

Multicore development environment for embedded processor in arduino IDE

Stefanus Kurniawan¹, Dareen K. Halim², Dicky H.³, Tang C. M.⁴

^{1,2}Universitas Multimedia Nusantara, Indonesia

^{3,4}Universiti Tunku Abdul Rahman, Malaysia

Article Info

Article history:

Received Jul 9, 2019

Revised Jan 11, 2020

Accepted Feb 21, 2020

Keywords:

Arduino IDE

Development environment

RUMPS401

IoT

MPSoC

ABSTRACT

Internet of things (IoT) technology has found more applications that require complex computation while still preserving power. Embedded processors as the core of the IoT system approaches the need for computation by employing a parallel processor system, namely MPSoC. While various MPSoCs hardware is widely available, there is limited software support form of user-friendly libraries and development platform. There is a need for such a platform to facilitate both the study and development of parallel embedded software. arduino as the widely used embedded development platform is yet to officially support multicore programming. This work proposes an arduino-based development environment that supports multicore programming while maintaining arduino's simple program structure, targeted at specific low-power MPSoC, the RUMPS401. The environment is fully functional, and while it targets only specific MPSoC, the proposed environment can easily be adopted to other MPSoCs with similar structures with minimal modification.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Dareen K. Halim,

Department of Computer Engineering,

Universitas Multimedia Nusantara,

Jl. Scientia Boulevard, Gading, Serpong, Tangerang, Banten 15227, Indonesia.

Email: dareen.halim@umn.ac.id

1. INTRODUCTION

The rise of internet of things (IoT) technology has led to adoption in more complex applications, such as medical, robotics, sports, disaster control [1-6]. This demands for hardware capable for complex computations while still maintaining the low-power characteristic of IoT [7, 8]. Parallel processor system has been the widely accepted approach to increase computational capability, implemented in almost every personal computer. Embedded processor as one of the crucial components in IoT adopts the same approach in the form of multi-processor system-on-chip (MPSoC) [9], a single integrated circuit containing multiple processors and other elements such as input-output (IO) and memory [10]. MPSoCs has been widely developed and used in various applications such as LTE modem [11], ECG analysis [12], image processing [13, 14]. One example of a low-power MPSoC is the RUMPS401. It is powered by four ARM Cortex-M0 based cores, connected internally via adaptive NoC [15].

Despite the number of MPSoCs available, parallel programming in embedded system is still uncommon due to limited support in the form of user-friendly libraries and development platform. Several works attempt to support parallel programming by providing frameworks [16, 17], parallel pipeline extraction [18], as well as teaching platform [19]. However, most of these platforms are rather specific in terms of their applications, languages, or program structures. A more general platform is hence desired to facilitate

both study and development of parallel embedded software. Widely known open-source platform for such purpose is arduino [20]. It manufactures ready-to-use development boards for various embedded processors hence providing hardware abstraction, while also defining user-friendly program structure along with numerous function libraries. Per now, to our best knowledge, arduino's official program structure is limited to single-core processor.

This work proposes an arduino-based development environment for embedded MPSoC as an approach to facilitating study and development of parallel embedded software. Rather than building the environment from scratch, the arduino approach averts the exclusivity problem observed in other related works. The environment is limited to Windows operating system as the work relies on windows batch script specific. Porting is needed to allow operation in other operating systems. The targeted MPSoC is the RUMPS401 due to its interesting multicore and low-power characteristic which is appealing to IoT application.

The rest of this paper is arranged as follows. Section 2 describes the research methodology, divided into few parts: RUMPS401 program loading process and its existing toolchain; arduino's program structure and compilation process; and the proposed development environment. The resulting environment is then tested for its functionality, whose result is discussed in section 3. The last section concludes this paper.

2. RESEARCH METHOD

This work aims to develop an arduino-based environment that allows multicore programming, compilation, and uploading into the target MPSoC, the RUMPS401. This section hence describes the method used in this work, divided into four parts, i.e. understanding the RUMPS401 programming environment, understanding the arduino IDE internal workflow, integration of the RUMPS401 programming environment into arduino IDE, and the testing method.

2.1. RUMPS401 programming support

The RUMPS401 is fully designed by Universiti Tunku Abdul Rahman's (UTAR) VLSI research center, emphasizing on low-power consumption [21]. Each individual core is a fully functional microcontroller with their own program, equipped with number of peripherals such as IOs, Flash, SRAM, timer. Cooperation between cores is also facilitated by the internal NoC. Each core can be put to sleep and woken up via NoC or external signal. The RUMPS401 current consumption is around 30mA with all cores running, and around 13uA when all cores are put to sleep [21]. It provides program uploading via scalable bootloader design which utilizes the in-chip hardware bootloader and software bootloader for distributing programs to individual cores via NoC [22]. One of the RUMPS401 core, the IO Core [23] is designed to run in two modes, the first for aiding the bootloading process, and second to run its normal functionality. The IO Core has its flash memory split into two sectors, one for storing the software bootloader program and the other for storing its normal program.

Uploading into the RUMPS401 is a two-steps process. First, the chip is put into hardware bootloading mode to allow its in-chip bootloader to receive software bootload program via specialized UART pins and save it into the software bootloader sector of the IO core's flash memory. The chip then is run in software bootloading mode, in which the IO Core executes the software bootloader program that receives program or firmware for each core via UART, and distributes those programs accordingly including to itself through NoC. Both bootload steps also define specific protocols to be followed by the uploading side. Should the upload succeed, the RUMPS401 can be started in normal operation mode to run its firmware on each core.

In this work, the RUMPS401 has been initialized by the designer team with the hardware bootloading process hence the arduino IDE only needs to perform the second upload step. Furthermore, the python-based uploader required to communicate with the software bootloader program has been provided as well. It takes four executables (one each core) compiled by the GNU embedded toolchain for ARM [24] as input and sends them to the software bootloader program run by the IO Core. This process is illustrated in Figure 1 [23]. On crucial aspect of this work is thus to integrate the RUMPS401 currently available upload process into the arduino environment.

2.2. Arduino IDE workflow

On the user-facing end, arduino IDE provides a simple program structure called sketch with ino file extension which includes only two functions, a setup function which is run once at the beginning, and a loop function which is run infinitely until the device is power cycled. On the background, the ino file will undergo several steps before yielding the final executable file. Specifically, the arduino IDE compilation process is divided into four steps, pre-processing, compile, linking, and executable extraction. This whole process is performed by arduino builder program built-in into the arduino IDE. During pre-processing, the arduino builder creates a new C++ file which basically is the ino sketch file with its extension changed to cpp, and line

directives as well as Arduino.h library include line added to the file content. The result is a valid C++ file containing definition of setup and loop function.

Proceeding to the compiling step, there are three C++ file groups, each compiled by the arduino builder into an intermediate object file. The first one is the C++ sketch file produced by the pre-processing step. The second group is files located in the arduino library folder, which contains built-in arduino libraries as well as user-added libraries. The last group is the files inside arduino core folder, which contains primary functions and definitions required by the compiled program, including the main.cpp file containing main function definition. In short, the main function calls the setup function once, and calls the loop function inside an infinite while loop.

The resulting object files are then taken by the linking step to produce an archive file containing all the object files, from which the final executable will be extracted. During this step, the setup and loop function defined in the C++ sketch file is linked with the setup and loop function prototype defined in the main file. The archive file also functions similarly to cache for the next compilation process, allowing the compiler to compile only recently changed files, hence providing speed up on consequent compilations. The final step performed by arduino builder is extracting binary executable file from the archive, which can be in hex or binary format depending on the target hardware. For simplicity, the terms executable and binary are interchangeable throughout this paper.

While sketch file pre-processing is performed by the arduino builder itself, the compile, linking, and extraction steps are performed by calling external tools such as GCC. The tools as well as their parameter are fully configurable through the arduino's hardware configuration file. This approach ensures modularity of the arduino IDE in supporting its wide range of official boards, as well as providing means for adding third party hardware vendors into the arduino environment. Along with the configurable toolchains, arduino builder also provides several break points called hooks between the compilation process where user can insert scripts, allowing modification towards the process. In total, there are twelve hooks available throughout the compilation [25]. This work utilizes only two hooks, one before sketch compilation and one after the binary file extraction, which will be detailed further in the subsequent section describing the proposed modification to the arduino IDE compilation.

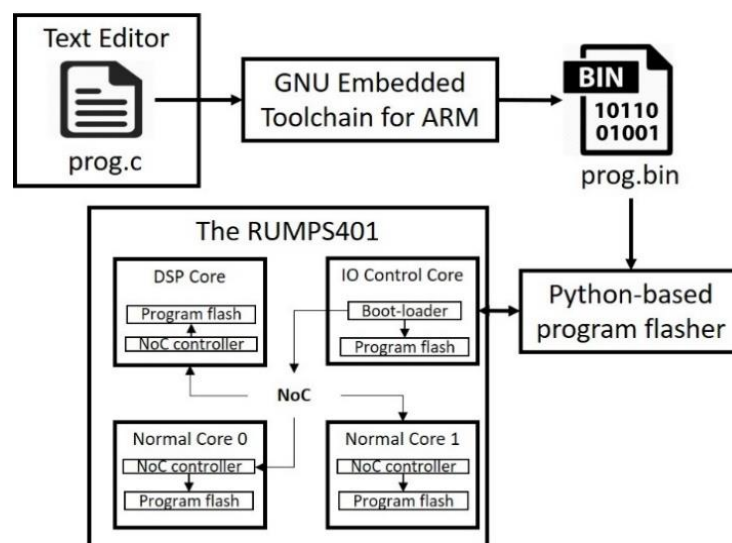


Figure 1. The RUMPS401 program upload process

2.3. Proposed development environment

Based on details discussed by the previous sections, this work derived a method for providing arduino-based multicore development environment by integrating RUMPS401 multicore programming toolchain into the arduino IDE environment. The proposed method comprises of three parts which are modification of the ino file program structure, integration of RUMPS401 existing toolchain into arduino builder, and hooks utilization to alter the compilation process.

To provide multicore programming support while still maintaining the simplicity of arduino program structure, this work proposes a modified program structure containing four pairs of setup loop functions as

shown in Figure 2. Those functions are located within a single ino file, with each pair describing the program for each core. As discussed in the previous section, arduino builder links the setup and loop function definition in the sketch file to their respective prototypes defined in the main function located in arduino cores folder. To incorporate the four pairs of setup and loop functions, four C++ files are created, each containing identical main function definition except for the setup and loop functions prototype, shown in Figure 3. During compilation and linking these files will be used alternately to produce executables for each core, which will be detailed further in the hook utilization part.

```

//----- //----- //----- //-----
// Core 0 // Core 1 // Core 2 // Core 3
//----- //----- //----- //-----
void setup() { void setup1() { void setup2() { void setup3() {

} } } }

void loop() { void loop1() { void loop2() { void loop3() {

} } } }

```

Figure 2. Proposed multicore ino program structure

Figure 3. Proposed copies of main functions

As described in previous section, the arduino builder utilizes external tools for compilation, linking, and executable extraction, which are modifiable through a simple configuration file. Since there is already an existing toolchain for the RUMPS401, modification can be achieved by simply changing the configuration file to target the RUMPS401 existing toolchains instead of arduino's default tools. Arduino configuration file is defined in a key-value pair form, hence modification can be made by simply changing the values to specific tools along with their parameters. The same also applies for uploading executable as it is performed by an external tool, which in this work is the Python-based uploader shown in Figure 1.

Two Windows batch scripts are written and executed using the arduino hooks to alter the compilation process, one before sketch compilation and the other after binary file extraction. For convenience, these scripts will be referred as pre-sketch and post-obj scripts. In general, these scripts control the compilation process by running the arduino Builder four times (once each core) while alternating between the four main files prepared earlier, and collecting binaries resulted from each run before uploading them into the RUMPS401. Figure 4 depicts the summarized compilation timeline.

In specific, the pre-sketch script performs two tasks that creates and uses text files for logging and control throughout the four compilation iterations. The first task initiates two files, one for marking compilation start time, and the other as compilation counter which is initiated to zero. These files are created only on the first iteration. The second task done by pre-sketch script is to replace the main.cpp file in arduino cores folder with one of the four main [num].cpp files created earlier, where num refers to the respective iteration count. This arrangement allows pre-sketch script to compile the proper setup and loop functions for each core.

The post-obj script is executed after binary extraction for each core. By default, the binary file is located at a temporary arduino build folder. Like the pre-sketch script, this script first initiates an additional text file required for logging and control, containing the arduino Builder parameters used in the current iteration. The script then checks the compilation counter and acts accordingly. If the counter is less than three (core is indexed by number zero to three), it will move the current executable to another temporary folder, increment the counter, then execute arduino builder again with the same parameters which in effect compiles

for the next core. Once the counter reaches three (which means the program for the last core is compiled), the script will collect the four executables produced from each iteration and move them to the default arduino build folder. Lastly, the post-obj script also calculates the compilation time by subtracting the compilation start time from the current timestamp. The resulting binaries are then uploaded to the RUMPS401 by the python-based uploader specified in arduino configuration file. It should be noted that the arduino builder parameters are kept the same throughout the iterative process to allow the compiler to use previously compiled files, if possible. Even with all the source C++ files remain unchanged, arduino Builder will automatically re-compiles everything if the compilation parameters differ.

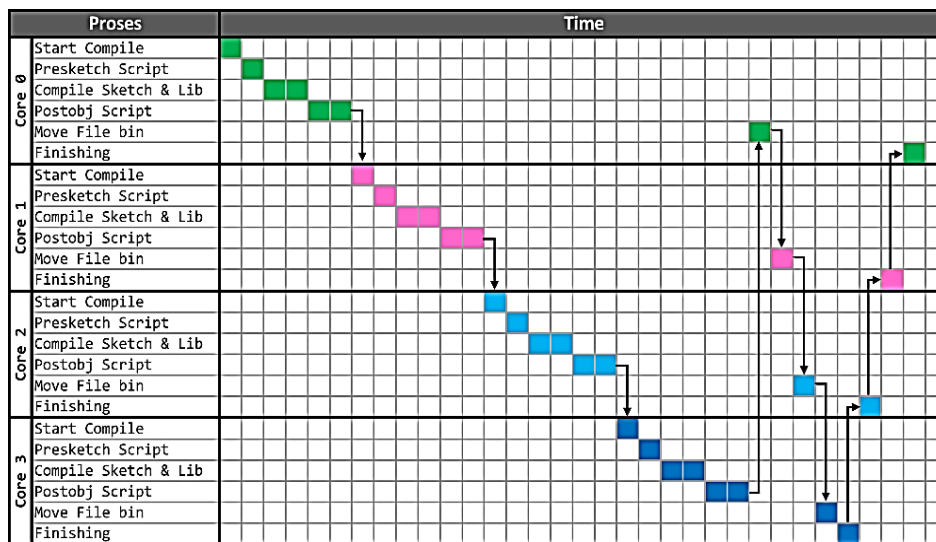


Figure 4. RUMPS401-arduino compilation process timeline

2.4. Testing and analysis method

Post implementation, the developed environment will first be tested for its functionality by writing simple programs in the proposed ino structure, compiling the programs and uploading the resulting binaries into the RUMPS401. The program includes basic functionalities such as GPIO & serial access, and delay. Nonetheless, the content of the program is not significant here as the compilation time is the main interest here. The same program will be compiled 20 times to measure its average compilation time. Measurements towards the compilation time will be performed by the same pre-sketch and post-obj scripts used to alter the compilation process, as explained earlier. While performing the test, the machine will be kept as constant as possible, i.e. with constant cooling and background applications interference kept to the minimum. The only difference among the 20 compilations is that the first compilation will be performed with a clean slate, i.e. no pre-compiled files, while the rest are carried without cleaning the pre-compiled files.

3. RESULTS AND ANALYSIS

The proposed environment is developed and tested on an Asus laptop with Intel i7-6700 processor and 8GB of RAM running windows 10 home edition. The arduino IDE being used throughout the work is version 1.8.9. Visual Studio Code is used as a general text editor for modifying the scripts and arduino configuration file. A simple button-LED blink test program is used throughout the functionality and compilation speed test, which snippet can be found in Figure 5. As with other arduino-compatible boards, arduino basic functions such as ones used in the test code are ported to the target hardware, the RUMPS401. The development of those functions is not discussed in this paper.

Figure 6 shows the modified content of arduino's hardware folder, which adheres to the documentation specified on the arduino's official github page [25]. The key components in this folder are the files named precompile_script.bat and postcompile_script.bat which contain the pre-sketch and post-obj script discussed in previous section, respectively. The platform.txt file contains the configuration regarding set of external tools used by arduino builder for compiling, linking, binary extraction, and upload, which in this case is the RUMPS401 toolchain.

```

#define button0 19
#define button1 34
#define button2 42
#define button3 50

#define LED0 18
#define LED1 32
#define LED2 40
#define LED3 48

void setup() {
  serial_begin(115200);
  pinMode(button0, INPUT);
  pinMode(LED0, OUTPUT);
}

void loop() {
  if(digitalRead(button0)){
    digitalWrite(LED0, HIGH);
    delay(3000);
    digitalWrite(LED0, LOW);
    delayMicroseconds(3000000);
    serial_println("Process Core 0");
  }
  else
  {
    digitalWrite(LED0, LOW);
  }
}

void setup1() {
  pinMode(button1, INPUT);
  pinMode(LED1, OUTPUT);
}

void loop1() {
  if(digitalRead(button1)){
    digitalWrite(LED1, HIGH);
    delay(500);
    digitalWrite(LED1, LOW);
    delayMicroseconds(500000);
  }
  else
  {
    digitalWrite(LED1, LOW);
  }
}

void setup2() {
  pinMode(button2, INPUT);
  pinMode(LED2, OUTPUT);
}

void loop2() {
  if(digitalRead(button2)){
    digitalWrite(LED2, HIGH);
    delay(1000);
    digitalWrite(LED2, LOW);
    delayMicroseconds(1000000);
  }
  else
  {
    digitalWrite(LED2, LOW);
  }
}

void setup3() {
  pinMode(button3, INPUT);
  pinMode(LED3, OUTPUT);
}

void loop3() {
  if(digitalRead(button3)){
    digitalWrite(LED3, HIGH);
    delay(200);
    digitalWrite(LED3, LOW);
    delayMicroseconds(200000);
  }
  else
  {
    digitalWrite(LED3, LOW);
  }
}

```

Figure 5. Multicore test program

tools	3/20/2019 6:03 PM	File folder	
variants	3/20/2019 6:03 PM	File folder	
boards.txt	5/17/2018 3:21 PM	TXT File	1 KB
platform.txt	6/27/2019 11:49 PM	TXT File	12 KB
postcompile_script.bat	7/10/2019 2:35 PM	Windows Batch File	4 KB
precompile_script.bat	6/30/2019 3:52 PM	Windows Batch File	1 KB
programmers.txt	11/4/2015 1:05 AM	TXT File	0 KB

Figure 6. Partial content of modified hardware folder

As discussed earlier, the batch scripts use several text files for logging and controlling the compilation process. Figure 7 shows the content of parameters file, counter file, and timestamp file. These files are removed after every successful four-core compilation and re-initialized on every subsequent compilation. The resulting binaries are named main[num].bin where num refers to the core number, and grouped in a folder as shown in Figure 8. These binaries are then uploaded into the RUMPS401 and tested, where the program run as expected.

```

rumps_compile_param.txt
1 ide.hardware.path=C:\Program Files (x86)\Arduino\hardware
2 local.arduino.packages=C:\Users\Stefanus K\AppData\Local\Arduino15\packages
3 rumpsduino1_0.hardware.path=C:\Users\Stefanus
  K\Documents\Arduino\Program\hardware\rumps\..
4 ide.tools.builder=C:\Program Files (x86)\Arduino\tools-builder
5 ide.tools.avr=C:\Program Files (x86)\Arduino\hardware\tools\avr
6 ide.libraries=C:\Program Files (x86)\Arduino\libraries
7 sketchbook.libraries.path=C:\Users\Stefanus
  K\Documents\Arduino\Program\hardware\rumps\...\libraries
8 build.fqbn=rumps:1.1:alpha
9 ide.version=10809
10 ide.build.path=C:\Users\STEFAN~1\AppData\Local\Temp\arduino_build_501411
11 build.warn_data_percentage=75
12 source.path=C:\Users\Stefanus K\Documents\Arduino\Program\DemosCode
13 source.name=DemosCode.ino
14 arduino.builder.path=C:\Program Files (x86)\Arduino
15

rumps_runlog.txt
1 core_count=0
2

rumps_exetime.txt
1 start_time=14:31:03.14
2

```

Figure 7. Logging and control files for multicore compilation





 main0.bin	7/10/2019 1:57 PM	BIN File	2 KB
 main1.bin	7/10/2019 1:57 PM	BIN File	1 KB
 main2.bin	7/10/2019 1:57 PM	BIN File	1 KB
 main3.bin	7/10/2019 1:57 PM	BIN File	1 KB

Figure 8. Resulting binaries

Upon passing the functionality test, the same program is compiled for 20 times to measure its average compilation time. The result is shown in Figure 9, where the compilation time averages at 12.1 seconds. The highest compilation time is observable on the first compilation attempt. This is expected as the first compilation attempt has no existing archive hence must compile all of the source files, whereas the consequent compilations utilize the archive file created by the first compilation. On the other hand, the rather high compilation time exhibited by tenth and sixteenth attempts should be caused externally by the fluctuating variations in CPU usage. This is deduced as the operating system or other background programs might be functioning during those tests, and it is observed as well that there is no difference at all in the compilation log files.

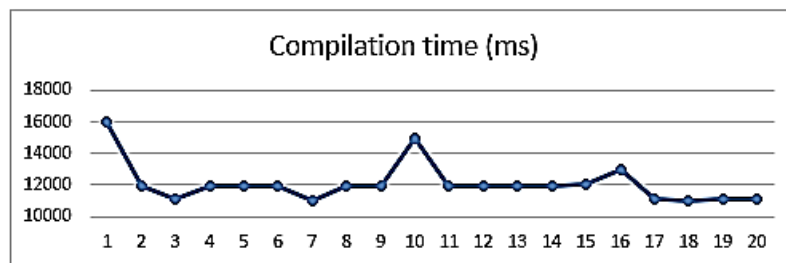


Figure 9. Result of compilation time test

4. CONCLUSION

This work has shown a fully functional arduino-based multicore development environment, targeted for a specific low-power MPSoC, the RUMPS401. With simple button & LED blink codes as test programs, the proposed environment compiles the four programs in 12.1 seconds in average. The proposed environment was built by leveraging the arduino support for third-party vendors, in which external compiler toolchain can be used. Support for multiple cores is provided by running the default arduino builder multiple times with different main.cpp file containing different setup and loop function pairs. The result is a single ino program structure with four pairs of setup and loop function pertaining similarity to arduino program structure. Currently, the environment can only run in Windows operating system as it depends on batch scripting. Nonetheless, the proposed environment can easily be adopted to other MPSoCs with similar structures with minimal modification. Future works may incorporate artificial intelligence into the compiler, allowing the IDE to split user's code which is a single program into parallel programs.

REFERENCES

- [1] S. Wang, Y. Hou, F. Gao and X. Ji, "A novel IoT access architecture for vehicle monitoring system," *Conference: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016.
- [2] M. A. Ikram, M. D. Alshehri and F. K. Hussain, "Architecture of an IoT-based system for football supervision (IoT Football)," *Conference: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015.
- [3] A. S. Gaur, J. Budakoti, C. Lung and A. Redmond, "IoT-equipped UAV communications with seamless vertical handover," *IEEE Conference on Dependable and Secure Computing*, 2017.
- [4] C. Mouradian, S. Yangui and R. H. Glitho, "Robots as-a-service in cloud computing: Search and rescue in large-scale disasters case study," *15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2018.
- [5] A. Muneer, S. M. Fati, S. Fuddah, "Smart health monitoring system using IOT based smart fitness mirror," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 18, no. 1, pp. 317-331, 2020.
- [6] N. Nguyen, Q. Cuong Nguyen, M. T. Le, "A novel autonomous wireless sensor node for IoT applications," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 5, pp. 2389-2399, 2019.

- [7] D. K. Halim, T. C. Ming, N. M. Song and D. Hartono, "Software-based turbo decoder implementation on low power multi-processor system-on-chip for Internet of Things," *2017 4th International Conference on New Media Studies (CONMEDIA)*, 2017.
- [8] V. Kanakaris, G. A. Papakostas, D. V. Bandekas, "Power consumption analysis on an IoT network based on wemos: a case study," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 5, pp. 2505-2511, 2019.
- [9] D. Belkacemi, Y. Bouchebaba, M. Daoui and M. Lalam, "Network on Chip and Parallel Computing in Embedded Systems," *IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, 2016.
- [10] D. Hartono, S. W. Lee, V. V. Yap, M. S. Ng and C. M. Tang, "A multicore system using NoC communication for parallel coarse-grain data processing," *Conference on New Media Studies (CoNMedia)*, 2013.
- [11] C. Jalier, D. Lattard, A. A. Jerraya, G. Sassatelli, P. Benoit, and L. Torres, "Heterogeneous vs homogeneous MPSoC approaches for a Mobile LTE modem," *Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, 2010.
- [12] E. H. E. Mimouni, M. Karim, and M.-Y. Amarouch, "An FPGA-based MPSoC for real-time ECG analysis," *Third World Conference on Complex Systems (WCCS)*, 2015.
- [13] J. Feng, J. Li, L. Ma and H. Chen, "Feature level fusion of SAR and optical image and MPSoC Implementation," *IET International Radar Conference*, 2013.
- [14] D. N. C. Loong, S. Isaak, Y. Yusof, "Machine vision based smart parking system using Internet of Things," *TELKOMNIKA Telecommunication Computing Electronics and Control*, vol. 17, no. 4, pp. 2098-2106, 2019.
- [15] F. Lokananta, S. W. Lee, M. S. Ng, Z. N. Lim and C. M. Tang, "UTAR NoC: Adaptive Network on Chip architecture platform," *3rd International Conference on New Media (CONMEDIA)*, 2015.
- [16] C. Eisserer, "Portable Framework for Real-Time Parallel Image Processing on High Performance Embedded Platforms," *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2015.
- [17] R. Giorgi, M. Procaccini and F. Khalili, "AXIOM: A Scalable, Efficient and Reconfigurable Embedded Platform," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.
- [18] D. Cordes, M. Engel, O. Neugebauer and P. Marwedel, "Automatic Extraction of pipeline parallelism for embedded heterogeneous multi-core platforms," *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, 2013.
- [19] H. Schuster, M. Wenzl and M. Zauner, "A framework for teaching embedded multi-core programming," *IEEE/ASME 8th International Conference on Mechatronic and Embedded Systems and Applications*, 2012.
- [20] M. Banzi, "Getting Started with arduino," O'Reilly Media, Inc. 2009.
- [21] A. Wicaksana, D.K. Halim, D. Hartono, F. Lokananta, S. W. Lee, M. S. Ng and C. M. Tang, "Case Study: First-Time Success ASIC Design Methodology Applied to a Multi-Processor System-on-Chip," *IntechOpen*, 2018. doi: 10.5772/intechopen.79855
- [22] D. Hartono, M. S. Ng, Z. N. Lim, S. W. Lee, V. V. Yap and C. M. Tang, "A scalable bootloader and debugger design for an NoC-based multi-processor SoC," *3rd International Conference on New Media (CONMEDIA)*, 2015.
- [23] Dureen Kusuma Halim, "Software defined radio-based transceiver system on low power multi-processor system-on-chip for internet of things," *Master dissertation/thesis Universiti Tunku Abdul Rahman (UTAR)*, 2018.
- [24] ARM, "Cortex M0 Technical Reference Manual," [Online], Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0432c/DDI0432C_cortex_m0_r0p0_trm.pdf. 2009. Accessed on March 2019 from ARM Information Center.
- [25] Matthijs Kooijman, "Arduino IDE 1.5 3rd party Hardware specification," [Online], Available: <https://github.com/arduino/Arduino/wiki/Arduino-IDE-1.5-3rd-party-Hardware-specification#pre-and-post-build-hooks-since-ide-165> (Accessed on January 2019).

BIOGRAPHIES OF AUTHORS



Stefanus Kurniawan recently graduated from Universitas Multimedia Nusantara, with a Bachelor of Engineering in 2019. After graduating, he works in of the company in Indonesia as a software developer. He is interested in technology especially for hardware such as microcontroller and networking. He also qualified with HCNA R&S certification in 2018.



Dareen Kusuma Halim received his bachelor's degree in computer engineering from Universitas Multimedia Nusantara (UMN), Indonesia in 2014, and his Master of Engineering Science (Electrical and Electronics) from Universiti Tunku Abdul Rahman, Malaysia. During his master's study, Dareen joined the UTAR VLSI Centre team and together has successfully taped-out quad-core ARM M0-based MPSoC, the RUMPS401. The chip works on the first tape-out, and its capability was demonstrated in a complex application of software-defined radio. He is currently lecturing in Computer Engineering Department, UMN. His research interest includes VLSI & embedded system, distributed system, and IoT platform.



CM Tang held senior technical and management positions in AT&T Bell Laboratories, Lucent Technologies, Agere Systems, Shanghai Hua Hong Semiconductor International, SZGC Technologies and GLX Technologies in a career in VLSI design that spanned over 35 years. Currently leads as Chairman, VLSI Design Research Center, Universiti Tunku Abdul Rahman, Malaysia.



Dicky Hartono is a System-on-Chip Design and Verification Engineer in a Semiconductor Company in Shanghai, China. He received his bachelor's degree in computer engineering from Universitas Multimedia Nusantara, Tangerang, Indonesia in 2011, and his MEngSc (Electrical and Electronics) from Universiti Tunku Abdul Rahman, Kampar, Malaysia in 2015. During his Master study period, he successfully leads a team of Engineering Students to design a quad-core ARM Cortex-M0 chip using NoC. He has more than six years of professional experience in SoC Design and Verification, Digital Backend Design, Analog Design, Memory and Build-in-Self Test development, and Embedded Application Development, and has been working in multiple ARM Cortex-based projects in Smart Meter, Home appliances and Motor Driver applications.