RC64 Software Development Tools

Yam Zamir, Isaac Carasso, Dan Dagan, Tsvika Israeli, Peleg Aviely
Ramon Chips, Ltd., 5 HaCarmel Street, Yoqneam Illit 2069201, Israel
+972- 722 216869

***Abstract* - RC64 many-core processor was developed targeting space applications, with an approach of software-defined payload in mind. RC64 includes 64 DSP/CPU cores, 23 I/O channels, 4MB of shared memory and a hardware task scheduler. Writing correct and efficient (high performance, low power) data streaming applications for RC64 is a complex and challenging task. RC64 software tools were constructed to support programmers along the development path. The tools enable the programmer to plan, express and exercise parallelism, develop and debug the application, detect and remove performance bottlenecks and achieve desirable performance goals. Due to the high level of parallel activity taking place on the many RC64 units, some new concepts were developed to adjust traditional tools to the massively parallel RC64.**

1. INTRODUCTION

RC64 is designed for high degree of parallelism to maximize utilization of its 64-compute cores and enable cores to operate concurrently with DMA activity to/from the I/O ports.

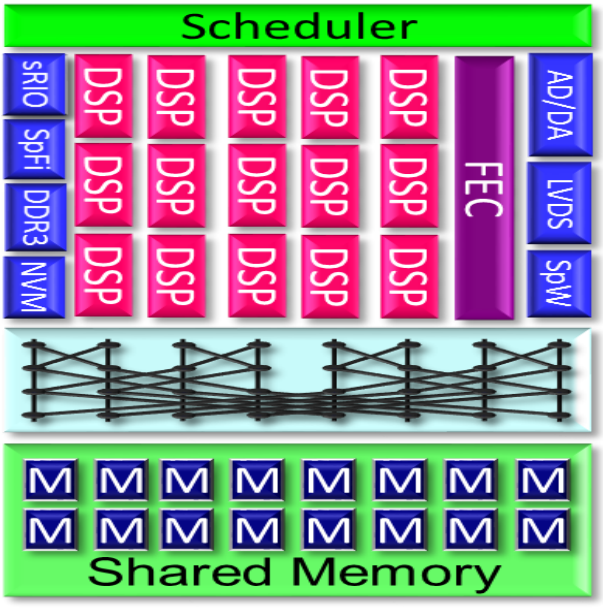


Figure 1: RC64 Block Diagram

RC64 application software is organized as a single application, single system image. The basic building block of concurrency is called *the Task*.

The Task is the operational code entity allocated to cores by the hardware scheduler for execution. RC64 Tasking concept define regular (single instance) tasks as well as duplicable (multiple instance) tasks.

At any given time, cores either run allocated tasks or are waiting for task allocation, or are in power saving mode awaiting to be interrupted for task allocation.

Special tools were planned and implemented to assist in development, debug and performance tuning of RC64 software.

The main tools are

- **Task graph compiler** to define the parallelism scheme and its derived task dependencies

- **Many-core debugger** for program code debug

- **Many core profiler** to assist in performance bottleneck detection and removal

- **Event Tracer** records time stamped events. Events include time stamped events such as task start and task end, interrupt handler start and end or application level user defined events (can be recorded from all cores). The analysis of traced events can assist in both debug (understand sequence of events as took place in real time) and performance analysis (understand actual task execution times, etc.)

Shared memory bandwidth planner defines the amount of shared memory bandwidth each I/O channel will be granted.

2. MANY-CORE DEBUGGER

In RC64 processor design, attention was given to enable 64-core wide synchronized execution stop (entering debug mode). Special focus was given as well to provide support for synchronized, 64-core-wide resume command to enable all cores to resume execution (out of debug mode) while keeping as close as possible the original execution time schedule.

The 64-core-wide stop and 64-core-wide resume during a debug session enables:

- Observing all program data (e.g. stack-based automatic variables) during failure (or breakpoint, while debugging towards the failure point), including global variables and buffers in shared memory. The 64-core wide stop provides stable memory content while system is at debug state, as all cores have stopped writing into shared memory. As shared memory content is stable, it reflects correctly the memory status for that failure point (or breakpoint) as does not change.

- Maintaining relative timing between different operations (coming from different tasks running on different cores) by synchronized, system wide stop and resume leads to high probability that timing-sensitive behaviors/scenarios (correct scenarios and incorrect scenarios (bugs)) will be reproduced when running with the debugger (and using its breakpoints)

Each of the 64 cores includes several break capabilities, which allow the core to break on any of the following events:

- Software Breakpoint
- Hardware Breakpoint
- Data Breakpoint / Watchpoint – Data Address
- Data Breakpoint / Watchpoint – address and value combination

The debugger supports all the above break conditions. When the user defines any of the above conditions as a breakpoint, and should one (or more) of the 64 cores fulfil the break condition, the whole RC64 processor will stop/suspend.

The parallel program running under the debugger's control is in one of two states:

- Running
- Suspended (stopped)

While in suspended state, one of the cores is always selected to be "the core in focus". Debugger commands, which target a single core (for example view of core register values) will be interpreted as referring to the "core in focus". The GUI enables the user switch to a different core (set a different core as the "core in focus"), and from here onwards, all core-specific commands will refer to the new "core in focus".

3. MANY-CORE PROFILER

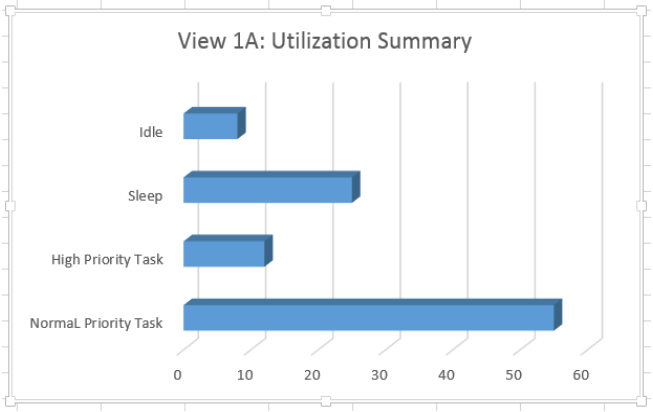
The profiler is intended to provide users with information of "where time is actually spent" and to provide the user with "under the hood" execution data which will lead to identification (and eventually removal) of performance bottlenecks.

The challenges in many core system profiler include:

- Low intrusiveness data collection
- Collection of profile data from a large number of executing cores
- Transferring the large amount of accumulated profile data (either to attached DDR memory or over SpaceWire connection to the PC host)

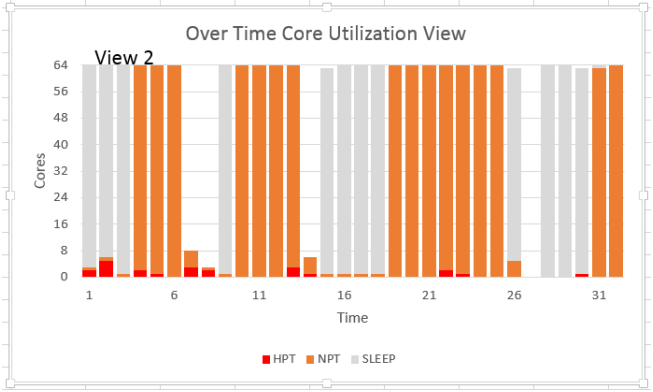
The profiler data will be analyzed post-execution and will provide the user with the following information:

- Overall system utilization view
- Breakdown by task
- For a specific task, breakdown by function



An example for the utilization chart is shown above. Typically, higher performance is reflected by higher percentage of time where the cores actually execute code of tasks (and are not in Idle mode or in Sleep mode).

An additional view shown system utilization over time:



4. EVENT TRACER

The RC64 event tracer capability was designed to assist with both debug tasks and performance tuning tasks.

The hardware implementation records, with minimal intrusiveness, events such as task start, task end, interrupt handling start and interrupt handling end. It then packs the records of the recorded traces in a compact way in shared memory, ready to be sent to either DDR memory or over the network to an accumulating host for post execution analysis.

All traces are time stamped and the granularity of the event collection is software defined.

Operating Method of the Event Tracer

The memory management side of the Event Tracer can be implemented using a double buffer.

While the "previous traces buffer" is being transferred to the host for use in pre-execution analysis, the "current traces buffer" is provided to the event tracing hardware, and is used by the hardware to record traces of current activity.

When both transfer and buffer write are completed, software will switch the roles between the two buffers.

Using Event Tracer as debug aid

Many "concurrency related raise conditions" with lead to software bugs can be detected by viewing the sequence of concurrent events, as they actually took place in chronological order. Viewing the actual sequence can show the programmer unexpected behaviors that his current implementation of the software system did not handle correctly.

Using Event Tracer as performance Tuning aid

As all recorded events are time stamped (using RC64 cycle counter, which is 64bit wide), viewing the recorded events can provide a clearer understanding of the duration required for various activities.

Among the activities the user will be able to understand total time required for execution of regular and duplicable (parallel, multi instance) tasks and total time required for completion of I/O transactions via all channel types.

The capability to view the sequence of events can assist in understanding activities taking longer than expected, performance hiccups and outliers.

Using Event Tracer to record application level events

Event Tracer hardware support in RC64 enables the application developer to define application level events (e.g. *frame handling start* and *frame handling end* within an application which handles image frames) and to record the events using the low intrusiveness Event Tracer hardware. The application level events can be recorded with or without the hardware scheduler events, which relate to RC64 task execution. The event tracer mechanism is configurable by software and can configure scheduler related events, application level events or both.

5. SHARED MEMORY BANDWIDTH PLANNER

In RC64 the amount of I/O channels, which may access shared memory, can be as high as 23 channels. Different channels may be active for different applications, and as the amount of shared memory bandwidth required by each channel is application dependent, there is a need for the application software architect to design and define the amount of shared memory bandwidth allocated for each channel.

RC64 hardware includes a mechanism to define the allocated bandwidth for each channel. The amount of bandwidth allocated for the I/O channels may also affect the effective shared memory bandwidth available for the sixty four compute cores.

The shared memory bandwidth planner tool will enable the architect to define the bandwidth requirements, and create a table, which will be programmed into RC64 (by its executive) to communicate to RC64 the bandwidth allocations for the various channels.

6. CONCLUSIONS

Writing parallel software for streaming data applications using many core RC64 represent new challenges for the tasks of parallelism definition, application debug and application performance tuning.

Many-core tools, which incorporate unique technologies, were developed by Ramon Chips. The tools will be used by software architects and developers to develop, debug and improve performance of streaming data software applications for RC64.

REFERENCES

[1] Ran Ginosar, Peleg Aviely, Tsvika Israeli and Henri Meirov. "RC64: High Performance Rad Hard Manycore". Aerospace Conference, IEEE, March 2016.

[2] Steve Parkes, Chris McClements, David McLaren, Albert Ferrer Florit, and Alberto Gonzalez Villafranca. "SpaceFibre: A multi-Gigabit/s interconnect for spacecraft onboard data handling." In Aerospace Conference, pp. 1-13. IEEE, 2015.

[3] SpaceWire homepage - http://spacewire.esa.int/content/Home/HomeIntro.php