FII RISC-V3.01 on FII-PRX100-S (ARTIX-7, XC7A100T) XILINX FPGA Board Coremark Porting Guide

V1.1

FRASER INNOVATION INC

Version Control

version	Date	Description
1.0	09/29/2020	Initial Release
1.1	10/06/2020	Add Comparison Figure and Full Description of PRX100

Copyright Notice:

Thank you for purchasing the FPGA development board. Please read the manual carefully before using the product and make sure that you know how to use the product correctly. Improper operation may damage the development board. This manual is constantly updated, and it is recommended that you download the latest version when using.

Official Shopping Website:

https://fpgamarketing.com/FII-PRX100-S-ARTIX-100T-XC7A100T-Xilinx-RISC-V-FPGA-Board-FII-PRX100-S-1.htm

Content

1. Introduction

Coremark has been EEMBC's CPU evaluation standard since 2009. EEMBC (Embedded Microprocessor Benchmark Consortium) is a non-profit organization with members including Huawei, Intel, ARM and Analog Devices. EEMBC is an important standard for evaluating embedded processors and compilers [1].

Coremark mainly detects ALU (Arithmetic Logic Unit), memory reference, pipeline and branch operations. It is designed to make it impossible for the CPU to run benchmark tests in advance, thus ensuring its fairness. During the specified test time, Coremark does not allow invoking third-party library, and the results are completely based on the optimization of the compiler and the execution processing time of the CPU. Because Coremark mainly provides testing of the CPU architecture, in order to abandon the superiority of the hardware manufacturing process, the final test results of Coremark will be normalized, that is to say, the final test results will be evenly divided into the system clock and the unit is Coremark/MHz. Coremark's main code is written in C language, including list processing (find and sort), matrix manipulation (common matrix operations), state machine (determine if an input stream contains valid numbers), and CRC (cyclic redundancy check) [2].

2. Porting

The first step is to download the c source directory from the EEMBC official website, or directly search for the EEMBC github (https://github.com/eembc/coremark). There will be 8 files in the source directory that needs to be copied to the project workspace (Here, FreedomStudio is used as the platform). They are as follows:

- core_list_join.c
- core_main.c
- core_matrix.c
- core_state.c
- core_util.c
- coremark.h
- core_portme.c
- core_portme.h

Only three of them need to be changed. These three files are "core_portme.h", "core_portme.c", and "coremark.h" (Use core_portme.h and core_portme.c under "simple" folder) . The others can just be added directly to the project.

2.1 Porting core_portme.h

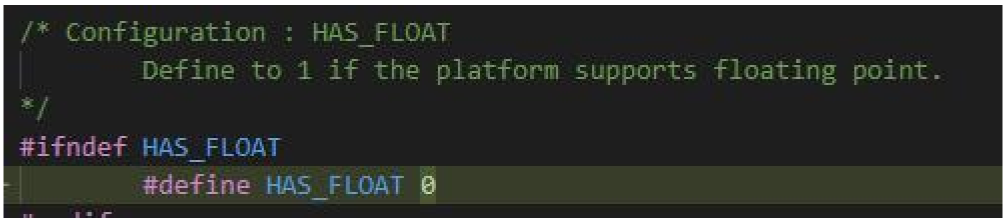
First of all, there are 14 macros in total. They are shown in Table 1 as follows:

Macro	Description
HAS_FLOAT	Define to 1 if platform supports floating point.
HAS_TIME_H	Define to 1 if platform has the time.h header file, and implementation of functions thereof.
USE_CLOCK	Define to 1 if platform has the time.h header file, and implementation of functions thereof.
HAS_STDIO	Define to 1 if the platform has stdio.h.
HAS_PRINTF	Define to 1 if the platform has stdio.h and implements the printf function.
COMPILER_VERSION	Put the compiler version here (e.g. GCC 7.2.0).
COMPILER_FLAGS	Put the compiler flags here (e.g. O3).
MEM_LOCATION	Put the memory location of code execution here (e.g. STACK).
CORTIMETYPE	Define type of return from the timing functions.

SEED_METHOD	Define the method to get seed values that cannot be computed at compile time.
MEM_METHOD	Define method to get a block of memory.
MULTITHREAD	Define for parallel execution.
MAIN_HAS_NOARGC	Needed if platform does not support getting arguments to main (This flag only matters if MULTITHREAD has been defined to a value greater then 1).
MAIN_HAS_NORETURN	Needed if platform does not support returning a value from main.

Table 1 Macros Description

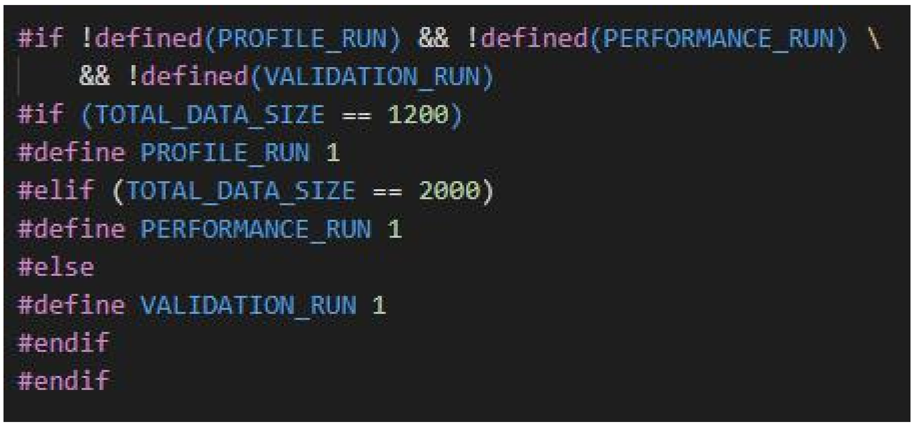
They should be modified and configured according to the porting system and platform. For example, HAS_FLOAT should be set to 0 if floating point operations are not supported. HAS_TIME_H and USE_CLOCK define whether the timer has the time.h header file and implementation of function thereof. time.h is imported to invoke the timer in the core_portme.c, which is then used in the main iteration loop to count the time. If the platform uses a different function to define time, it should be overwritten. HAS_PRINTF defines whether the platform uses the standard I/O library to print. It could be redefined according to the needs. MEM_LOCATION is very important, because it defines the location where the code is executed.



```
/* Configuration : HAS_FLOAT
        Define to 1 if the platform supports floating point.
*/
#ifndef HAS_FLOAT
        #define HAS_FLOAT 0
```

Figure 1 Configure HAS_FLOAT

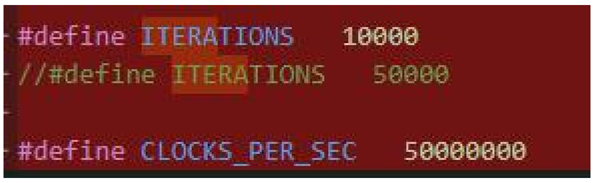
Besides, there is an execution mode could be configured. As shown in Figure 2, there are PROFILE_RUN, PERFORMANCE_RUN, and VALIDATION_RUN that could be selected. TOTAL_DATA_SIZE could be modified in coremark.h.



```
#if !defined(PROFILE_RUN) && !defined(PERFORMANCE_RUN) \
    && !defined(VALIDATION_RUN)
#if (TOTAL_DATA_SIZE == 1200)
#define PROFILE_RUN 1
#elif (TOTAL_DATA_SIZE == 2000)
#define PERFORMANCE_RUN 1
#else
#define VALIDATION_RUN 1
#endif
#endif
```

Figure 2 Configuration of the execution mode

Some parameters like ITERATIONS could be defined here as well, as shown in Figure 3.



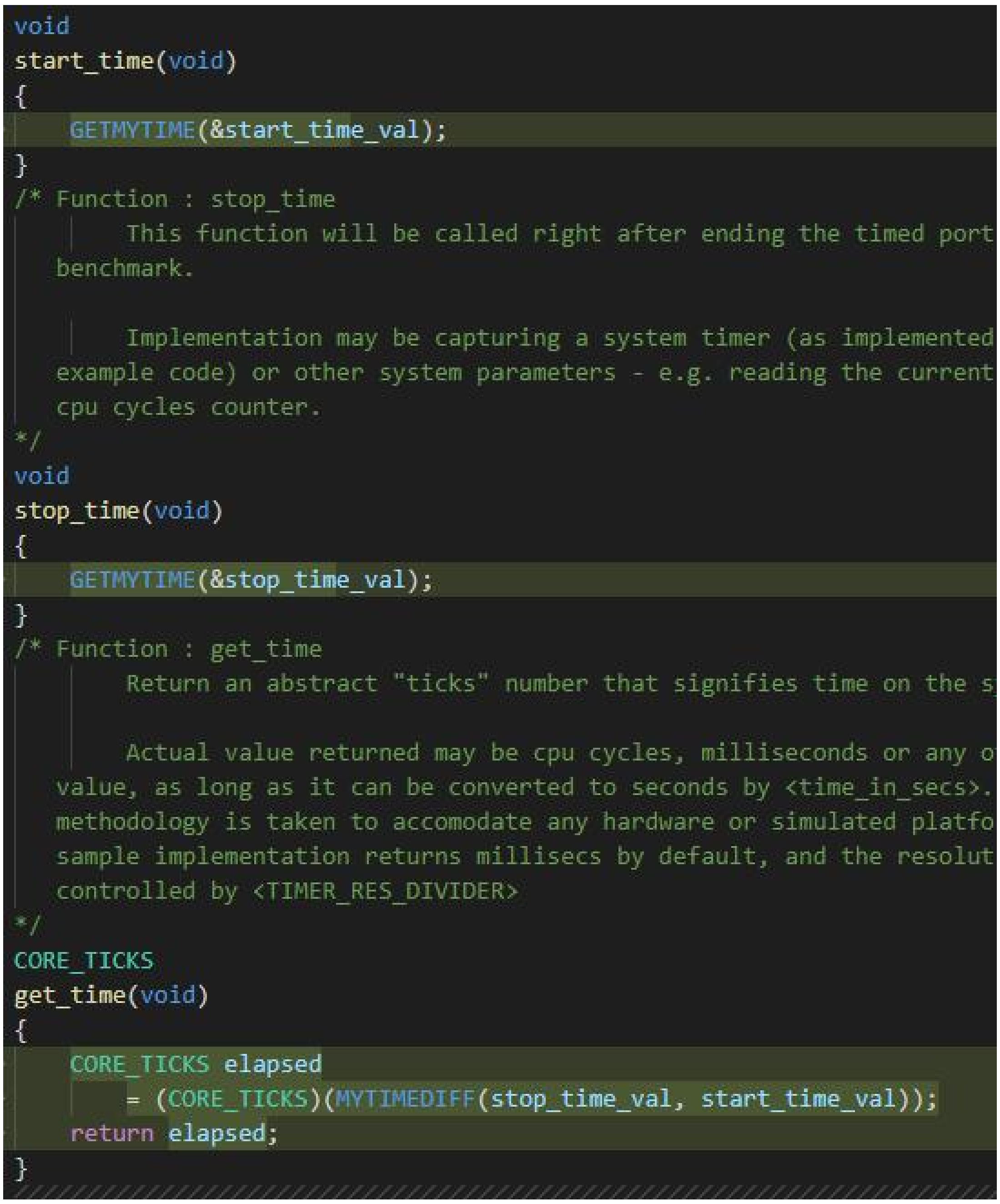
```
#define ITERATIONS    10000
//#define ITERATIONS    50000

#define CLOCKS_PER_SEC    50000000
```

Figure 3 Parameter configuration

Last but not least, the corresponding libraries should be modified with respect to the configurations.

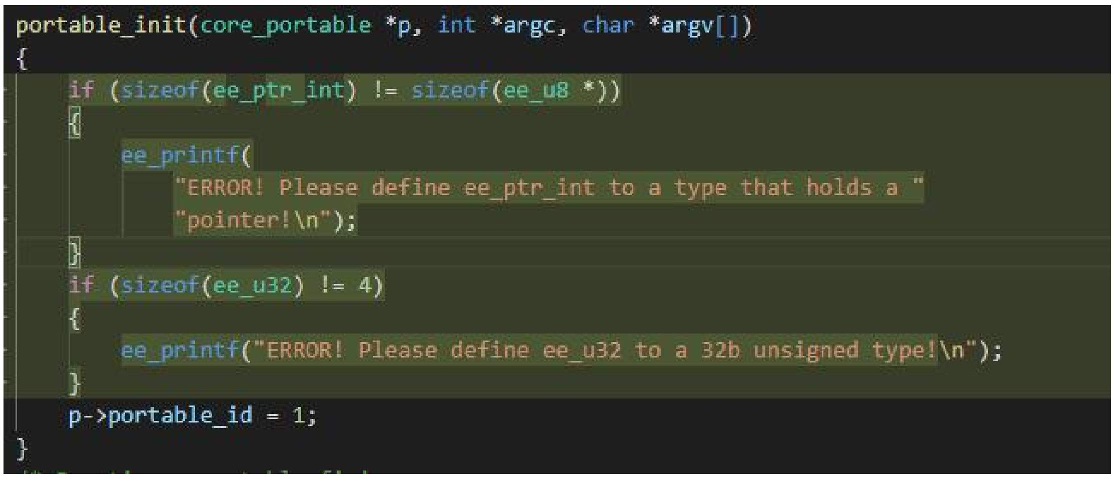
2.2 Porting core_portme.c

Parameter "EE_TICKS_PER_SEC" is the total ticks for every second, which is related to the system clock. It should be modified accordingly. The most important functions are related to timer, which are "start_time","stop_time", and "get_time". As mentioned above, the time-related function could be modified as needed. See Figure 4 for the details of time-related functions. In the main of core_main.c. The first ever function being called is "portable_init". The initialization and debug print information could be implemented there as shown in Figure 5. Note that the libraries should also be correspondingly added.



```
void
start_time(void)
{
    GETMYTIME(&start_time_val);
}
/* Function : stop_time
        This function will be called right after ending the timed port
    benchmark.

        Implementation may be capturing a system timer (as implemented
    example code) or other system parameters - e.g. reading the current
    cpu cycles counter.
*/
void
stop_time(void)
{
    GETMYTIME(&stop_time_val);
}
/* Function : get_time
        Return an abstract "ticks" number that signifies time on the s

        Actual value returned may be cpu cycles, milliseconds or any o
    value, as long as it can be converted to seconds by <time_in_secs>.
    methodology is taken to accomodate any hardware or simulated platfo
    sample implementation returns millisecs by default, and the resolut
    controlled by <TIMER_RES_DIVIDER>
*/
CORE_TICKS
get_time(void)
{
    CORE_TICKS elapsed
        = (CORE_TICKS)(MYTIMEDIFF(stop_time_val, start_time_val));
    return elapsed;
}
```

Figure 4 Time-related functions

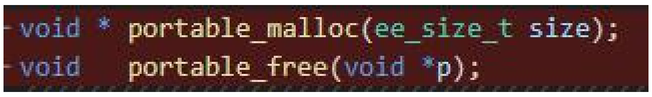


```
portable_init(core_portable *p, int *argc, char *argv[])
{
    if (sizeof(ee_ptr_int) != sizeof(ee_u8 *))
    {
        ee_printf(
            "ERROR! Please define ee_ptr_int to a type that holds a "
            "pointer!\n");
    }
    if (sizeof(ee_u32) != 4)
    {
        ee_printf("ERROR! Please define ee_u32 to a 32b unsigned type!\n");
    }
    p->portable_id = 1;
}
```

Figure 5 Initialization function

2.3 Porting coremark.h

The memory definition such as “malloc” and “free” could be modified here as shown in Figure 6.

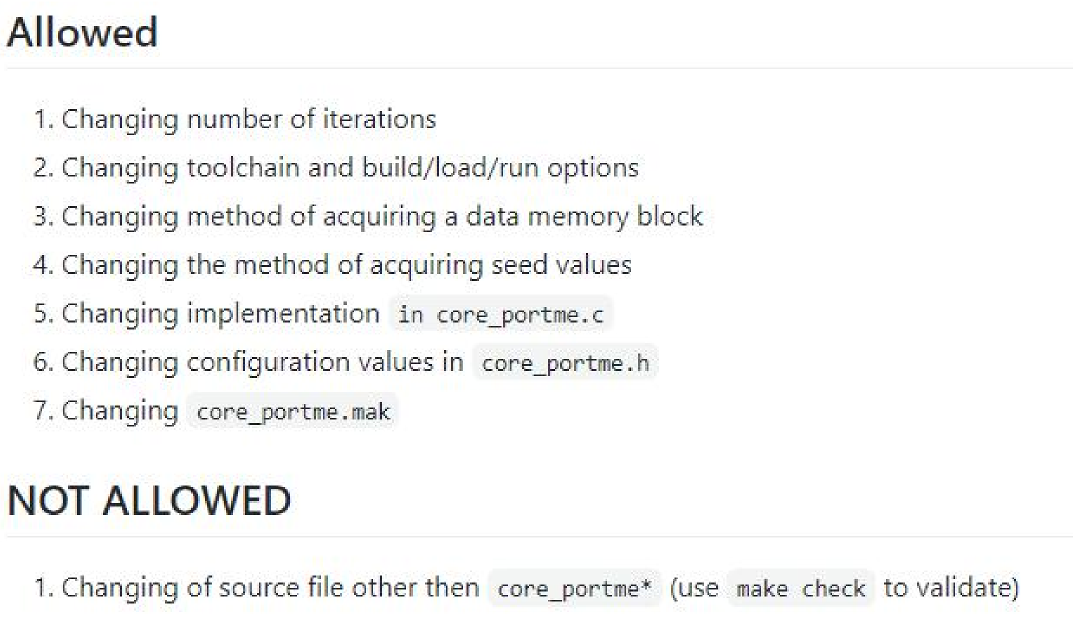


```
void * portable_malloc(ee_size_t size);
void   portable_free(void *p);
```

Figure 6 Memory-related function

2.4 Other Points

The whole program is required to run for at least 10 seconds. The iteration times could be modified accordingly. Theoretically, the more iterations it runs, the more accurate the result will be. Also, README.md is included in the directory. See attached files for more information. For instance, as shown in Figure 7, it lists some rules ensuring the Coremark result is valid.



Allowed

1. Changing number of iterations
2. Changing toolchain and build/load/run options
3. Changing method of acquiring a data memory block
4. Changing the method of acquiring seed values
5. Changing implementation `in core_portme.c`
6. Changing configuration values in `core_portme.h`
7. Changing `core_portme.mak`

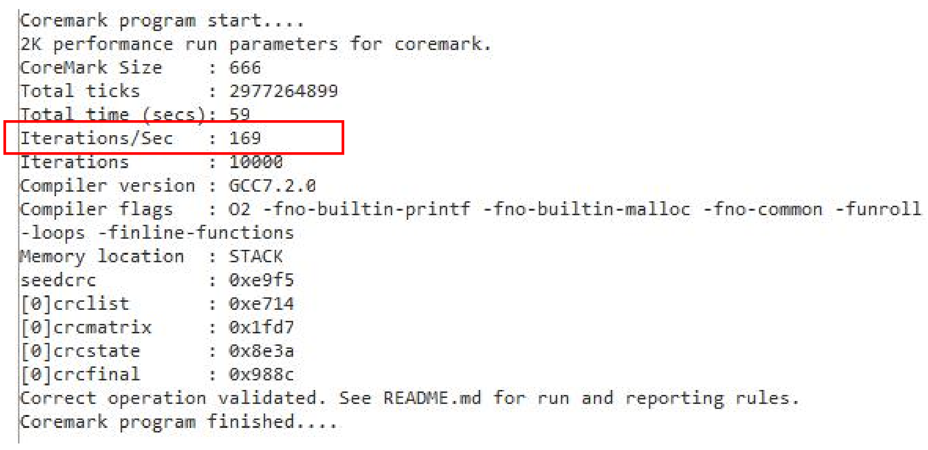
NOT ALLOWED

1. Changing of source file other then `core_portme*` (use `make check` to validate)

Figure 7 README.md Information

3. Test Result Evaluation

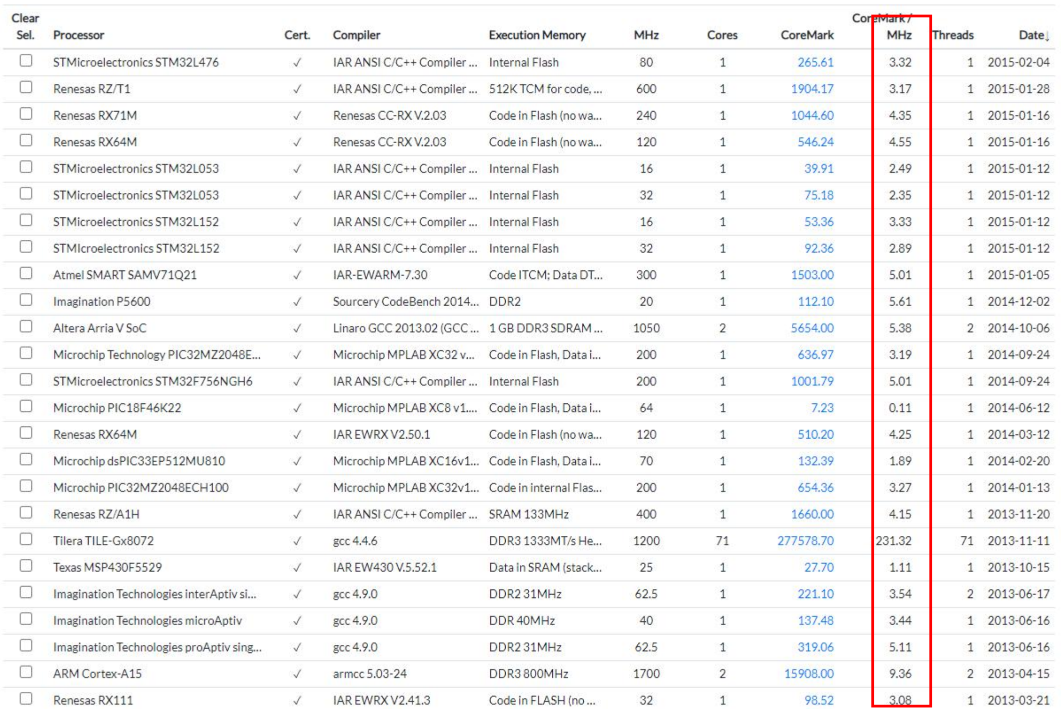
The FII RISC-V3.01 on FII-PRX100-S (ARTIX-7, XC7A100T) XILINX FPGA Board (https://fpgamarketing.com/FII-PRX100-S-ARTIX-100T-XC7A100T-Xilinx-RISC-V-FPGA-Board-FII-PRX100-S-1.htm) system clock is 50MHz, and the Coremark test score shown in Figure 8 is 3.38 (169/50 Coremark/MHz).



```
Coremark program start....
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 2977264899
Total time (secs): 59
Iterations/Sec   : 169
Iterations       : 10000
Compiler version : GCC7.2.0
Compiler flags   : O2 -fno-builtin-printf -fno-builtin-malloc -fno-common -funroll
-loops -finline-functions
Memory location  : STACK
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Correct operation validated. See README.md for run and reporting rules.
Coremark program finished....
```

Figure 8 FII RISC-V3.01 Coremark

Figure 9 is a screenshot of the CPU Coremarks provided on the EEMBC website with certification.



Clear Sel.	Processor	Cert.	Compiler	Execution Memory	MHz	Cores	CoreMark	CoreMark / MHz	Threads	Date↓
☐	STMicroelectronics STM32L476	✓	IAR ANSI C/C++ Compiler ...	Internal Flash	80	1	265.61	3.32	1	2015-02-04
☐	Renesas RZ/T1	✓	IAR ANSI C/C++ Compiler ...	512K TCM for code, ...	600	1	1904.17	3.17	1	2015-01-28
☐	Renesas RX71M	✓	Renesas CC-RX V.2.03	Code in Flash (no wa...	240	1	1044.60	4.35	1	2015-01-16
☐	Renesas RX64M	✓	Renesas CC-RX V.2.03	Code in Flash (no wa...	120	1	546.24	4.55	1	2015-01-16
☐	STMicroelectronics STM32L053	✓	IAR ANSI C/C++ Compiler ...	Internal Flash	16	1	39.91	2.49	1	2015-01-12
☐	STMicroelectronics STM32L053	✓	IAR ANSI C/C++ Compiler ...	Internal Flash	32	1	75.18	2.35	1	2015-01-12
☐	STMicroelectronics STM32L152	✓	IAR ANSI C/C++ Compiler ...	Internal Flash	16	1	53.36	3.33	1	2015-01-12
☐	STMIcroelectronics STM32L152	✓	IAR ANSI C/C++ Compiler ...	Internal Flash	32	1	92.36	2.89	1	2015-01-12
☐	Atmel SMART SAMV71Q21	✓	IAR-EWARM-7.30	Code ITCM; Data DT...	300	1	1503.00	5.01	1	2015-01-05
☐	Imagination P5600	✓	Sourcery CodeBench 2014...	DDR2	20	1	112.10	5.61	1	2014-12-02
☐	Altera Arria V SoC	✓	Linaro GCC 2013.02 (GCC ...	1 GB DDR3 SDRAM ...	1050	2	5654.00	5.38	2	2014-10-06
☐	Microchip Technology PIC32MZ2048E...	✓	Microchip MPLAB XC32 v...	Code in Flash, Data i...	200	1	636.97	3.19	1	2014-09-24
☐	STMicroelectronics STM32F756NGH6	✓	IAR ANSI C/C++ Compiler ...	Internal Flash	200	1	1001.79	5.01	1	2014-09-24
☐	Microchip PIC18F46K22	✓	Microchip MPLAB XC8 v1....	Code in Flash, Data i...	64	1	7.23	0.11	1	2014-06-12
☐	Renesas RX64M	✓	IAR EWRX V2.50.1	Code in Flash (no wa...	120	1	510.20	4.25	1	2014-03-12
☐	Microchip dsPIC33EP512MU810	✓	Microchip MPLAB XC16v1...	Code in Flash, Data i...	70	1	132.39	1.89	1	2014-02-20
☐	Microchip PIC32MZ2048ECH100	✓	Microchip MPLAB XC32v1...	Code in internal Flas...	200	1	654.36	3.27	1	2014-01-13
☐	Renesas RZ/A1H	✓	IAR ANSI C/C++ Compiler ...	SRAM 133MHz	400	1	1660.00	4.15	1	2013-11-20
☐	Tilera TILE-Gx8072	✓	gcc 4.4.6	DDR3 1333MT/s He...	1200	71	277578.70	231.32	71	2013-11-11
☐	Texas MSP430F5529	✓	IAR EW430 V.5.52.1	Data in SRAM (stack...	25	1	27.70	1.11	1	2013-10-15
☐	Imagination Technologies interAptiv si...	✓	gcc 4.9.0	DDR2 31MHz	62.5	1	221.10	3.54	2	2013-06-17
☐	Imagination Technologies microAptiv	✓	gcc 4.9.0	DDR 40MHz	40	1	137.48	3.44	1	2013-06-16
☐	Imagination Technologies proAptiv sing...	✓	gcc 4.9.0	DDR2 31MHz	62.5	1	319.06	5.11	1	2013-06-16
☐	ARM Cortex-A15	✓	armcc 5.03-24	DDR3 800MHz	1700	2	15908.00	9.36	2	2013-04-15
☐	Renesas RX111	✓	IAR EWRX V2.41.3	Code in FLASH (no ...	32	1	98.52	3.08	1	2013-03-21

Figure 9 Part of the CPU Coremark result of EEMBC

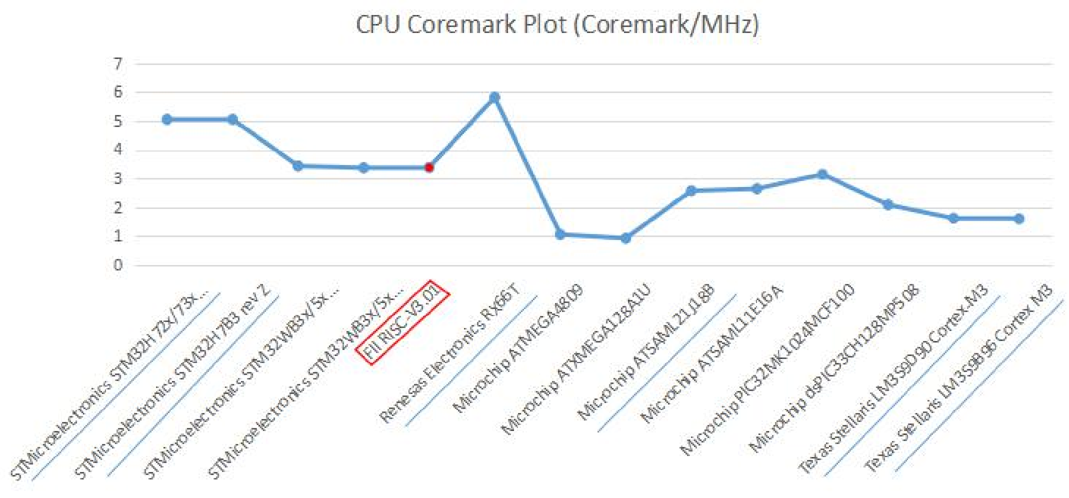


Figure 10 CPU Coremark Comparison

FII RISC-V3.01 is a single-core, a mix of 2-stage and 3-stage pipeline CPU. Figure 10 lists some other single-core CPUs' Coremark being certified by EEMBC. FII RISC-V3.01 has been highlighted using red strokes. It can be seen that FII RISC-V3.01 Coremark is above the average Coremark among the listed 15 CPUs. For the three CPUs which have obviously higher Coremark, they are STMicroelectronics STM32H72x/73x rev Z (highlight as blue), STMicroelectronics STM32H7B3 rev Z (highlight as blue) and Renesas Electronics RX66T (highlight as blue). From the official manual by STMicroelectronics , STM32H72x/73x rev Z and STMicroelectronics STM32H7B3 rev Z both use Cortex-M7, which has a 6-stage super scalar pipeline. Renesas Electronics RX66T uses RXv3 core, which has improved 5-stage pipeline. Since with more stages of pipeline, undoubtedly the better performance of CPU is, to make the comparison of performance more fair, compared with Texas Stellaris Cortex-M3 (highlight as blue), which is also a

3-pipeline processor as FII RISC-V3.01. Nevertheless, FII RISC-V's Coremark is greatly larger than theirs, even more than two times. Compared with another processor, Microchip ATSAML21J18B (highlight as blue), which is also a three-stage pipeline, the Coremark of FII RISC-V3.01 is still much higher.To conclude, with the same amount core stages of pipeline constrained, FII-RISCV3.01 performs outstandingly and is favourable.

4. References

[1] "EEMBC | Wikiwand", *Wikiwand*, 2020. [Online]. Available:

https://www.wikiwand.com/en/EEMBC. [Accessed: 29- Sep- 2020].

[2] "EEMBC", *Eembc.org*, 2020. [Online]. Available:

https://www.eembc.org/coremark/index.php. [Accessed: 29- Sep- 2020].