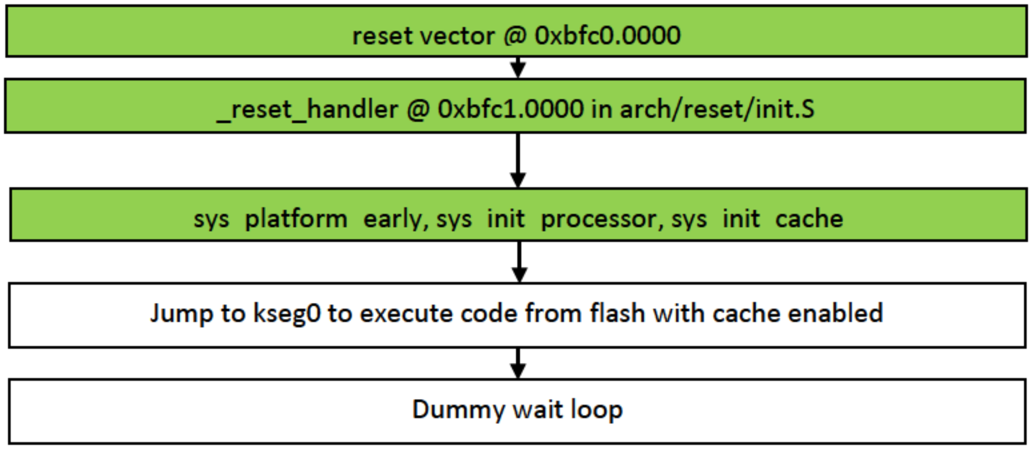
```
shell_arch(void) arch/shell/platform/shell_platform.c
  shell_cksum_init
  shell_register_cmd
          :
          :

  shell_cpu_init()
```

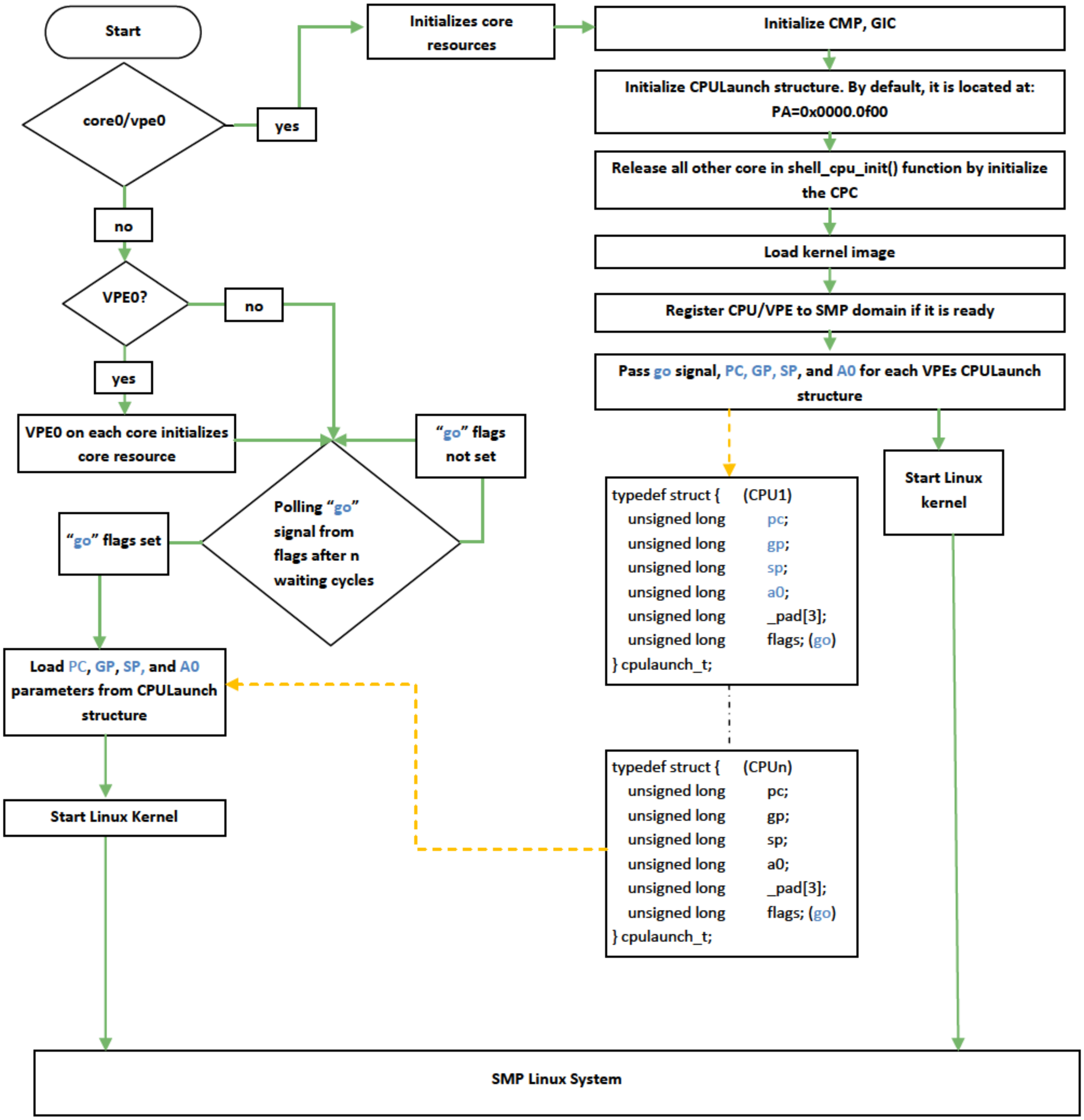
```
shell_cpu_init in shell/cpulaunch.c
  release VPE1 by executing evpe instruction if MT is enabled
  shell  register  cmd
          :
          :
  gcmp_start_cores()
```

```
gcmp_start_cores in shell/cpulaunch.c
  release other core by executing gcmp  requester(core  id)
  Core released by program CPC_CMD_REG[CMD]=0x3
```

When all other cores in the multi-core cluster have powered-up, they will begin fetching code from the reset vector. The orange arrow shows how other cores/VPEs (other than Core0) boot up and enter the dummy loop.


reset vector @ 0xbfc0.0000
_reset_handler @ 0xbfc1.0000 in arch/reset/init.S
sys platform early, sys init processor, sys init cache
Jump to kseg0 to execute code from flash with cache enabled
Dummy wait loop

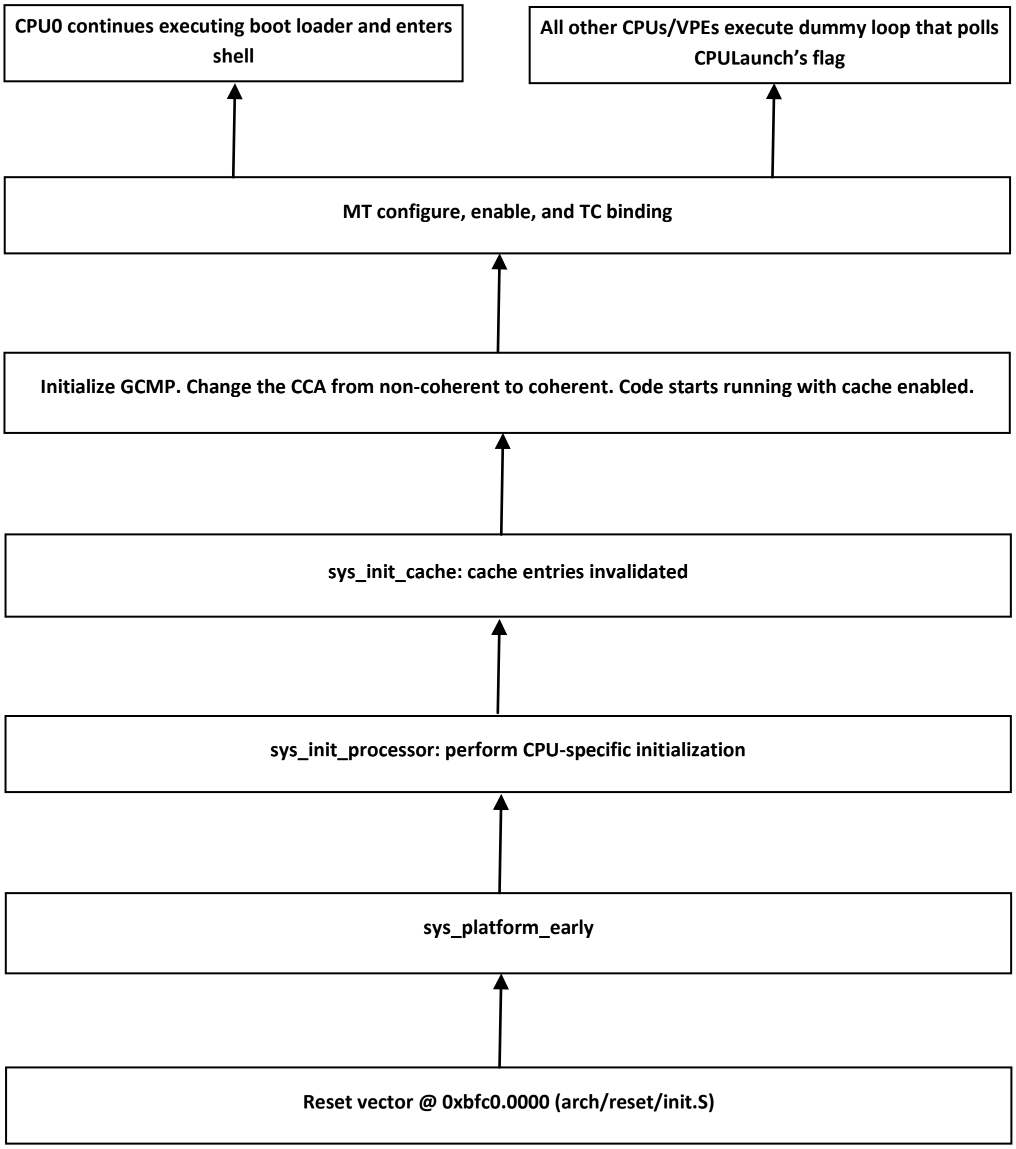
YAMON Bring up and Linux Handshake Flow Diagram


Start
core0/vpe0
yes
no
Initializes core resources
Initialize CMP, GIC
Initialize CPULaunch structure. By default, it is located at: PA=0x0000.0f00
Release all other core in shell_cpu_init() function by initialize the CPC
Load kernel image
Register CPU/VPE to SMP domain if it is ready
Pass go signal, PC, GP, SP, and A0 for each VPEs CPULaunch structure
VPE0?
no
yes
VPE0 on each core initializes core resource
"go" flags not set
Polling "go" signal from flags after n waiting cycles
"go" flags set
Load PC, GP, SP, and A0 parameters from CPULaunch structure
Start Linux Kernel
Start Linux kernel
SMP Linux System

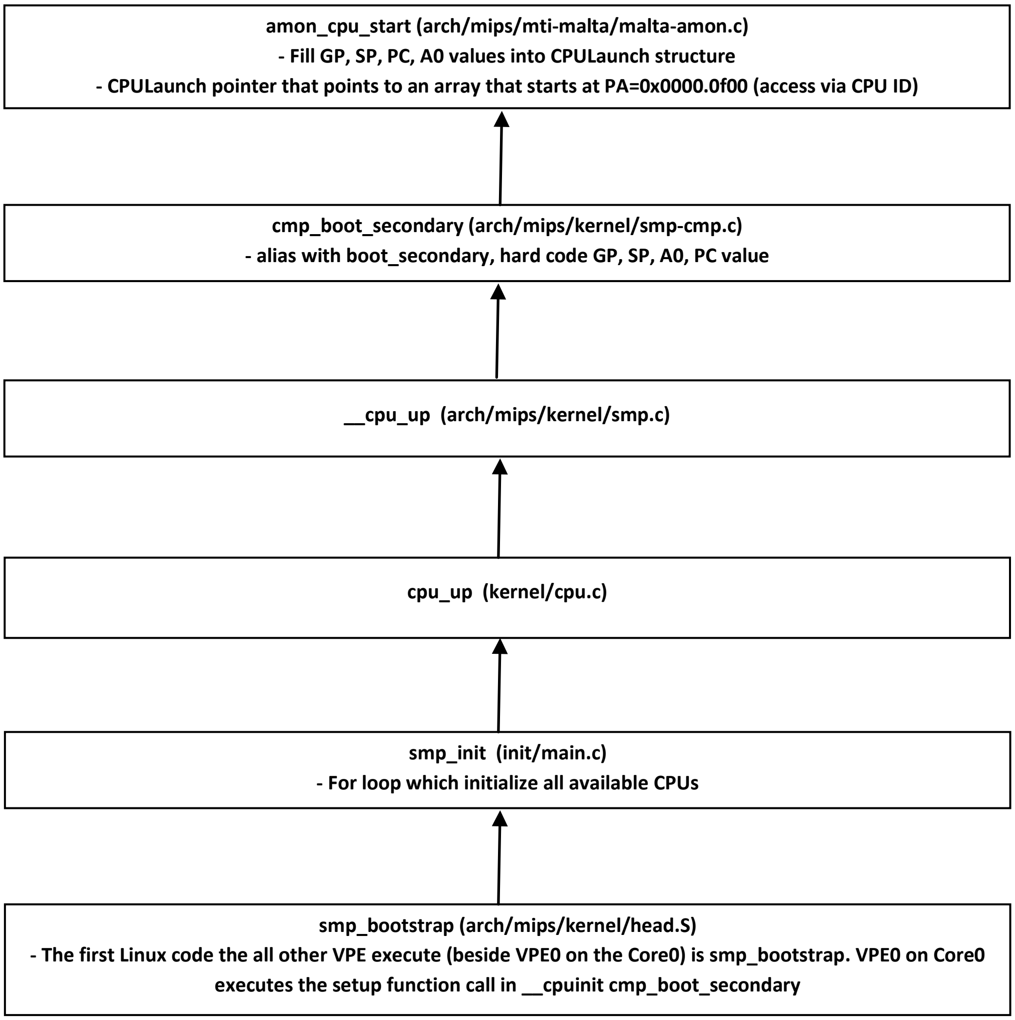
```
typedef struct {    (CPU1)
  unsigned long       pc;
  unsigned long       gp;
  unsigned long       sp;
  unsigned long       a0;
  unsigned long       _pad[3];
  unsigned long       flags; (go)
} cpulaunch_t;
```

```
typedef struct {    (CPUn)
  unsigned long       pc;
  unsigned long       gp;
  unsigned long       sp;
  unsigned long       a0;
  unsigned long       _pad[3];
  unsigned long       flags; (go)
} cpulaunch_t;
```

2.2 YAMON High-level Set-up Flow for SMVP Linux Boot Up


CPU0 continues executing boot loader and enters shell
All other CPUs/VPEs execute dummy loop that polls CPULaunch’s flag
MT configure, enable, and TC binding
Initialize GCMP. Change the CCA from non-coherent to coherent. Code starts running with cache enabled.
sys_init_cache: cache entries invalidated
sys_init_processor: perform CPU-specific initialization
sys_platform_early
Reset vector @ 0xbfc0.0000 (arch/reset/init.S)

2.3 Linux High-level Set-up Flow for SMVP Linux Boot Up



amon_cpu_start (arch/mips/mti-malta/malta-amon.c)
- Fill GP, SP, PC, A0 values into CPULaunch structure
- CPULaunch pointer that points to an array that starts at PA=0x0000.0f00 (access via CPU ID)

cmp_boot_secondary (arch/mips/kernel/smp-cmp.c)
- alias with boot_secondary, hard code GP, SP, A0, PC value

__cpu_up (arch/mips/kernel/smp.c)

cpu_up (kernel/cpu.c)

smp_init (init/main.c)
- For loop which initialize all available CPUs

smp_bootstrap (arch/mips/kernel/head.S)
- The first Linux code the all other VPE execute (beside VPE0 on the Core0) is smp_bootstrap. VPE0 on Core0 executes the setup function call in __cpuinit cmp_boot_secondary

2.4 Details of Calling Flow in SMP Linux

Below is the detailed SMP Linux calling flow. The orange background color indicates that the functions are executed by VPE0 on Core0 (or Core0 for non-MT cores). The blue background indicates that the call is executed only by other VPEs/CPUs.

Function name & calling flow						File where function is defined	Console output
__kernel_entry ()						./arch/mips/kernel/head.S	
	init start kernel ();					./init/main.c	
		smp_setup_processor_id()					
		lockdep_init();					
		debug_objects_early_init();					
		boot_init_stack_canary();					
		cgroup_init_early();					
		local irq disable();					
		early_boot_irqs_off();					
		early_init_irq_lock_class();					
		lock_kernel();					
		tick_init();					
		boot_cpu_init();					
			set cpu online(cpu, true);				
			set_cpu_active(cpu, true);				
			set_cpu_present(cpu, true);				
			set_cpu_possible(cpu, true);				
		page address init();					
		setup_arch(&command_line);				arch/mips/kernel/setup.c	
			cpu probe();			arch/mips/kernel/cpu-probe.c	
				cpu_probe_mips(c, cpu);		arch/mips/kernel/cpu-probe.c	
					decode_configs();	arch/mips/kernel/cpu-probe.c	
					spram_config()		
				cpu_probe_vmbits(c);			
			prom_init			arch/mips/mti-malta/malta-init.c	Display "Linux" on LED and "LINUX started..." to console"
				prom_init_cmdline();		arch/mips/mti-malta/malta-cmdline.c	
				prom_meminit();		arch/mips/mti-malta/malta-memory.c	
				console_config();		arch/mips/mti-malta/malta-init.c	
				gcmp_probe(GCMP_BASE_ADDR, GCMP_ADDRSPACE_SZ);			
				register_smp_ops(&cmp_smp_ops); or register_smp_ops(&vsmp_smp_ops);			

Function name & calling flow						File where function is defined	Console output
			setup_early_printk();			arch/mips/kernel/early_printk.c	
				register_console(&early_console);		./kernel/printk.c	Display "bootconsole [early0] enabled"
			cpu_report();			arch/mips/kernel/cpu-probe.c	Display ""CPU revision is:"
			check_bugs_early();				
			arch_mem_init(cmdline_p);			arch/mips/kernel/setup.c	Print Memory map "Determined physical RAM map"
				plat_mem_setup();			
				print_memory_map();			Print Memory map " memory: 00001000 @ 00000000 (reserved)"
				parse_early_param();			
				bootmem_init();		arch/mips/kernel/setup.c	Display "Wasting 44064 bytes for tracking 1377 unused pages"
					init_initrd();		
					init_bootmem_node();		
					reserve_bootmem(PFN_PHYS(mapstart), bootmap_size, BOOTMEM_DEFAULT);		
					finalize_initrd();		Display "Initrd not found or empty - disabling initrd"
				sparse_init();			
				paging_init();		./arch/mips/mm/init.c	
					pagetable_init();		
					kmap_init();		
					kmap_coherent_init();		
					free_area_init_nodes	./mm/page_alloc.c	Display "Zone PFN ranges:" "Movable zone start PFN for each node" "early_node_map[1] active PFN ranges"
			resource_init();				
			plat_smp_setup();			arch/mips/include/asm/smp-ops.h	
				cmp_smp_setup();		arch/mips/kernel/smp-cmp.c	Display "Detected 5 available secondary CPU(s)"
		mm_init_owner(&init_mm, &init_task);					
		setup_command_line(command_line);					
		setup_nr_cpu_ids();					
		setup_per_cpu_areas();				./mm/percpu.c	
			pcpu_embed_first_chunk			./mm/percpu.c	Display "PERCPU: Embedded 7 pages/cpu @81203000 s5888 r8192 d14592 u65536"
				pcpu_setup_first_chunk		./mm/percpu.c	
					pcpu_dump_alloc_info	./mm/percpu.c	Display "pcpu-alloc: s5888 r8192 d14592 u65536 alloc=16*4096" "pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3 [0] 4 [0] 5"
				pcpu_free_alloc_info		./mm/percpu.c	
				free_bootmem		./mm/bootmem.c	
		smp_prepare_boot_cpu();				./arch/mips/kernel/smp.c	

Function name & calling flow							File where function is defined	Console output
			set cpu possible(0, true);					
			set_cpu_online(0, true);					
			cpu set(0, cpu callin map);					
		build_all_zonelists();					./mm/page_alloc.c	
			build_zonelists(pgdat);				./mm/page_alloc.c	Display "Built 1 zonelists in Zone order, mobility grouping on. Total pages: 65024"
			build zonelist cache(pgdat);					
		page_alloc_init();						
		printk(KERN_NOTICE "Kernel command line: %s\n", boot_command_line);						Display "Kernel command line: init=/init ip=dhcp console=ttyS0,38400n8r"
		parse early param();						
		parse_args("Booting kernel", static_command_line, __start___param, __stop___param - start param, &unknown bootoption);						
		pidhash_init();					./kernel/pid.c	
			alloc large system hash				./mm/page alloc.c	Display "PID hash table entries: 1024 (order: 0, 4096 bytes)"
		vfs_caches_init_early();					./fs/dcache.c	
			dcache init early();				./fs/dcache.c	
				alloc large system hash			./mm/page alloc.c	Display "Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)"
				INIT_HLIST_HEAD				
			inode init early();				./fs/inode.c	
				alloc large system hash			./mm/page alloc.c	Display "Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)"
				INIT_HLIST_HEAD				
		sort main extable();					./kernel/extable.c	
			sort_extable();				./lib/extable.c	
		trap init();					arch/mips/kernel/traps.c	
			per_cpu_trap_init();				arch/mips/kernel/traps.c	
				change c0 status				
				cpu_cache_init();			arch/mips/mm/cache.c	
					r4k cache init();		arch/mips/mm/c-r4k.c	
						probe_pcache();		Display "Primary instruction cache 32kB, 4-way, VIPT, linesize 32 bytes." "Primary data cache 32kB, 4-way, PIPT, no aliases, linesize 32 bytes"
						setup_scache();		Display "MIPS secondary cache 512kB, 8-way, linesize 64 bytes."
						r4k_blast_dcache_page_setup();		
						r4k blast dcache page indexed setup();		
						r4k_blast_dcache_setup();		

Function name & calling flow							File where function is defined	Console output
						r4k blast icache page setup();		
						r4k_blast_icache_page_indexed_setup();		
						r4k blast icache setup();		
						r4k_blast_scache_page_setup();		
						r4k blast scache page indexed setup();		
						r4k_blast_scache_setup();		
						build clear page();		
						build_copy_page();		
						local r4k flush cache all(NULL);		
						coherency_setup();		
				tlb init();				
			set_handler(0x180, &except_vec3_generic, 0x80);					
			set except vector(i, handle reserved);					
			board_ejtag_handler_setup();					
			set except vector(23, handle watch);					
			set_vi_handler(i, NULL);					
			parity_protection_init();				arch/mips/kernel/traps.c	Display "Writing ErrCtl register=00000000" "Readback ErrCtl register=00000000"
			board be init();					
			board_nmi_handler_setup();					
			signal init();/signal32 init();					
			local_flush_icache_range(ebase, ebase + 0x400);					
			flush tlb handlers();					
		mm_init();					./init/main.c	
			page cgroup init flatmem();					
			mem init();				arch/mips/mm/init.c	Display "Memory: 252512k/255328k available (3412k kernel code, 2452k reserved, 851k data, 1376k init, 0k highmem)"
				calculate totalram_pages, num_physpages, totalhigh_pagescodesize, datasize, initsize				
			kmem_cache_init();					
			pgtable cache init();					
			vmalloc_init();					
		sched init();						
		preempt_disable();						
		rcu init();					./kernel/rcupdate.c	
			__rcu_init				./kernel/rcutree.c	
				rcu bootup announce			./kernel/rcutree plugin.h	Display "Hierarchical RCU implementation."
				__rcu_init_preempt				

Function name & calling flow					File where function is defined	Console output
				open softirq(RCU SOFTIRQ, rcu process callbacks);		
			cpu_notifier(rcu_barrier_cpu_hotplug, 0);			
		early irq init();				Display "NR IRQS:256"
		init_IRQ();			arch/mips/kernel/irq.c	
			set irq noprobe(i)			
			arch init irq();		arch/mips/mti-malta/malta-int.c	Display "CPU0: status register was 11002400" "CPU0: status register now 11002400" "CPU0: status register frc 11003c00"
				init_i8259_irqs		
				mips cpu irq init();	arch/mips/kernel/irq cpu.c	
				init_msc_irqs		
				set vi handler(MIPSCPU INT I8259A, malta hw0 irqdispatch);		
				set_vi_handler(MIPSCPU_INT_COREHI, corehi_irqdispatch);		
				setup irq(MIPS CPU IRQ BASE+MIPSCPU INT I8259A, &i8259irq);		
				setup_irq(MIPS_CPU_IRQ_BASE+MIPSCPU_INT_COREHI,&corehi_irqaction);		
				fill ipi map();		
				gic_init(GIC_BASE_ADDR, GIC_ADDRSPACE_SZ, gic_intr_map, ARRAY SIZE(gic intr map), MIPS GIC IRQ BASE);		
		prio_tree_init();				
		init timers();				
		hrtimers_init();				
		softirq init();				
		timekeeping_init();				
		time init();			./arch/mips/kernel/time.c	Display "CPU frequency 24.99 MHz"
			plat time init();		./arch/mips/mti-malta/malta-time.c	
				estimate_cpu_frequency();		
				mips scroll message();		
				plat_perf_setup();		
			mips clockevent init()			
			init_mips_clocksource()			
		profile init();				
		early_boot_irqs_on();				
		local irq enable();				
		set_gfp_allowed_mask(__GFP_BITS_MASK);				
		kmem cache init late();				
		console_init();			drivers/char/tty_io.c	
			con initcall start			
				con_init()	drivers/char/vt.c	Display "Console: colour dummy device 80x25"

Function name & calling flow				File where function is defined	Console output
		lockdep info();			
		locking_selftest();			
		page cgroup init();			
		enable_debug_pagealloc();			
		kmemtrace init();			
		kmemleak_init();			
		debug objects mem init();			
		idr_init_cache();			
		setup per cpu pageset();			
		numa_policy_init();			
		if (late time init) late time init();			
		sched_clock_init();			
		calibrate_delay();		./init/calibrate.c	Display "Calibrating delay loop... 200.25 BogoMIPS (lpj=2093056)"
		pidmap init();			
		anon_vma_init();			
		thread info cache init();			
		cred_init();			
		fork init(totalram pages);			
		proc_caches_init();			
		buffer init();			
		key_init();			
		security init();			
		vfs_caches_init(totalram_pages);			
			dcache init();		
			inode_init();		
			files init(mempages);		
			mnt_init();	./fs/namespace.c	Display "Mount-cache hash table entries: 512"
			bdev cache init();		
			chrdev_init();		
		radix tree init();			
		signals_init();			
		page writeback init();			
		cgroup_init();			
		cpuset init();			
		taskstats_init_early();			
		delayacct init();			

Function name & calling flow	File where function is defined	Console output
check bugs();		
acpi_early_init();		
sfi init late();		
ftrace_init();		
rest init();	./init/main.c	
rcu_scheduler_starting();		
kernel thread(kernel init, NULL, CLONE FS | CLONE SIGHAND);		
kernel_init	./init/main.c	
smp prepare cpus(setup max cpus);		
cmp_prepare_cpus	arch/mips/kernel/smp-cmp.c	
mips mt set cpuoptions		
do_pre_smp_initcalls();		
initcall start		
start_boot_trace();		
smp init();	./init/main.c	
cpu_up(cpu);	./kernel/cpu.c	
cpu maps update begin();		
_cpu_up(cpu, 0);	./kernel/cpu.c	
cpu hotplug begin();		
raw notifier call chain(&cpu chain, CPU UP PREPARE | mod, hcpu,-1, &nr calls);		
__cpu_up(cpu);	arch/mips/kernel/smp.c	
fork idle(cpu);		
boot_secondary(cpu, idle);	arch/mips/kernel/smp-cmp.c	
cmp boot secondary(cpu, idle);	arch/mips/kernel/smp-cmp.c	
amon_cpu_start(cpu, pc, sp, (unsigned long)gp, a0);	arch/mips/mti-malta/malta-amon.c	
update struct cpulaunch, each core start execute smp bootstrap()		
smp_bootstrap()	arch/mips/kernel/head.S	
mips ihb	arch/mips/kernel/entry.S	
setup_c0_status_sec	arch/mips/kernel/head.S	
smp slave setup	arch/mips/kernel/head.S	
start_secondary	arch/mips/kernel/smp.c	
cpu probe()	arch/mips/kernel/cpu-probe.c	
cpu_report();		
per cpu trap init();		
mips_clockevent_init();		

Function name & calling flow															File where function is defined	Console output
														mp ops->init secondary();		
														calibrate_delay();		
														notify cpu starting(cpu);		
														mp_ops->smp_finish();		
														set cpu sibling map(cpu);		
														cpu_set(cpu, cpu callin map);		
														synchronise_count_slave();	arch/mips/kernel/sync-r4k.c	
														cpu idle();		
											cpu_isset(cpu, cpu_callin_map)					
											udelay(100);					
											cpu_set(cpu, cpu_online_map);					
										set cpu active(cpu, true);						
										raw_notifier_call_chain(&cpu_chain, CPU_ONLINE | mod, hcpu);						
										cpu_hotplug_done();						
									cpu maps update done();							
								printk(KERN_INFO "Brought up %ld CPUs\n", (long)num online cpus());								Display "Brought up 6 CPUs"
								smp_cpus_done(setup_max_cpus);							arch/mips/kernel/smp.c	
									mp ops->cpus done();						arch/mips/kernel/smp-cmp.c	
										cmp_cpus_done					arch/mips/kernel/smp-cmp.c	
									synchronise count master();						arch/mips/kernel/sync-r4k.c	
							sched_init_smp();									
							do basic setup();									
				numa_default_policy();												
				kernel thread(kthreadd, NULL, CLONE FS | CLONE FILES);												
				find_task_by_pid_ns(pid, &init_pid_ns);												
				unlock kernel();												
				init_idle_bootup_task(current);												
				preempt enable no resched();												
				schedule();												
				preempt disable();												
				cpu_idle();												

NOTES:

1. The function `asmlinkage __cpuinit void start_secondary(void)` in `arch/mips/kernel/smp.c` is the first C code executed by all secondary CPUs.
2. The first function executed by the primary CPU is `__kernel_entry ()`. The first function executed by the secondary CPU is `smp_bootstrap()`.
3. The functions `synchronise_count_master()` and `synchronise_count_slave()` are used to synchronize the clock source among primary and secondary CPUs. Two possible clock sources can be used: the traditional Count/Compare from each core or the global counter in GIC.

3 Q & A

3.1 How does the code determine which CPU is executing?

The code can check the *CPUNum* field in the CP0 *EBase* register. For a two-core/four-VPE configuration, VPE0 in Core0 has a value of 0. The other values are:

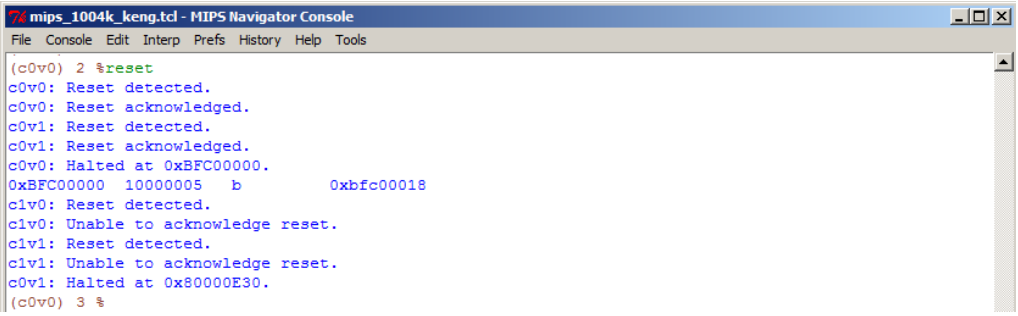
VPE1 on Core0 has value of 1
VPE0 on Core1 has value of 2
VPE1 on Core1 has value of 3

On two-core two-VPE configurations:

VPE0 on Core0 has value of 0
VPE0 on Core1 has value of 1

3.2 Why is Core1 unable to acknowledge reset?

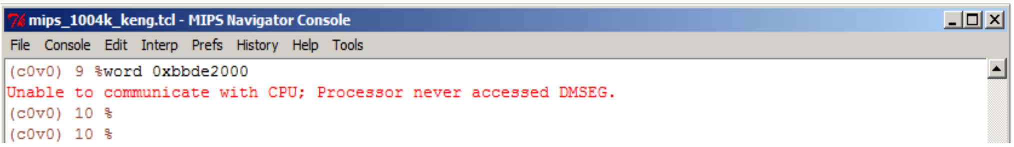
When using the Navigator Console to reset the target (EJTAGBOOT), the console will display "Unable to acknowledge reset". This indicates that the CPC did not release Core1 in order that both VPE0 and VPE1 on Core1 cannot respond to the reset request from JTAG until the code issues a power-up command.



```
(c0v0) 2 %reset
c0v0: Reset detected.
c0v0: Reset acknowledged.
c0v1: Reset detected.
c0v1: Reset acknowledged.
c0v0: Halted at 0xBFC00000.
0xBFC00000  10000005   b         0xbfc00018
c1v0: Reset detected.
c1v0: Unable to acknowledge reset.
c1v1: Reset detected.
c1v1: Unable to acknowledge reset.
c0v1: Halted at 0x80000E30.
(c0v0) 3 %
```

3.3 What does the message "Unable to communicate with CPU; Processor never accessed DMSEG" mean?

When using the System Navigator probe to read/write the CPC register, and the error "Unable to communicate with CPU; Processor never accessed DMSEG" appears on the console, most likely the CPC is not enabled. Set *GCR_CPC_BASE.CPC_EN* to 0x1 to enable the CPC unit.



```
(c0v0) 9 %word 0xbbde2000
Unable to communicate with CPU; Processor never accessed DMSEG.
(c0v0) 10 %
(c0v0) 10 %
```

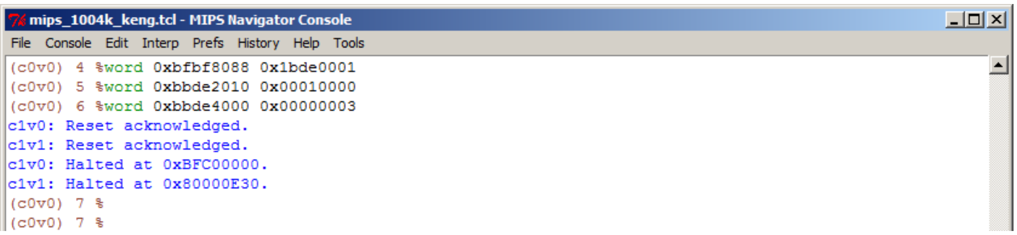
3.4 What is the high-level view of the GCR, CPC, and GIC address mapping used on the Malta board?

Below is the address mapping used by YAMON on the Malta board for the GCR, CPC, and GIC. The GCR is 64KB from the CPC, and the CPC is mapped next to the GIC.

	Start Addr	**Size**	**Name**	**Note**
Global Control Registers	0x1fbf.e000	8K	Global Debug Block	By default based address is hardwired to 0x1fbf.8000. Can be reprogramed by SW via GCR Base Register[GCR_BASE].
	0x1fbf.c000	8K	Core-Other Control Block	
	0x1fbf.a000	8K	Core-Local Control Block	
	0x1fbf.8000	8K	Global Control Block	
	0x1bde.8000	64K	Unused	
Cluster Power Controller	0x1bde.6000	8K	Unused	Based address is set by Cluster Power Controller Base Address Register[CPC_BaseAddress] bits.
	0x1bde.4000	8K	Core-Other Control Block	
	0x1bde.2000	8K	Core-Local Control Block	
	0x1bde.0000	8K	Global Control Block	
Global Interrupt Controller	0x1bdd.0000	64K	User-Mode Visible Section	Based address is set by Global Interrupt Controller Base Address Register [GIC_BaseAddress] bits.
	0x1bdc.c000	16K	VPE-Other Section	
	0x1bdc.8000	16K	VPE-Local Section	
	0x1bdc.0000	32K	Shared Section	

3.5 How is Core1 released from the Navigator Console?

To release Core1 from the System Navigator probe, enter the commands shown below in the Navigator Console.


mips_1004k_keng.tcl - MIPS Navigator Console
File Console Edit Interp Prefs History Help Tools

```
(c0v0) 4 %word 0xbfbf8088 0x1bde0001
(c0v0) 5 %word 0xbbde2010 0x00010000
(c0v0) 6 %word 0xbbde4000 0x00000003
c1v0: Reset acknowledged.
c1v1: Reset acknowledged.
c1v0: Halted at 0xBFC00000.
c1v1: Halted at 0x80000E30.
(c0v0) 7 %
(c0v0) 7 %
```

The first command, `word 0xbfbf8088 0x1bde0001`, writes 0x1bde0001 to 0xbfbf8088 in KSEG1, which is the address of the *Cluster Power Controller Base Address Register* (*GCR_CPC_BASE*). The value 0x1bde0001 is used to set the physical address of the CPC to 0x1bde 0000 and enable the CPC:

```
GCR_CPC_BASE[CPC_BaseAddress] = 0x1bde0
GCR_CPC_BASE [CPC_EN] = 0x1
```

The second command, `word 0xbbde2010 0x00010000`, writes 0x00010000 to the KSEG1 address 0xbbde 2010, which is the address of *CPC_OTHER_REG*. The value of 0x00010000 is used to set *CORENUM* to 0x1in that register.

```
CPC_OTHER_REG[CORENUM]= 0x1
```

This programs the CPU IDs of the other cores so that the *Core Other Addressing Register* can be used to power-up that core.

The third command powers up Core 1. The command `word 0xbbde4000 0x00000003` writes 0x0000 0003 to the KSEG1 address 0xbbde 4000, which is the address of the Other Cores Cluster Power Controller's command register. The value of 0x00000003 writes the power-up command to that register and powers up the core.

```
CPC_CMD_REG[CMD]=0x3
```

3.6 How does YAMON bind TCs to VPEs on a Coherent Processing System core?

By default, YAMON binds TC0 to VPE0 and binds all other TCs to VPE1 on each core. Only TC1 on VPE1 runs the dummy polling loop. All other TCs remain inactive.