Shared interrupt multi-core architecture for low power applications

US 20150143141A1

(19) United States

(12) Patent Application Publication

Echeverri Escobar et al.

(10) Pub. No.: US 2015/0143141 A1

(43) Pub. Date: May 21, 2015

(54) SHARED INTERRUPT MULTI-CORE ARCHITECTURE FOR LOW POWER APPLICATIONS

(71) Applicant: **NXP B.V.**, Eindhoven (NL)

(72) Inventors: **Juan Diego Echeverri Escobar**, Nijmegen (NL); **Jose de Jesus Pineda de Gyvez**, Eindhoven (NL)

(73) Assignee: **NXP B.V.**, Eindhoven (NL)

(21) Appl. No.: **14/083,398**

(22) Filed: **Nov. 18, 2013**

Publication Classification

(51) **Int. Cl.**

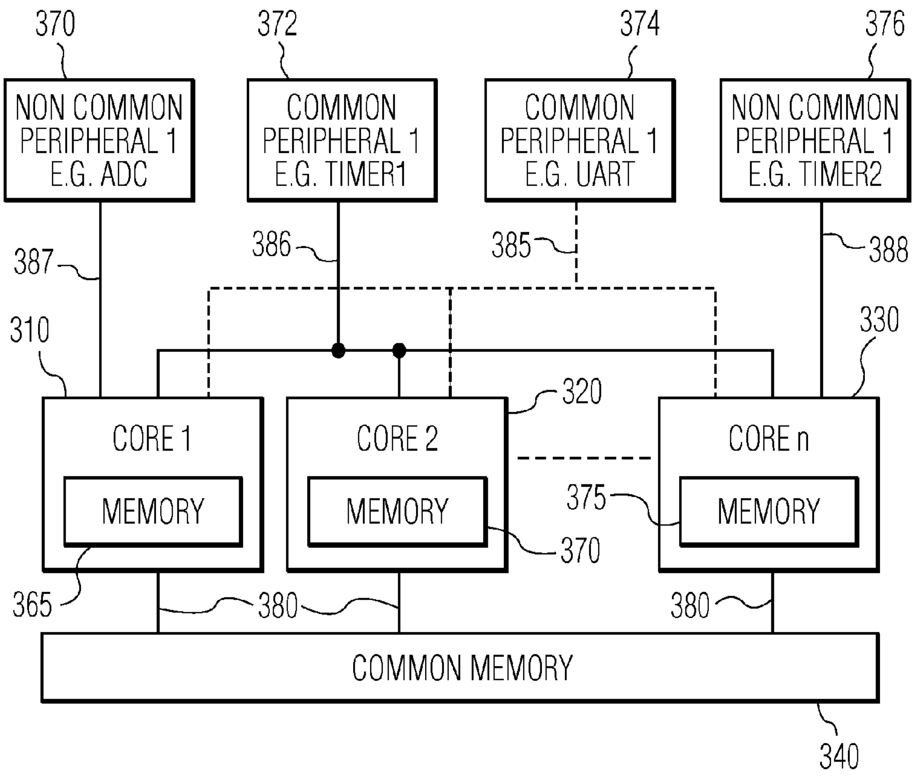
G06F 1/32	(2006.01)
G06F 13/24	(2006.01)

(52) **U.S. Cl.**

CPC ***G06F 1/3296*** (2013.01); ***G06F 13/24*** (2013.01)

(57) **ABSTRACT**

A multicore architecture is configured to exploit explicit task parallelism to save power by sharing interrupt sources that trigger independent tasks.


370
372
374
376
NON COMMON PERIPHERAL 1 E.G. ADC
COMMON PERIPHERAL 1 E.G. TIMER1
COMMON PERIPHERAL 1 E.G. UART
NON COMMON PERIPHERAL 1 E.G. TIMER2
387
386
385
388
310
330
320
CORE 1
CORE 2
CORE n
MEMORY
MEMORY
MEMORY
375
370
365
380
380
COMMON MEMORY
340

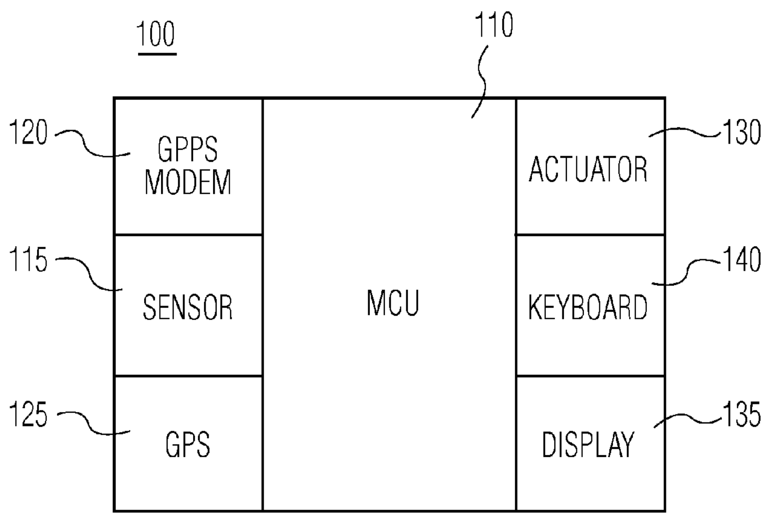


FIG. 1
PRIOR ART

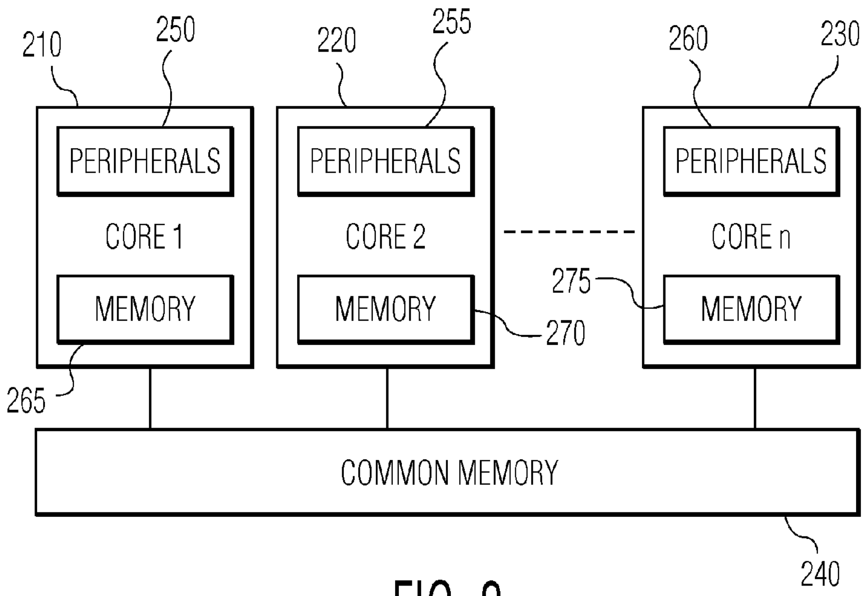


FIG. 2
PRIOR ART

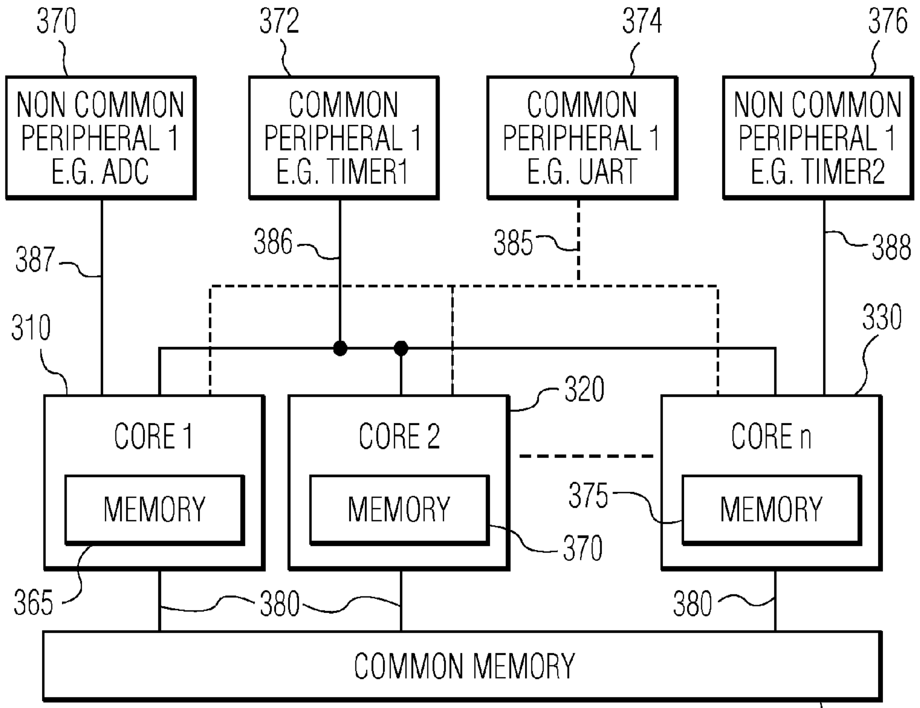

370
372
374
376
NON COMMON PERIPHERAL 1 E.G. ADC
COMMON PERIPHERAL 1 E.G. TIMER1
COMMON PERIPHERAL 1 E.G. UART
NON COMMON PERIPHERAL 1 E.G. TIMER2
387
386
385
388
310
330
320
CORE 1
CORE 2
CORE n
MEMORY
MEMORY
MEMORY
375
370
365
380
380
COMMON MEMORY
340

FIG. 3

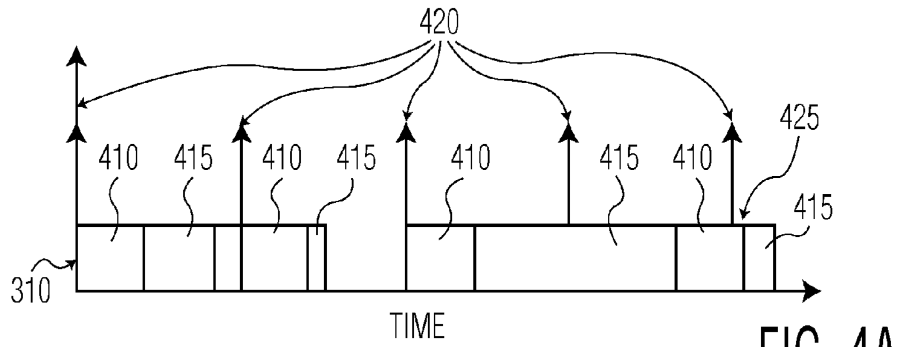


FIG. 4A
PRIOR ART

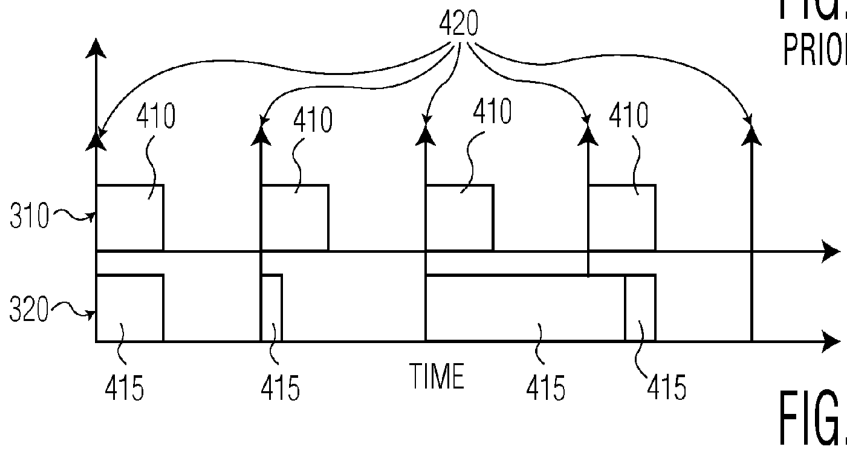


FIG. 4B

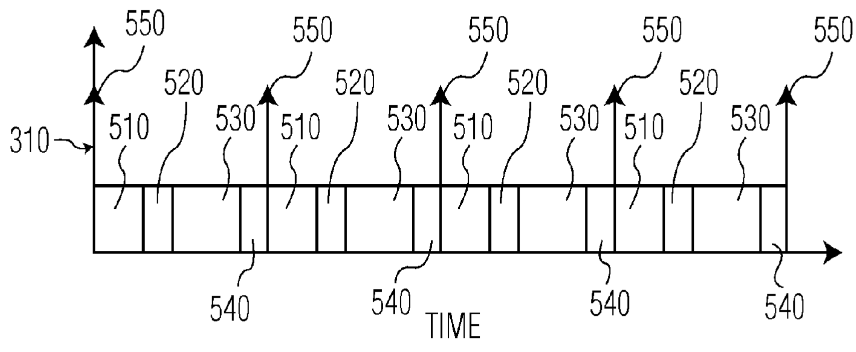


FIG. 5A
PRIOR ART

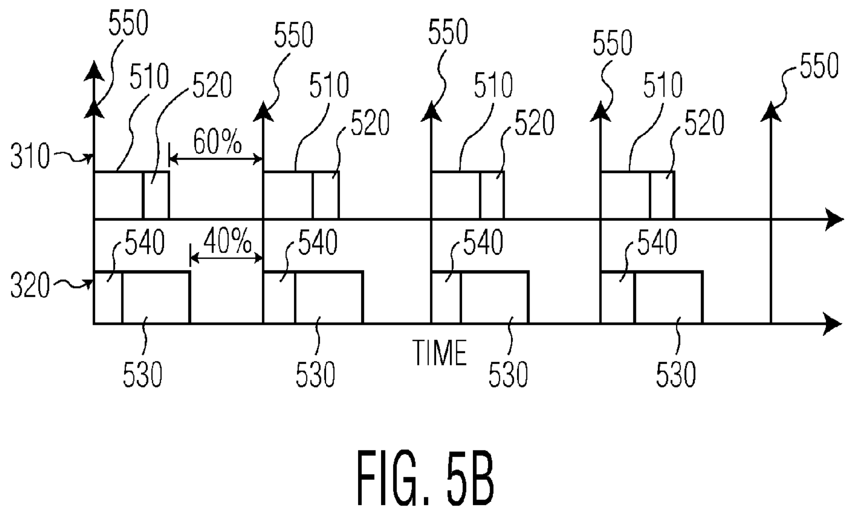


FIG. 5B

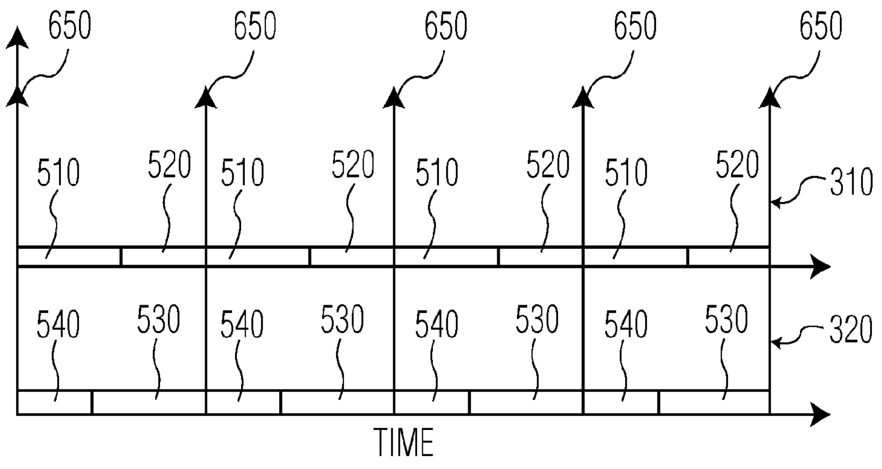

650
650
650
650
650
510
520
510
520
510
520
510
520
310
540
530
540
530
540
530
540
530
320
TIME

FIG. 6

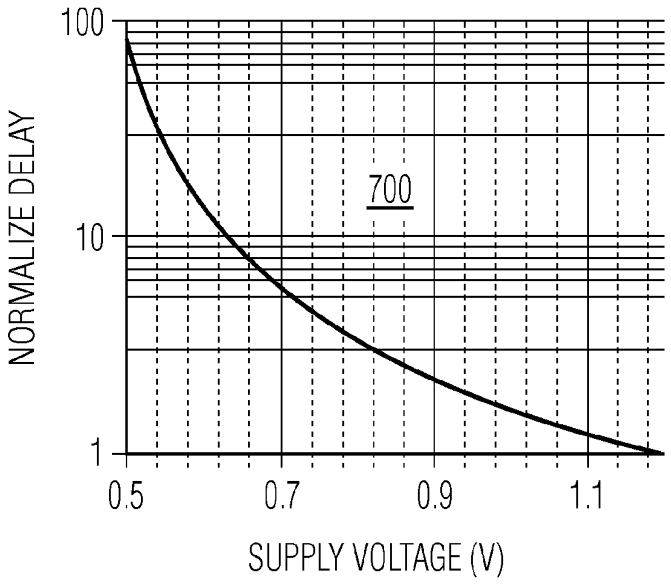

100
10
1
NORMALIZE DELAY
700
0.5
0.7
0.9
1.1
SUPPLY VOLTAGE (V)

FIG. 7

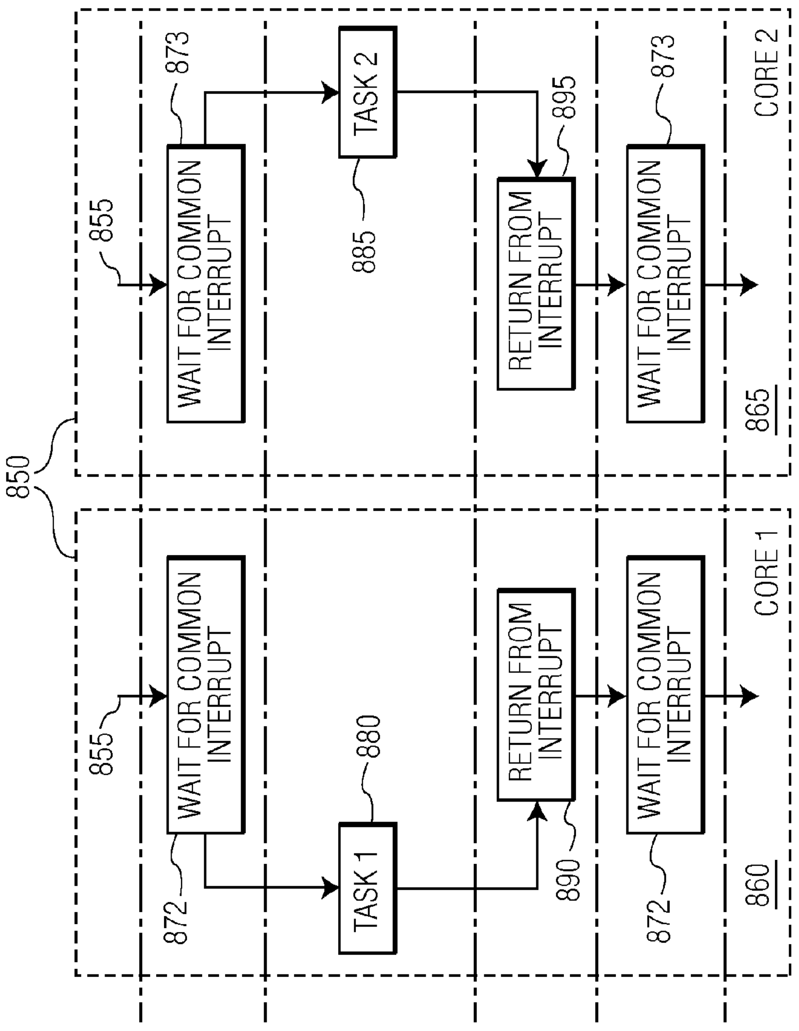


FIG. 8B

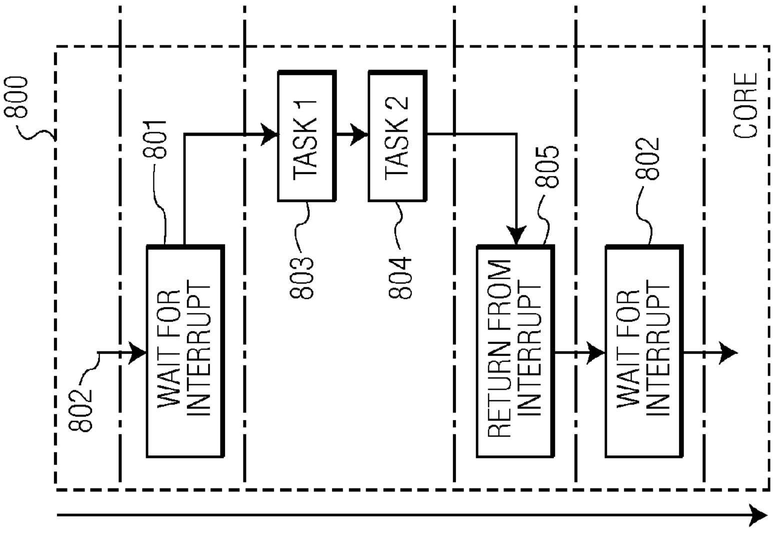


FIG. 8A
PRIOR ART

SHARED INTERRUPT MULTI-CORE ARCHITECTURE FOR LOW POWER APPLICATIONS

BACKGROUND

[0001] The use of parallel architecture in processors is a typical way to reduce power consumption without a performance penalty at the architectural level, see for example, "Low Power Digital CMOS Design, IEEE Journal of Solid State Circuits, pp. 473-484, April 1992. For a given performance level, the use of parallelism allows a task to be distributed and the frequency and voltage can typically be scaled down without performance losses.

[0002] There is a trend for multi-core architecture to be used even in small microcontrollers. The challenge is typically how to effectively and advantageously use the additional resources that are available in a multi-core architecture.

[0003] Many applications in the area of small microcontrollers are typically based on an interrupt that triggers the execution of multiple tasks. FIG. **1** shows system **100** that uses multiple peripherals connected to microcontroller (MCU) **110**. In a given time interval, e.g. 1 ms, MCU **110** checks sensor **115**, General Packet Radio Service (GPRS) modem **120** connectivity, Global Positioning System (GPS) **125** position, keyboard **140** for input, addresses actuator **130** and updates display **135** if needed. Typically, system **100** is implemented by setting up a timer (not shown) so that when the timer interrupt occurs, all tasks are executed. Explicit parallelism exists in system **100**. For example, the tasks of checking sensor **115** and addressing actuator **130** are independent of checking GPRS modem **120** connectivity and GPS **125** position.

[0004] However, typical microcontrollers do not provide for the capability of distributing tasks to different cores for execution. The microcontroller code needs to be written to manage all the tasks at the same time while utilizing only one resource. If some tasks can be executed in a second core but still share common memory with the first core, the implementation is simplified while the power consumption may be reduced through voltage and frequency scaling without performance losses.

[0005] FIG. **2** shows typical multi-core architecture **200** where core **210**, core **220** . . . and core **230** are connected and share common memory **240**. Core **210** has peripherals **250**, core **220** has peripherals **255** . . . and core **230** has peripherals **260** where each core **210**, **220** . . . and **230** has its own memory **265**, **270** . . . and **275**, respectively.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] FIG. **1** shows a prior art microcontroller with peripheral arrangement.

[0007] FIG. **2** shows a prior art embodiment of a prior art multi-core microcontroller.

[0008] FIG. **3** shows an embodiment in accordance with the invention.

[0009] FIG. **4***a* shows task execution in accordance with the prior art.

[0010] FIG. **4***b* shows task execution in an embodiment in accordance with the invention.

[0011] FIG. **5***a* shows task execution in accordance with the prior art.

[0012] FIG. **5***b* shows task execution in an embodiment in accordance with the invention.

[0013] FIG. **6** shows task execution in an embodiment in accordance with the invention.

[0014] FIG. **7** shows the normalized delay versus supply voltage in accordance with the invention.

[0015] FIG. **8***a* shows program flow in an embodiment accordance with the prior art.

[0016] FIG. **8***b* shows program flow in an embodiment accordance with the invention.

DETAILED DESCRIPTION

[0017] FIG. **3** shows an exemplary embodiment of common multi-core architecture **300** in accordance with the invention with cores **310**, **320** . . . and **330**. Core **310** contains memory **365**, core **320** contains memory **370** . . . and core **330** contains memory **375**. Cores **310**, **320** . . . and **330** connect to common memory **340** over common memory bus **380**. Cores **310**, **320** . . . and **330** are each connected to common peripherals **372** and **374** such as, for example, a timer, a Universal Asynchronous Receiver/Transmitter (UART), a General Purpose Input/Output (GPIO), a Serial Peripheral Interface Bus (SPI), Inter-Integrated Circuit bus (I2C), Analog-to-Digital Converter (ADC) or Digital-to-Analog Converter (DAC) by common interrupt lines **386** and **385**, respectively. In the exemplary embodiment shown in FIG. **3**, non-common peripherals **370** and **376** such as a timer or Analog-to-Digital Converter (ADC) are connected to cores **310** and **330** by dedicated interrupt lines **387** and **388**, respectively. This exemplary embodiment allows distribution of tasks triggered from common peripherals **372** and **374** to be distributed over an arbitrary number of "n" cores, cores **310**, **320** . . . and **330** and allows performance improvement or reduction of the power consumption through voltage scaling.

[0018] An exemplary embodiment in accordance with the invention uses "n" equal two cores with cores **310** and **320** (see FIG. **3**). With respect to FIG. **4***a*, an application is running on core **310** where between each timer tick **420**, real time task **410** needs to be executed as well as non-real time task **415**. However, using only core **310**, real time task **410** is not completed in the required time at time **425** because real time task is still executing at time **425**. In an embodiment in accordance with the invention, core **310** can be selected to execute real time tasks **410** as shown in FIG. **4***b* and core **320** is selected to execute non-real time tasks **415**. Hence, increased performance is provided by the addition of core **320**. Note that in accordance with the invention there is no requirement that either task be a real time task as shown in the example.

[0019] FIG. **5***a* shows an application running on core **310** with tasks **510**, **520**, **530** and **540** to be executed between each timer tick **550**. The dynamic power, $P_{dynamic}$, required for the application running on core **310** is modeled by Eq. (1):

$$P_{dynamic}=C_{eff}FV^2 \tag{1}$$

where C_{eff} is the total effective capacitance being switched per clock cycle, F is the running frequency of the application and V is the operating voltage. C_{eff} can be typically determined through post-layout simulation using standard electronic design automation tools.

[0020] FIG. **5***b* shows that after tasks **540** and **530** are moved to core **320**, performance is higher than required. In particular, idle time for core **310** is 60% and idle time for core **320** is 40%. This performance excess can be used to save power. The operating frequency of both core **310** and **320** can be lowered so that core **310** fulfills the timing requirements for both tasks **510** and **520** while core **320** fulfills the timing requirements for both tasks **530** and **540**. The appropriate operating frequency and voltage in accordance with the invention may be determined through user task profiling, for example. In this case, the user runs the desired application and determines the length of time required to execute the tasks. Then using the phase lock loop (PLL) and the programmable low-dropout (LDO) regulator in each core, the user can set the appropriate voltage and operating frequency. Note that this approach requires each core to have both a PLL and LDO.

[0021] The total dynamic power $P_{dynamic}=P_{core\ 310}+P_{core\ 320}$ required for the application running on both core **310** and **320** is modeled by Eq. (2) below (assuming no power consumption occurs when a core is idle) with reference to the example shown in FIG. **5***b* where core **310** is active 40% of the time and core **320** is active 60% of the time:

$$P_{dynamic}=0.4(C_{eff}FV^2)_{core310}+0.6(C_{eff}FV^2)_{core320} \tag{2}$$

where Eq. (2) assumes there is no overhead in connecting cores **310** and **320**. The coefficients, here "0.4" and "0.6", depend on how individual tasks are distributed between core **310** and core **320**, affecting idle time. The coefficients are determined by the execution time of the tasks and the coefficients change in dependence on the length of the tasks.

[0022] If the running frequency is lower, the voltage can be scaled to match the new running frequency as shown in FIG. **6** with timer ticks **650**. In the case shown in FIG. **6**, the power required for running the application is modeled as:

$$P_{dynamic(scaled)}=C_{eff}(0.4F)_{core310}V^2_{new1}+C_{eff}(0.6F)_{core320}V^2_{new2} \tag{3}$$

where V_{new1} and V_{new2} can be determined using a normalized delay vs. voltage relationship for a given semiconductor technology (e.g. 90 nm, 60 nm etc.). For this, a simple alpha-power model described by Eq. (4) below may be used (see, for example, "Alpha-Power Law MOSFET Model and its Applications to CMOS Inverter Delay and Other Formulas", IEEE Journal of Solid State Circuits, pp. 584-594, April (1990), incorporated by reference in its entirety):

$$D(V) \propto \frac{V}{(V-V_{th})^{\alpha}} \tag{4}$$

where V_{th} is the threshold voltage of the transistor and α is the parameter associated with a specific semiconductor process technology (e.g. 90 nm, 60 nm etc.). Assuming that V=1.2 volts, V_{th}=0.43 volts and α=2.2 which corresponds to 90 nm technology, the normalized delay with respect to the delay at 1.2 volts is modeled by:

$$D_{norm}(V_{new}) = \frac{D(V_{new})}{D(V)} = \frac{V_{new}(V-V_{th})^{\alpha}}{V(V_{new}-V_{th})^{\alpha}} \tag{5}$$

Plot **700** for Eq. (5) is shown in FIG. **7** for α=2.2.

[0023] For core **310**, the running frequency is scaled down to 40% of the original frequency F for a particular task. Using FIG. **7** with a normalized delay (where normalized delay is defined as the ratio between the new clock period and the old clock period) of 2.5 (1.0/0.4) for core **310**, the supply voltage, V_{new1}, is given by 0.86 volts. Similarly, for core **320** the running frequency is scaled down to 60% of the original frequency F. Using FIG. **7** with a normalized delay of 1.66 (1.0/0.6), the supply voltage. V_{new2}, is given by 0.98 volts.

[0024] The power savings factor $P_{savings}$ is modeled by Eq. (6) below (assuming insignificant power leakage):

$$P_{savings} = \frac{P_{dynamic}}{P_{dynamic(scaled)}} = \frac{V^2}{0.4\ V^2_{new1}+0.6\ V^2_{new2}} \tag{6}$$

with V=1.2 volts, V_{new1}=0.86 volts and V_{new2}=0.98 volts and gives $P_{savings}$=1.65 as the power savings factor.

[0025] In accordance with the invention, the power savings can be achieved if both cores **310** and **320** have a PLL (not shown) and a programmable LDO (not shown) (note that the power supply may be external to cores **310** and **320** in which case there is an external programmable LDO adjustable by the user), a DC-DC converter (typically for higher loads) or a switch capacitor converter (typically for lower loads). With respect to the example discussed above, after the user determines that the task running in core **310** executes for 40% of the time (with the idle time being 60%), the user sets up division of the output frequency by 2.5 using the configuration registers of the PLL integrated into core **310**. Then, using the programmable LDO in core **310**, the user sets the voltage to 0.86 volts for core **310**. The same setup procedure is executed in core **320**, but in this case the PLL integrated into core **320** is set up to divide the output frequency by 1.66 and the programmable output voltage is set to 0.98 volts using the programmable LDO in core **320**. After the core setup is completed, the task can be executed with the appropriate power savings factor, $P_{savings}$.

[0026] The analysis above for power savings factor, $P_{savings}$, may be extended to n cores as modeled by Eq. (7):

$$P_{savings} = \frac{V^2}{\sum_{i=1}^{n} l_i V^2_{new_i}} \tag{7}$$

[0027] where l_i is the frequency scaling factor for a task running on processor i and V_{new_i} is the voltage that corresponds to the new operating frequency.

[0028] For example, a multi-core system having 10 cores where each core now runs at 1/10th of the original operating frequency results in the supply voltage for each core being reduced to 0.63 volts (see FIG. **7**). Using Eq. (7) this gives a power savings factor, $P_{savings}$, of 3.62 where n=10, V=1.2 volts, l_i=0.1 and V_{new_i}=0.63 volts.

[0029] FIGS. **8***a* and **8***b* show how program flow is typically modified by rewriting the program code to exploit the parallelism in a two core architecture in accordance with the invention to achieve power savings. In FIG. **8***a*, single core microcontroller **800** is in "wait for interrupt" state **801** when it receives "interrupt" **802**. In response to interrupt **802**, microcontroller **800** sequentially executes task **803** and task **804**. Upon completion of task **804**, "return from interrupt" instruction **805** is executed and microcontroller **800** returns to "wait for interrupt" state **801**.

[0030] In FIG. **8***b*, dual core microcontroller **850** has cores **860** and **865** which are both in "wait for common interrupt" states **871** and **873**, respectively, when "common interrupt" **855** is received by cores **860** and **865**. In response to "common interrupt" **855**, core **860** executes task **880** while core **865** executes task **885**. Upon completion of task **880**, "return from interrupt" instruction **890** is executed and core **860** returns to "wait for interrupt" state **872** while upon completion of task **885**, "return from interrupt" instruction **895** is executed and core **865** returns to "wait for interrupt" state **873**.

[0031] While the invention has been described in conjunction with specific embodiments, it is evident to those skilled in the art that many alternatives, modifications, and variations will be apparent in light of the foregoing description. Accordingly, the invention is intended to embrace all other such alternatives, modifications, and variations that fall within the spirit and scope of the appended claims.

1. A microcontroller system comprising:

a microcontroller having a plurality of cores, each of the plurality of cores having an interrupt controller, an operating frequency and voltage; and

common peripherals electrically coupled to each of the plurality of cores by interrupt lines, the interrupt controller of each of the plurality of cores handling a common interrupt received from one of the common peripherals such that a first task signaled by the common interrupt is handled in parallel with a second task signaled by the common interrupt by having the first task running on a first one of the plurality of cores and the second task running on a second one of the plurality of cores wherein the operating frequency and voltage of the first one of the plurality of cores and the operating frequency and voltage of the second one of the plurality of cores are scaled down to reduce power consumption.

2. The microcontroller system of claim **1** wherein the plurality of cores is two.

3. The microcontroller system of claim **1** wherein one of the plurality of cores comprises a switch capacitor converter.

4. The microcontroller system of claim **1** wherein one of the plurality of cores comprises a programmable low-dropout regulator.

5. The microcontroller system of claim **1** wherein the operating voltage of the first and second one of the plurality of cores is scaled down according to a normalized delay versus a voltage relationship for a specific semiconductor process technology.

6. The microcontroller system of claim **1** wherein the one of the common peripherals is selected from the group consisting of a timer, a UART and an analog to digital converter.

7. The microcontroller system of claim **1** wherein the first task is a real time task.

8. The microcontroller system of claim **1** wherein a power supply is external to each of the plurality of cores.

9. The microcontroller system of claim **1** wherein a user determines the scaled down voltage and operating frequency for the second task.

10. A method for making a microcontroller system comprising:

providing a microcontroller having a plurality of cores, each of the plurality of cores having an interrupt controller, an operating frequency and voltage; and

providing common peripherals electrically coupled to each of the plurality of cores by interrupt lines, the interrupt controller of each of the plurality of cores handling a common interrupt received from one of the common peripherals such that a first task signaled by the common interrupt is handled in parallel with a second task signaled by the common interrupt by having the first task running on a first one of the plurality of cores and the second task running on a second one of the plurality of cores wherein the operating frequency and voltage of the first one of the plurality of cores and the operating frequency and voltage of the second one of the plurality of cores are scaled down to a reduced operating frequency and a reduced operating voltage.

* * * * *