PPC Multicore example with Cosmic Tools:

how to quickly run three cores with no libraries

Multicore systems are certainly harder to develop, test and debug than single-core systems, but sometimes, with the right tools and knowledge, it's not as hard as you would believe: this Application Note describes a very simple example that, using no other resources than a header file, starts and runs the three cores of the NXP MPC5748G chip: the example runs on a MPC574XG-MB board mounted with a MPC574XG-324DS daugther board.

MPC574XG-MB Motherboard and MPC574XG-324DS Daughther Card

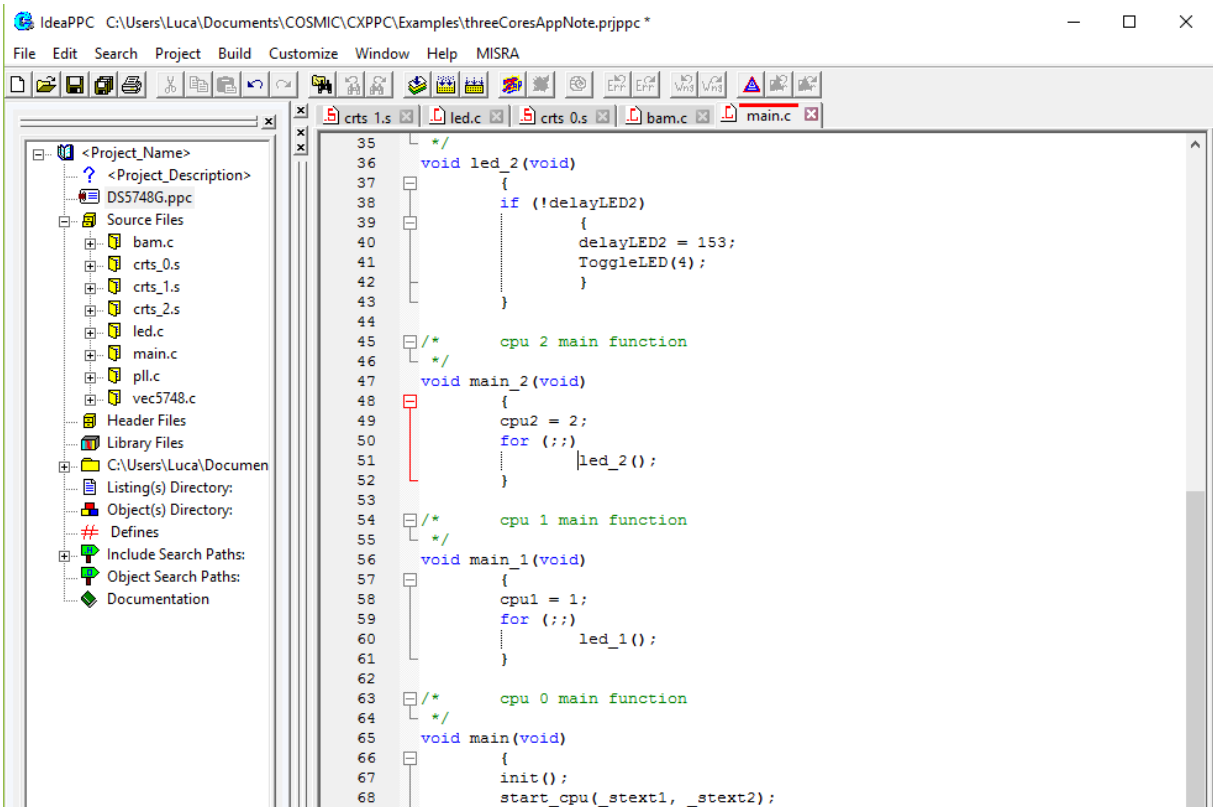
The idea behind this example is that every core shows that it is running by toggling a led that is specific to that core, at a specific frequency: in this way it is easy to visually check on the target board that all 3 cores are running (when the three leds are blinking, each at its expected frequency) or if one of the cores did not start for any reason.

The code is organised as follows:

- the main core is started and does some initilizations (PLL, interrupt system..)
- once the hardware is initialized, the main core starts the 2 other cores, by telling them the start address of the code they should execute and then setting a bit that actually starts the core.
- the three cores share some subroutines (the led toggling function for example) and variables (the counters used to vary the toggling frquency of each led)

– a timer interrupt is managed by the main core and provides a time base for the whole system

The picture below shows the project structure within IDEA:



The content of the files is:

- crts_0.s : assembler file containing the initialization code for the main core. This file is part of the standard compiler package.
- crts_1.s and crts_2.s : these files are derived from the file above, but they are simpler: they contain a few assembler lines that initialise the cores 1 and 2 with the specific information they need (private stack address, dedicated main function..)
- bam.c : a constant structure containing some configuration constants for the whole chip (this is common to all Power Architecture chips)
- pll.c : the system clock initialization routine.
- vec5748.c : the vector table. Only one interrupt is used in this simple example (called it_pit1()), so the table is largely empty, but it's always useful to have the whole table

so that if you add another interrupt you just need to fill in the corresponding line with the name of the interrupt function.

- main.c: contains the main function (for each core). This file is explained in detail later.
- led.c : contains the led management, the timer interrupt function and the code that starts the cores 1 and 2 (the main core is called core 0)

Let's see in more details what the code does by looking at the content of the file main.c

```
/*      EXAMPLE PROGRAM FOR MPC5748G
 *      Copyright (c) 2016 by COSMIC Software
 */
void ToggleLED(int);
void start_cpu(void (*)(void), void(*)(void));
void _stext1(void);
void _stext2(void);

volatile int delayLED0, delayLED1, delayLED2;
int cpu1, cpu2;

/*      cpu 0 LED toggling
 */
void led_0(void)
        {
        if (!delayLED0)
                {
                delayLED0 = 456;
                ToggleLED(1);
                }
        }

/*      cpu 1 main function
 */
void main_1(void)
        {
        cpu1 = 1;
        for (;;)
                led_1();
        }

/*      cpu 0 main function
 */
void main(void)
        {
        init();
        start_cpu(_stext1, _stext2);
        for (;;)
                led_0();
        }
```

As you can see, it's pretty simple:

- the main core does some initialisations (explained later), then starts the other cores and the manges its own led (called led_0)
- the core 1 executes the function main_1 (shown) that just blink its led in an infinite loop (the core 2, not shown, does exactly the same thing with its own led)
- The led management routine (the one for the core 0 is shown, the 2 other are similar) wait for their specific counter to reach zero (the counters are decremented in the interrupt routine managed by the core0) then toggle the led and reload the counter.

Let's now see the content of led.c, that is the other important file of this example:

```c
/*	start cpu 1 and cpu 2
 */
void start_cpu(void (*pcpu1)(void), void (*pcpu2)(void))
	{
	if (pcpu1)
		{
		MC_ME_CCTL2 = 0xFC;			/* DRUN mode */
		MC_ME_CADDR2 = (unsigned long)pcpu1;	/* reset address */
		MC_ME_CADDR2 |= 1;			/* reset request */
		}
	if (pcpu2)
		{
	MC_ME_MCTL = 0x30005AF0;			/* mode transition to DRUN */
	MC_ME_MCTL = 0x3000A50F;
	while (MC_ME.GS.B.S_MTRANS)			/* wait for transition complet
		;
	}

/*	Timer interrupt
 */
@interrupt void it_pit1(void)
	{
	extern int delayLED0, delayLED1, delayLED2;

	if (delayLED0)
		--delayLED0;
	if (delayLED1)
		--delayLED1;
	if (delayLED2)
		--delayLED2;
	PIT_TFLG1 = 1;			/* clear interrupt */
	INTC_EOIR0 = 0;			/* end of interrupt */
	}

/*	Toggle LED
 */
void ToggleLED(unsigned int position)
	{
	if (position & 1)
		SIUL2.GPDO[LED1].R ^= 1;
```

As you can see, starting the secondary CPUs (cpu1 and cpu2) only requires a few lines: you have to specify the reset address (like the symbol _stext1 that is at the beginning of the startup file crts1.s), then set a bit that resets the processor and then wait for the hardware reset procedure to complete.
The interrut routine (whose name is linked to the actual hardware interrupt address in the interrupt table in file vec5748.c) is also pretty simple: it decreases the led counters and does some internal interrupt management stuff.
And finally, the led toggling routine simply change the state of the GPIO pin to which the led is attached.

Overall this project uses about 1k of code (segment vtext below) and a few bytes of ram (the vector table is actually much bigger that the code itself !)

```
                        --------
                        Segments
                        --------

start 00fc0004 end 00fc0020 length   28 segment rchw
start 00fc0020 end 00fc0020 length    0 segment sconst
start 00fc2000 end 00fc4000 length 8192 segment vector
start 00fc4000 end 00fc4418 length 1048 segment vtext
start 40000004 end 40000004 length    0 segment sdata
start 40000004 end 40000018 length   20 segment sbss
start 00000000 end 00001de0 length 7648 segment .debug
start 00000000 end 0000046a length 1130 segment .info.
start 00fc4418 end 00fc4420 length    8 segment .init
```

We hope you will find this project useful as an example, but also as a starting point if you are developping bare-metal, multicore examples based on the Power Architecture chips.
If you would like this project ported (or extended) to your specific board or processor, don't hesitate to contact us at contact@cosmic.fr: helping our Customers to bring up their hardware is a service that is included in the compiler support package (conditions apply, please check with us).