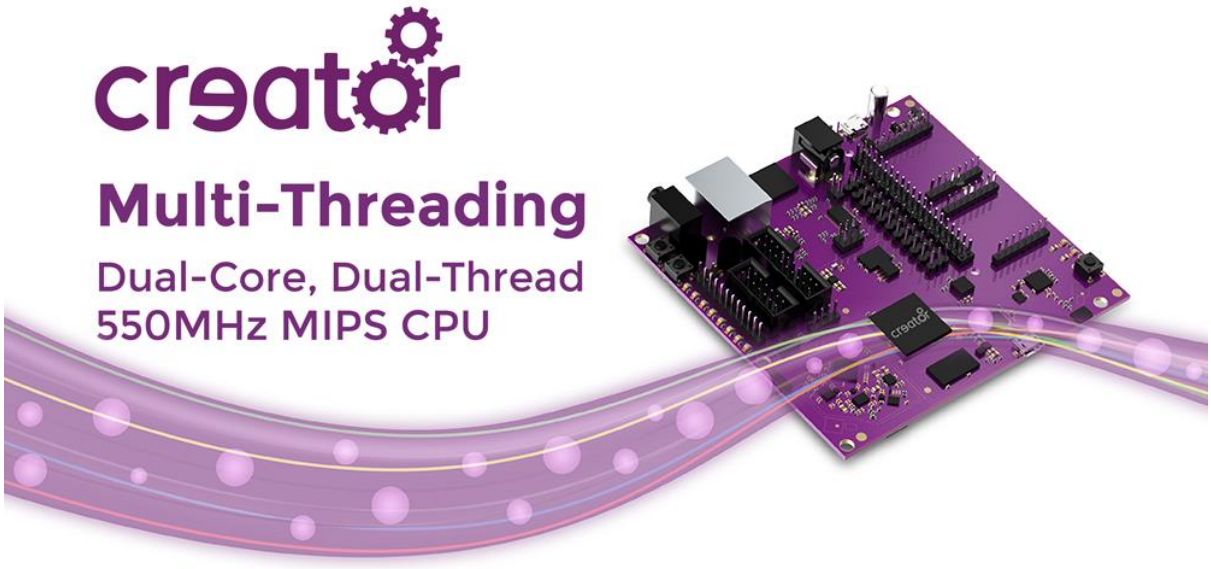
MIPS Creator Ci40

Multithreading Benchmarks



Filename	:	MIPS Creator Ci40.Multithreading Benchmarks.docx
Version	:	1.03.50 External Issue
Issue Date	:	23 Oct 2016
Author	:	Imagination Technologies Limited

Contents

1. Introduction 3
1.1. Ci40 Hub Hardware Overview 3
1.2. Board Setup 4
1.3. Host Setup for cross compiling 5
2. Multi-Threading 6
2.1. Introduction 6
2.2. Accumulator 6
2.3. CoreMark – An industry standard benchmark 8
2.4. Internet of Things 9
3. Conclusion 12
4. Where to find additional information 12

List of Figures

Figure 1 Ci40 Creator board 3
Figure 2 Single thread execution on a single CPU 6
Figure 3 Multi-thread (VPEs) execution on a single CPU 6
Figure 4 Ci40 with two click sensor boards 9

1. Introduction

1.1. Ci40 Hub Hardware Overview

The Creator Ci40 IoT Hub is a readily available and powerful development platform designed to allow developers to easily create connected projects based on MIPS hardware.

Distributed globally by the likes of Mouser, the board can be purchased at

http://www.mouser.co.uk/ProductDetail/Imagination-Technologies/VL-62913/

The high-performance, low-power microcomputer features a dual core MIPS interAptiv CPU clocked at 550MHz. Each core is a multi-threaded design with two threads per core. Under SMP Linux these threads are visible as logical cores numbered from 0 to 3 where (0,1) represents core 1 and (2,3) represents core 2. Multi-threading is explained in section 2 on page 6.The connectivity protocols supported include 802.11 ac 2x2, 802.15.4, 6LoWPAN Bluetooth 4.1 (Smart + Classic) and wired 802.3 Ethernet. Memory and storage specifications include, 256MB RAM, 512MB NAND flash, 16MB NOR flash. OpenWrt is installed on NAND flash.

Other operating systems can be supported

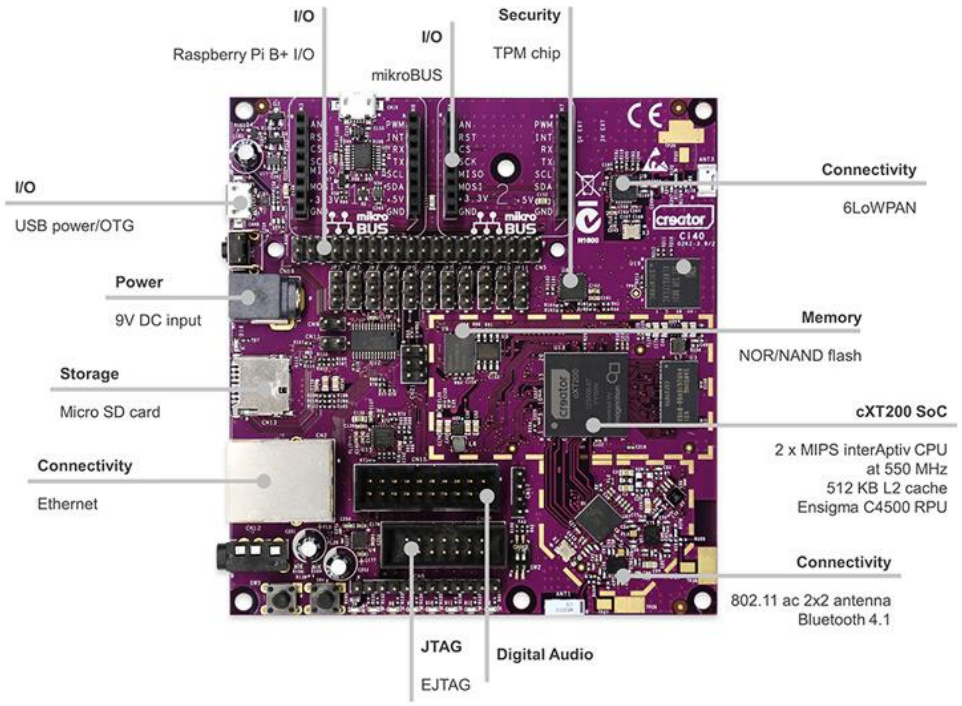


Figure 1 Ci40 Creator board

Creator Ci40 can be used as a stand-alone development board or as part of an IoT development kit such as the Creator Ci40 IoT kit, which is designed to allow fast prototyping of IoT products. In the case of IoT, power consumption plays a significant role. In modern CPUs, the more idle a CPU is, lesser the power it consumes. Multi-threading can improve throughput and increase the idle time of a CPU core.

The board has slots for, two MikroElektronika ‘click’ sensor boards. The slots are mikroBus I/O slots as shown in the above diagram. The board also supports additional microSD card for installing a larger root file system, with the board.

An x86 host running Linux is required to help us cross-compile and communicate with the board.

1.2. Board Setup

The following steps outline how to get the Ci40 connected to your host network. For in-depth 'Getting Started' instructions please refer to the Creator Ci40 documentation website which has a wealth of information from board schematics to datasheets, detailed instructions and videos for assistance with the board.

- https://docs.creatordev.io/ci40/

To connect your Ci40 to your host network

1. First, power-on the board, by connecting the AC-DC adapter (9V) to the power input shown in Figure 1.
2. Connect the Ci40 serial micro-USB (between the mikroBUS-1 strips) to the x86 host machine. The baud-rate to communicate with the board serially is 115200. An application like 'screen' can be used on the **host** as follows.

```
screen /dev/ttyUSBx 115200
```

   The system will boot into OpenWrt login prompt where we login using the default credentials (username: root, password: root).

3. Setup network connection:

   To connect to host networks as a client you need to edit files then restart the network process :

   Edit /etc/config/wireless

   Use an editor such as vi to open and edit /etc/config/wireless. For example:

```
vi /etc/config/wireless
```

   Add the following lines, to identify host wireless router.

```
config wifi-iface
                    option device 'radio0'
                    option network 'sta'
                    option mode 'sta'
                    option ssid '<your SSID>'
                    option encryption 'psk2'
                    option key '<your password>'
```

   Edit /etc/config/network

   Now edit /etc/config/network in the same way:

```
vi /etc/config/network
```

   In this file you need to comment out the 'defaultroute' line under 'sta' by adding #.

```
config 'interface' 'sta'
                    option 'proto'    'dhcp'
#     option 'defaultroute' '0'
```

   Restart the network process

   Finally, restart the network process using the following command

```
/etc/init/d/network restart
```

This should connect the Ci40 board to the same network as the host-machine. This setup enables us to transfer files between the board and host machine. This comes in handy, to transfer cross-compiled binaries later.

1.3. Host Setup for cross compiling

Section two of this document discusses various examples of hardware multi-threading using code examples. These code examples need to be cross compiled for our target platform (Ci40) on our host machine.

This sub-section discusses configuring the host for cross-compiling.

To download the source package and configuration script

The source package can be cloned on to the host machine using the following command:

```
git clone https://github.com/CreatorDev/Ci40-multithreading-whitepaper
```

The package includes:

- Example source files (written in C)
- Wrapper API library for various sensors (click boards) and communication channels (I2C, GPOI, SPI etc).
- `setup.sh` file

This script file downloads and installs the MIPS Codescape Toolchain.

To configure the script and download the toolchain

Execute the script as follows.

```
chmod +x setup.sh
./setup.sh
```

This should download and install the appropriate toolchain for your host OS, which can be later used with the make files included in the package.

Other versions of the toolchain can be found at this address:

http://codescape-mips-sdk.imgtec.com/components/toolchain/2016.05-03/downloads.html

2. Multi-Threading

2.1. Introduction

A salient feature of the Creator Ci40, making it great for IoT applications is temporal multithreading that built into the interAptiv core. This feature ensures high CPU utilization on tasks which are inherently parallel and independent of each other.

Multi-threading (MT) is a hardware level multi-threading feature where, a single processor core appears as multiple 'virtual' cores to an SMP operating system. We term each of these multiple cores as Virtual Processing Elements (VPEs). Each VPE has its own copy of the processor state. The VPEs of a core share resources like ALU, FPU and registers of the CPU. They also share a single pipeline.

MT aims to improve CPU utilization, thus getting more work done in a given cycle of the CPU. To understand how Multi-threading benefits us, let us consider, a single threaded CPU core's execution.

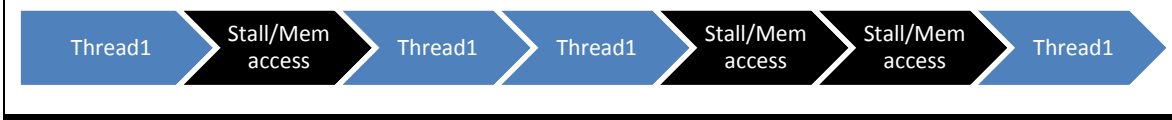


Figure 2 Single thread execution on a single CPU

Memory access can be often slower than the execution speed. On a single-threaded CPU, execution has to wait (stall) until data from memory is available. As shown in Figure 2, during the stall/memory access phase the CPU is basically idle. In this time the CPU could do more useful work. This is essentially what we can remedy with MT. Now let us look at the same with two threads, running on a MT core with two VPEs.

Figure 3 Multi-thread (VPEs) execution on a single CPU

From Figure 3, it is clear that the problem of CPU sitting idle, now has been resolved, they we are getting more work done in a given slice of time.

This can be very useful in IoT applications where we may be reading multiple sensors in parallel for example. The speed up in running time, while running two threads, may not be twice as a single thread, but the running time will certainly be less than running two threads in a sequential manner. In the next few sub-sections, we are going to look at some example applications and an industry standard benchmark benefiting from multi-threading.

2.2. Accumulator

An accumulator program sums up 'n' values in an array into a single variable. This very simple program, can be used to showcase the benefits of multi-threading. Here is a snippet of the accumulator function.

```
int *a;
a = malloc(sizeof(int)*DATA_SIZE)
clock_gettime(CLOCK_THREAD_CPUTIME_ID, &begin);
for (int j=0; j < 10000; j++) {
      for(i=0 ;i< DATA_SIZE; i++){
        sum += a[i];

    }
}
```

```
clock_gettime(CLOCK_THREAD_CPUTIME_ID, &end);
```

The above snippet shows our accumulator method, where we add all the elements of the array of DATA_SIZE , 10000 times, this is timed with the help of clock_gettime function. The full source is available in the package, under (on the x86 host)

- `/accumulator/multi_thread_acc.c`

The application can schedule the threads on various cores exposed to SMP Linux (VPEs).

To build the accumulator example

We can create the executable by using the makefile provided with the package.

```
cd accumulator/
make
```

To transfer the binary to target

Using scp, command on the Ci40 board, transfer the compiled binary from the host machine:

```
scp <user>@<host>:/path/to/accumulator/multi_thread_add
```

Replace <host> with your host name and <user> with your username.

To run the accumulator binary on the target

We can execute and understand the advantages of multi-threading as follows.

```
#CPU Topology (0,1) threads of core 1 and (2,3) threads of core 2
#Scheduling a single thread to run on cpu 2
root@OpenWrt:/diskr# ./multi_thread_acc 2
Number of processors: 4
Scheduled Thread ID: 2002551616, CPU: 2
Thread ID: 2002551616, CPU: 2 Time taken(s): 1.130000
#Scheduling two pthreads on CPUs 2, 3 (Multi-Threading)
#Read the results as two threads starting at the same time.
#i.e We do not sum up the times as they are run in parallel.
root@OpenWrt:/diskr# ./multi_thread_acc 2 3
Number of processors: 4
Scheduled Thread ID: 2007835456, CPU: 2
Scheduled Thread ID: 1999446848, CPU: 3
Thread ID: 2007835456, CPU: 2 Time taken(s): 1.880000
Thread ID: 1999446848, CPU: 3 Time taken(s): 1.880000
#Note running twice the single workload results only takes 1.66x more time
#It would 2x more time in sequential execution
#we are 34% faster and efficient. on a single CPU using multi-threading

#What if we run on different cores?(multi-core)
#This will help us understand differences between MT and multi-core
root@OpenWrt:/diskr# ./multi_thread_acc 2 1
Number of processors: 4
Scheduled Thread ID: 2005631808, CPU: 2
Scheduled Thread ID: 1997243200, CPU: 1
Thread ID: 2005631808, CPU: 2 Time taken(s): 1.100000
Thread ID: 1997243200, CPU: 1 Time taken(s): 1.140000
#Again running in parallel they take almost the same time as single core
run
```

```
#The primary benefit of MT is increased efficiency of a single core.
```

As shown in the results, multi-threading improves single-core performance by 17 percent, making it more efficient. In cases where the workload is lighter, less CPU intensive and more I/O intensive, speed up from MT can be even higher, approaching multi-core speedups. To experiment further, tweak the code provided, to further study the benefits of multi-threading.

2.3. CoreMark – An industry standard benchmark

CoreMark is a simple, industry standard, benchmark that is designed specifically to test the functionality of a processor core. CoreMark is not system dependent (the data it resides in CPU cache), therefore it functions the same regardless of the platform (e.g. big/little endian, high-end or low-end processor). Running CoreMark produces a single-number score allowing users to make quick comparisons between processors.

The code is written in C and contains implementations of the following algorithms; list processing (find and sort), matrix manipulation (common matrix operations), state machine (determine if an input stream contains valid numbers), and CRC. To learn more about CoreMark visit the official web page.

- http://www.eembc.org/coremark/index.php

Our package includes a pre-compiled binary for CoreMark found at.

- `/coremark/coremark.elf`

To transfer the binary to target

Using the scp, command on the Ci40 board transfer the compiled binary from the host machine.

```
scp <user>@<host>:/path/to/coremark/coremark.elf .
```

To run the coremark binary on the target

Before we execute the binary we need to disable one CPU core (in this case core 2, that is, threads/CPUs 2,3). This can be achieved by typing the following command in the shell (on Ci40).

```
for i in 2 3 do; echo 0 > /sys/devices/system/cpu/cpu$i/online; done
```

We need to disable the 2nd core to ensure the benchmark can be scheduled only on a single core.

The following commands shows how to run and interpret the benchmark.

```
#single thread run
root@OpenWrt:/diskr# ./coremark.elf
Iterations/Sec   : 1773.993259
#Binary prints other information apart from iterations/sec
#They have been removed for simpliciry
#Iterations/Sec is the main result of this benchmark
#We compute the final score using the formula Coremark/Mhz =
(Iterations/Sec)/CPU_freq_mhz
#The ci40 runs around 536Mhz on openwrt(1773.993259/536) => 3.3
Coremark/Mhz

#Dual thread run
root@OpenWrt:/diskr# ./coremark.elf M2
Iterations/Sec   : 2071.823204
#Coremark/Mhz 3.8. 15% performance improvement on a single core using Dual
threads.
```

As the results above show, MT has a clear impact on performance, increasing our final score by 15 percent on a single core CPU.

Now we can turn on the second core by using the following command on the shell. It is very important to do this before proceeding with any other examples or tests.

```
for i in 2 3 do; echo 1 > /sys/devices/system/cpu/cpu$i/online; done
```

2.4. Internet of Things

We look at a sample Internet of Things (IoT) application in this sub-section.

In our sample application we use two mikroElektronika Click sensor boards. They need to be placed on the two mikroBus slots available on the Ci40. The following image shows the sensors place on the board.

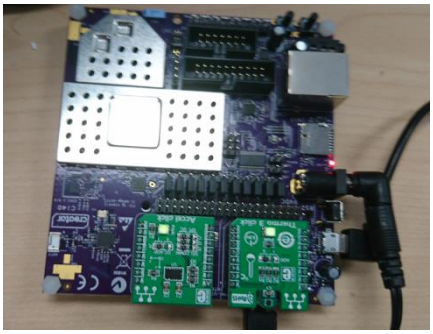


Figure 4 Ci40 with two click sensor boards

The two sensors we will be using are Thermo3 temperature sensor and Accel sensor(Accelerometer). They can found at.

- http://www.mikroe.com/click/thermo3/
- http://www.mikroe.com/click/accel/

However, many alternative sensors can be used for this sensor with some minor modifications.

Let us examine the temperature sensor read function snippet.

```
clock_gettime(CLOCK_THREAD_CPUTIME_ID,&begin);
i2c_init();
i2c_select_bus(MIKROBUS_1);

thermo3_click_enable(0);
for(int i=0;i<STEPS;i++){
  thermo3_click_get_temperature(&temperature);
        sum += temperature;
}
clock_gettime(CLOCK_THREAD_CPUTIME_ID,&begin)
avg = sum/STEPS
```

The function initiates an I2C bus communication to the sensor from the board, read the temperature sensor STEPS times and finally computes the average temperature. It measures the time taken to read and sums the values.

Next, let us look at the code snippet for the accel sensor.

```
clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&begin);
spi_init();
spi_select_bus(MIKROBUS_2);
accel_click_enable();
for(int i=0;i<STEPS;i++){
```

```
        accel_click_get_measure(&a[i].X, &a[i].Y, &a[i].Z);
        roll = atan2(a[i].Y, a[i].Z) * 180/M_PI;
        pitch = atan2(-a[i].X, sqrt(a[i].Y*a[i].Y + a[i].Z*a[i].Z)) *
180/M_PI;
    }
    accel_click_disable();
    spi_release();
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&end);
```

The function initiates a SPI bus communication to the sensor, read the current x,y,z axis values and compute the current roll and pitch.

The package includes source code for sample applications. This can be found at

- /iot/serial_sensor_read.c
- /iot/pthread_sensor_read.c

To build the example

We can build the application using the provided make file under the directory.

```
cd iot/
make
```

To transfer the binary to target

Using scp, command on the Ci40 board transfer the compiled binary from the host machine.

```
scp user@host:/path/to/iot/serial_read
scp user@host:/path/to/iot/pthread_read
```

As the names suggest. one program will read from the sensors serially, while the other does the same in a multi-threaded fashion.

To run the IoT example on target

Run as shown below:

```
#Read the temperature sensor followed by accelerometer
root@OpenWrt:/diskr# ./serial_read
Reading Temperatures
Average Temperature  27.189°C
Reading accelerometer values
Current roll and pitch -135.000 and 35.264
Time Taken serial execution(s):  1.280000


#Now read them both parallel schedule on CPUs 2 and 3
root@OpenWrt:/diskr# ./pthread_read
Scheduled Temperature Thread ID: 2001916736, CPU: 2
Reading Temperatures
Scheduled Accelerometer Thread ID: 1993528128, CPU: 3
Reading accelerometer values
Current roll and pitch -135.000 and 35.264
Time Taken accel thread execution(s):  0.530000
Average Temperature 27.065
Time Taken temperature thread execution(s):  0.910000

#Time taken by serial execution is 1.28s ,In parallel the time is about
0.91(higher of the two sensors)
#Thus CPU is idle for 0.37s more
```

The CPU while using multi-threading is 40 percent faster, with one thread idle for 0.75s and the other for 0.37s compared to single-threaded execution. This results in significant power savings compared to the single threaded execution. At the same time our throughput has improved significantly as well. This application can be extended to different combinations of sensors and actuators.

3. Conclusion

The Creator Ci40 is a great IoT development platform that supports connectivity tailored for IoT. It supports two sensors out of the box. It includes a modern MIPS interAptiv CPU, providing multi-Threading, a highly efficient feature that is not available from similarly classified, competing CPU IP cores. This feature provides increased CPU utilization, efficiency, performance and cd power savings vital for today's IoT applications.

Multi-threading resulted in a performance gain of 17 percent in the accumulator program, 15 percent more performance in CoreMark and gained 40 percent speedup in our dual-threaded IoT application. In our IoT application the speedup resulted in more CPU idle time, yielding power savings.

With support for multiple operating systems including OpenWrt and Brillo, the Creator Ci40 is a great development board for the MIPS architecture that fully enables the end user to prototype an IoT application with ease.

4. Where to find additional information

Purchasing a Creator Ci40 board

- http://www.mouser.co.uk/ProductDetail/Imagination-Technologies/VL-62913/

Download the source code and project that accompanies this whitepaper

- https://github.com/CreatorDev/Ci40-multithreading-whitepaper

More documentation on the Creator Ci40

- https://docs.creatordev.io/ci40/

Download the Codescape MIPS SDK

- https://community.imgtec.com/developers/mips/tools/codescape-mips-sdk/

Download MIPS Toolchains

- http://codescape-mips-sdk.imgtec.com/components/toolchain/2016.05-03/downloads.html

Feedback: Please join the forum discussion

- https://forum.creatordev.io/