A Cell-based Implementation of ARM Cortex-M0 SoC

Hye-Seung Sun[1,a] and In-Shin Cho[2]
[1]Department of Semiconductor Design, University of Korea Polytechnics
[2]Integrated Circuit Design Education Center, Korea Advanced Institute of Science and Technology
E-mail : [1]shspoly@kopo.ac.kr, [2]ischo@idec.or.kr

***Abstract* - For academic research purposes, ARM provided the Cortex SoC platforms. This study presents the methods and results of implementing the Cortex-M0 as a chip. The behavioral memory was replaced with the Samsung memory, and the design was modified to allow software to be written externally to the chip at any time. Efforts were made to fully utilize as many EDA tools as possible during the chip implementation to maximize the chip's functionality and performance. The fabricated Cortex-M0 chip was mounted on a test board and was verified to work well in conjunction with the software. A hierarchical approach was adopted during the chip implementation, with efforts made to minimize its size and facilitate its use as a platform. In this study, the Samsung 28nm LPP process was applied.**

***Keywords* – ARM, Cortex-M0, Cell based Digital ASIC, Platform, Samsung 28nm, Hierarchical Design**

I. INTRODUCTION

The most convenient opportunity for participating in chip design at Korean universities is the IC Design Education Center (IDEC) MPW program. This is because participants can receive financial support for participating in overseas processes, and in the case of domestic processes, they can receive fast and accurate technical support. Through the process of implementing university ideas into chips, students can gain practical experience, generate research results through chip measurements and demos, and even achieve research accomplishments. Currently, many Korean professors are participating in the IDEC MPW program, but looking at the distribution, the digital field is very marginal. Analog and mixed fields account for more than 90% of the ratio, while digital design shows about 10%. There may be various reasons for this situation, but the absence of platforms is also a contributing factor.

A platform can be compared to a large building made of assembly blocks. If some people build a sturdy and large building, they can create connecting bridges with other buildings or stack floors higher. This means good scalability. Having a verified platform enables the attachment of independent function units or IP blocks and confirming the independent function verification and integration with the entire system, which is very useful for SoC configuration. Design companies in the SoC field have various platforms including ARM, allowing them to create larger systems by integrating various IPs through verification. However, there are budget and manpower issues in most universities, so it is difficult to create the qualified modules or IPs. Furthermore, even if someone provides a platform to the university, it is difficult to actively adopt if it is not widely known or widely used, as graduates would need to learn new skills after entering a job.

In this regard, the fact that ARM provides ARM processors and SoC platforms to professors and researchers at universities free of charge is very welcome news. Upon following the guidance of ARM personnel, after joining the ARM Developer site and going through the NDA procedure, it is possible to download SoCs including ARM Cortex A, R, M series as well as AMBA buses and peripheral controllers. There are various types of systems, including sources modeling ARM-made FPGA boards and providing FPGA images, making it easy for newcomers to use ARM SoC platforms. Not only hardware source code but also software support is provided. Keil compiler, arm-linux-gcc, ARM Development Studio, and various basic examples are provided for free. Moreover, if there is a Debug Access Point on the ARM Core, debugging is possible by setting break points through software development tools. ARM also provides lecture slides, online lectures, user guide documents, professional books on ARM Core, and practical examples, making it virtually comprehensive [1]. Now, what's important next is documents like "How to implement on Semiconductor" from a user perspective. This paper covers ARM Cortex-M0 based SoC platforms and discusses what needs to be considered when implementing them into chips.

II. CONSIDERATION FOR SOC IMPLEMENTATION

A. Hardware and Software Description of ARM Cortex-M0

The ARM Cortex-M0 is the first CPU Core in the Micro Controller series of Cortex, operating at a 50MHz clock speed on TSMC 180nm process, including CPU core, AMBA bus AHB Lite, and APB bus. It also includes memory interfaces and multiple AHB, APB slaves like GPIO, timer, Watchdog timer, UART Controller, etc. All components are written in Verilog Language, allowing them to be opened with text editors like vim. Inside the source code, there are

a. Corresponding author; shspoly@kopo.ac.kr

Manuscript Received Aug. 12, 2024, Revised Sep. 20, 2024, Accepted Sep. 26, 2024

parameters in defines.v, allowing for flexible settings. The Fig. 1 shows the architecture of Cortex-M0.

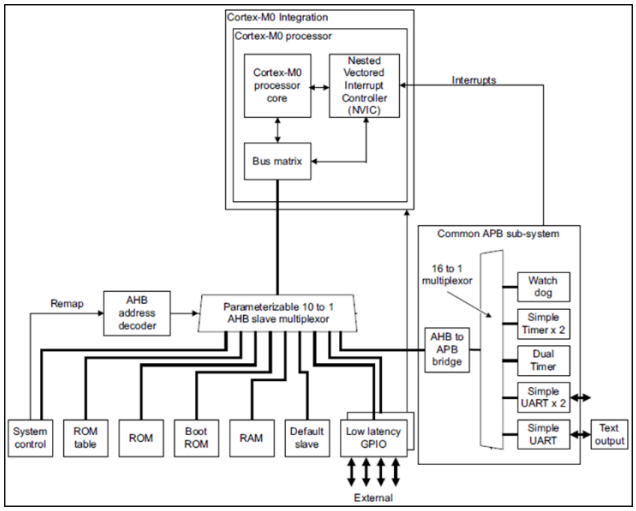

Fig. 1. Hardware Structure of ARM Cortex-M0 SoC.

The software development environment is operable on both Windows and Linux. For Linux, the arm-linux-gcc compiler is provided, and for Windows, ARM Development Studio tool and Keil microvision are provided [2]. When the ARM Cortex-M0 integration package is uncompressed, basic software examples are visible. In this study, microvision and the essential pack were installed before compiling the examples provided as defaults. Examples such as hello, uart, interrupt, timer, watchdog timer, etc., are diverse, allowing the designer to operate the SoC as desired.

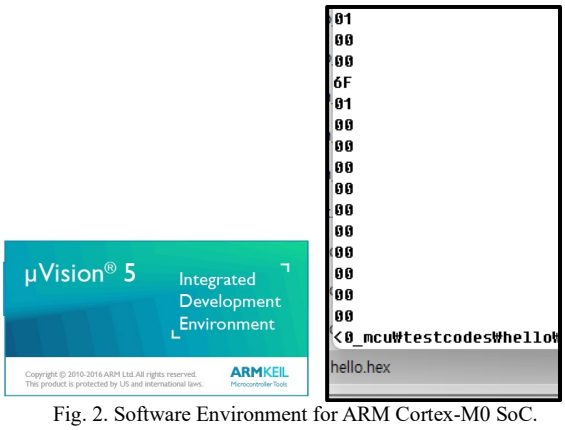

Fig. 2. Software Environment for ARM Cortex-M0 SoC.

Fig. 2 shows the GUI of the microvision, which is the software development tool in this study. If the hex file generated by compilation is opened, it can be observed that each line contains two numbers as hex code. For SoC verification, it is important to integrate this hex file with the hardware to verify its operation. Inside the package, there is a Testbench, and clkreset.v describes the operation of clock and reset, allowing verification using a Verilog simulator. In this study, Synopsys VCS and Verdi were used, as shown in Fig. 3, demonstrating the situation where Cortex-M0 operates with software [2]. The normal operation messages of Cortex-M0 are visible, and the time taken in nanoseconds can also be verified. This study focuses not on the internal issues of the CPU such as the ISA and pipeline configuration of Cortex-M0 but rather on verifying Cortex-M0 and implementing it into silicon following the Digital Flow.

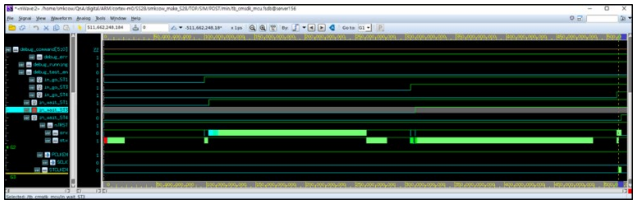

Fig. 3. Hardware and Software operation of ARM Cortex-M0 SoC.

B. Change from Behavioral memories to Macro Memories for implementing SoC

The first issue to consider when implementing the SoC platform into hardware is memory. Inside the enclosed package are 64KB sized RAM and ROM in behavioral model format. Both memories contain [13:0] reg [31:0], and simulation can be conducted by specifying the location of the compiled hex file through the readmemh() syntax in the initial section. However, these RAM and ROM models are not suitable for hardware implementation. Generating two-dimensional regs during FPGA implementation consumes a lot of compilation time, occupies a large amount of device capacity, and lacks consideration for delays, making it impossible to ensure proper operation. As an alternative, macro blocks within the FPGA must be used. Running Megawizard or IP Generator to create memory requires careful verification of the device specs to prepare the FPGA board since two 64KB memories need to be created. If a suitable board is not available, utilizing external memory on the FPGA board is also an option. Becoming familiar with the datasheet of DDR or DRAM and implementing the controller as RTL source code or obtaining appropriate IPs is essential. If implementing with chips instead of FPGA, a chip designer should use the memory provided by the foundry. Each foundry has its own supported memory compiler, and if it offers SRAM, it is suitable for use with Cortex-M0. In this study, Samsung foundry was utilized, which provides memory compilers supporting various sizes, thus enabling the creation of a 64KB macro.

C. Change ROM Area with SRAM for implementing SoC

The second issue to consider in SoC implementation is the use of ROM. Filling ROM involves software compiled into hex or binary files. Since software code needs numerous debugging and modifications before completion, using ROM from the initial project stage greatly increases the likelihood of chip malfunction due to software bugs. Ultimately, reconfigurable EEPROM or rewritable flash appears to be a better option than ROM. However, the Samsung foundry available for domestic universities in Korea only provides SRAM, ROM, and register file. Therefore, in this study, ROM is replaced with SRAM, and ROM code is proposed to be rewritten externally via UART communication. To facilitate this, PCB boards are configured to accommodate USB-to-serial chips for connecting to the PC via USB cable when UART channels

are available on the packaged chip's pins. The SRAM is programmed by creating a hex file using the microvision tool on a PC and transferring it to the ROM area of the SoC via the HyperTerminal program. To ensure smooth progress in this process, the design of a module capable of controlling the transmission of hex files from start to finish is necessary. This module is responsible for controlling SRAM, UART controller, and data flow.

The description of the first control is about SRAM, among the three, as follows. When generating SRAM with a memory compiler, documentation for that memory is usually produced, allowing confirmation of basic memory operations. However, the methods for controlling continuous read and write operations are often not specified. Therefore, designers need to conduct extensive simulations based on basic information about reads and writes to avoid errors in memory control.

The second control is about UART. Since hex codes will be transmitted between the PC and the board via UART cable and HyperTerminal, the communication speed and flow control need to be determined in advance. Designers must choose an appropriate baudrate from those supported by the HyperTerminal program, and once the speed is determined, the values of UART control registers need to be adjusted to match the clock speed received by the UART controller. Additionally, various simulations are required for flow control, parity bits, interrupt handling, and other factors.

The third is about data flow control. Since the ROM code to be stored in the on-chip SRAM via UART communication is a continuous stream of long strings, it needs to be received and stored in SRAM in predetermined sizes. Additionally, when the UART operates, the SRAM on the other side must wait, so a module design that can sequentially control the flow of data is necessary. First, when examining the hex code, each line consists of two hex codes, such as 2F and 8A. Since UART communicates in ASCII code, if 2 is sent first from 2F, it must be converted to 0x52 before sending. Within the on-chip system, when 0x52 is received via UART communication, it must be restored to 2, temporarily stored, and then wait for the next value, F, to form 2F. This process must be repeated four times to receive a line, 32-bit, and then it needs to be written. Since Cortex-M0 is a 32-bit processor, it needs to write to memory in 32-bit word units. It is also important to note that ARM Cortex-M0 uses Little Endian mechanism, so the received codes need to be correctly shifted and stored. Taking all these into account, once a 32-bit word is formed, the UART transmission pauses, and it is written to the on-chip SRAM. After that, during the reception of the next 1-word via UART communication, the chip select of the on-chip SRAM should be disabled, and the address should be incremented by 1.

So far, the need for three controls has been explained. In this study, a module implementing all the described operations was named FIFO, and it was designed to perform sequential operations using FSM. The FIFO operates with a 50MHz clock, along with the UART controller and memory. The implemented FIFO module can write and read to/from the on-chip SRAM and also transmit and receive messages via UART. Instruction codes can be written to the on-chip SRAM and read back for inspection by the designer. Fundamentally, the transmission of hex codes should occur before the CPU operates, so while the FIFO module is operating, the Cortex-M0 SoC is reset and the main clock is gated to minimize unnecessary power consumption. Once the FIFO operation is completed, the clock entering the FIFO is gated, the clock entering the Cortex-M0 SoC is restored, and the reset is released. Since the hex codes are properly written into the on-chip SRAM, the channel with FIFO is disabled, and the channel with the Cortex-M0 is enabled. Through Fig. 4, the connection of FIFO, UART, and memory can be observed, and through Fig. 5, the proper operation of the FIFO can be confirmed by displaying messages on the terminal.

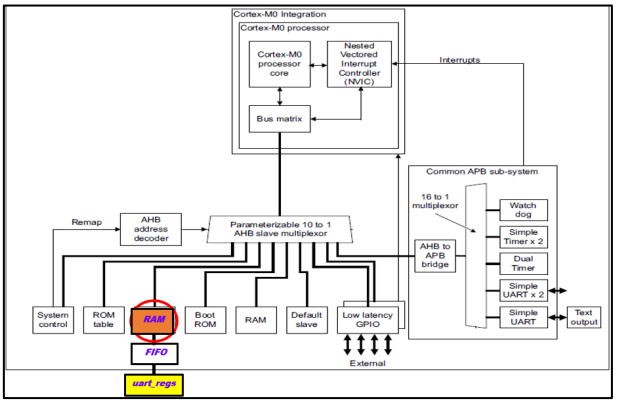

Fig. 4. Proposed SoC including SRAM, FIFO and UART.

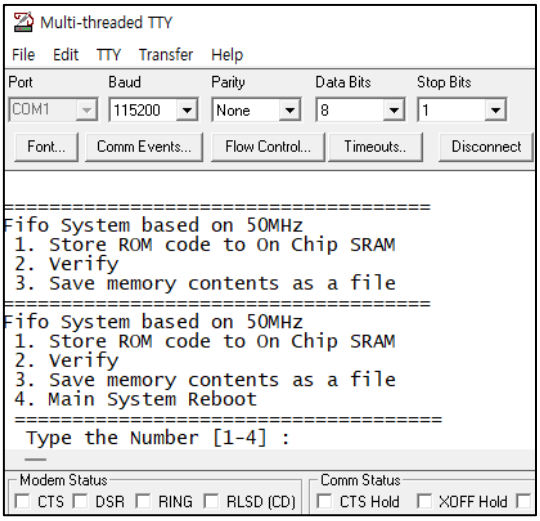

Fig. 5. Co-operation of SRAM, FIFO and UART.

D. Timing Consideration between Core and Memories

The third issue to consider is the timing structure between the CPU, AMBA bus, and memory. Fig. 6(a) illustrates the Cortex-M0 CPU, which acts as the master connected to the AHB bus, and the AHB_ROM, which serves as the slave. The timing diagram shown in the upper part of Fig. 6(b) represents the operation of the enclosed Cortex-M0 SoC package. This package contains behavioral memory within the AHB_ROM module, designed for simulation purposes but not suitable for hardware implementation. One of its key characteristics is that it performs read and write operations without delay, allowing operations that typically require two

cycles to be executed in one. In the upper part of Fig. 6(b), at time T0, right after the reset, the Cortex-M0 CPU sends an instruction fetch request by issuing the initial address. Simultaneously, the address is assigned to HADDR and reaches the AHB_ROM. At time T1, the AHB_ROM module decodes the AMBA signals and decides to assert a read signal to the behavioral memory. The address and read signal are expected to be delivered to the memory by T2. However, since the behavioral memory operates immediately, it outputs the code instruction at time T1, allowing the Cortex-M0 CPU to receive the code at time T2.

In this study, the goal is to replace the behavioral memory with Samsung's 28nm memory while ensuring that the Cortex-M0 CPU receives the requested instruction at time T2, as it did with the behavioral memory. Since Samsung memory is not a behavioral memory, it operates by receiving address values and read control signals at time T2, and the problem is that the Cortex-M0 CPU receives the instruction code at time T3. The solution, which can be seen at the bottom of Fig. 6-b, is to perform the tasks performed at time T1 in advance. The logic that sets the address value and control signal value within the AHB_ROM module is made to operate as a negative clock. Setting the address value and control signal value in advance before T1 helps resolve setup timing violation, making chip implementation easier.

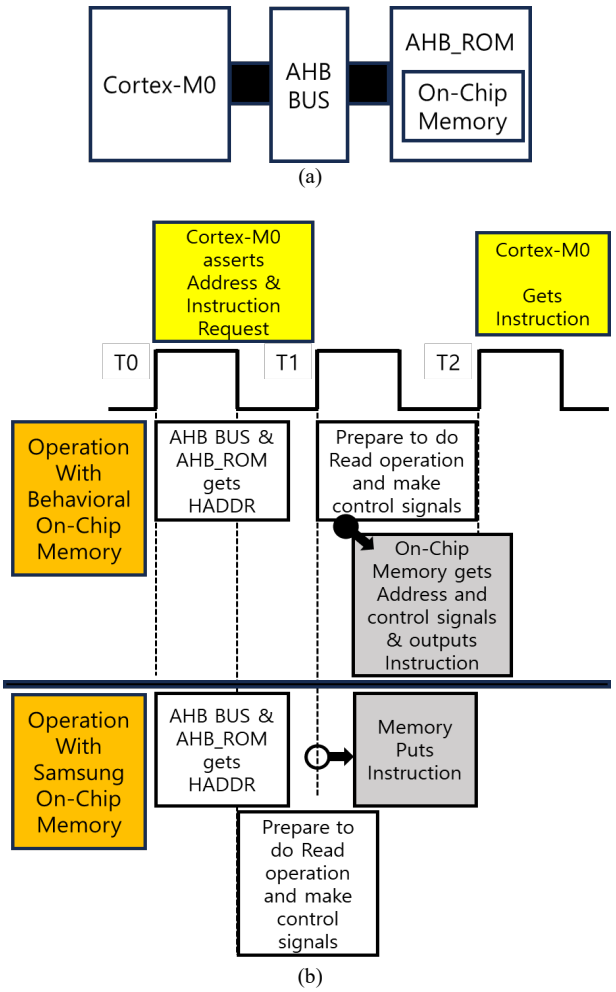

Fig. 6 (a) AHB AMBA BUS Architecture and (b) Operation with Behavioral and Samsung On-Chip Memory.

E. Hierarchical Design

The fourth issue to consider in SoC implementation of Cortex-M0 is Hierarchical Design. It is not common to implement the entire SoC Design in a flattened manner. Typically, SoC Design is divided into small block units, and implementation is carried out in parallel by teams of many people. Experienced designers will consider Front-End implementation in terms of constraints and scenarios, and they will think about strategies for Back-End implementation that make it easy to implement in terms of block shape, size, and pin placement. Additionally, they will consider how to verify at the higher level, which will include timing, power, functionality, DRC and LVS. For example, at the Front-End stage, when constraints are being designed at the Top Design level, consideration needs to be given as to whether constraints for timing, including clocks, should be applied to sub-modules. If there are PLLs in the system or if there are different types and operating cycles of clocks that need to be applied to each sub-module due to Half CLK, then constraints will need to be written and implemented individually for each case. At the Back-End stage, it will be necessary to determine the direction of pins considering the size and position of small blocks and their connectivity so that routing can proceed in the shortest distance when implemented. Additionally, from a functional scenario perspective, if there are specific periods where small blocks do not need to operate, gating the CLK entering those blocks or gating the power to reduce power consumption can also be considered. Hierarchical Design is advantageous because if a specific block needs to be modified, only that design needs to be modified, rather than having to re-implement the entire system as with the flattened approach. In this study, the Cortex-M0 Core was pre-implemented up to the Front-End stage, and the on-chip RAM and ROM were pre-implemented up to the Back-End stage before being utilized in the overall SoC implementation. Since memory occupies the most area in the Floorplan in Cortex-M0 SoC implementation, it was progressed to the Back-End, and the Core, which needs to operate with the highest CLK, was separately synthesized to prepare mapped ddc for overall integration. What needs to be checked in advance is that there is a block inside the Cortex-M0 core that is composed of nets, so it is not easy to read and is impossible to modify. However, because this part is implemented with a considerable length of combination logic, a significant amount of time and effort is required to ensure there are no setup timing violations after synthesis. Excluding the Core from the overall design, there are no significant difficulties in implementation, making it advantageous to pre-implement only the Cortex-M0 core for SoC implementation. Another part where Hierarchical Design is needed is with memory. Fig. 7 shows a notable example of the results of auto-routing on the pins of the memory. In situations where 1'b1 or 1'b0 are commonly assigned, routing is first connected to a specific pin in the memory, and when connected to the next pin, it is carried out in the inner area of the memory. Another similar case is where clock nets are well connected to the CLK pin of the memory but the clock shield nets are going to the inside of memory. These occurrences causing errors stem from using the memory as a

phantom cell in the Auto PnR stage, a characteristic of the files provided by the foundry to designers, leaving designers with no choice but to verify them directly. In this study, a memory wrapper was created by instantiating SRAM and connecting the input and output pins externally. With no special functions, only wire connections were used to ensure that there is no impact on the functionality and timing of the memory, and to address the issues described above during Auto PnR. If the Auto Router makes mistakes, the problematic patterns are manually moved out to allow metal and VIA to be positioned outside the memory. The memory wrapper, completed up to the Back-End stage, was applied commonly to both ROM and RAM.

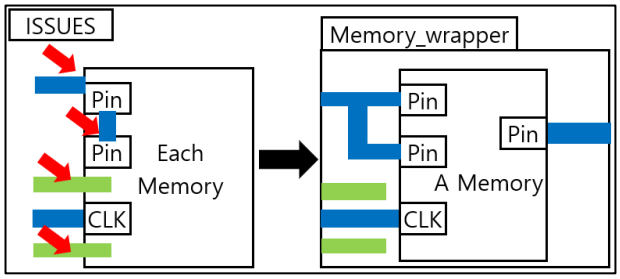

Fig. 7. Fatal Result of Auto PnR toward memory.

III. IMPLEMENTATION FROM RTL TO GDS

In this study, a system is implemented on a chip by attaching a FIFO and UART controller to the basic ARM Cortex-M0 system. Samsung 28nm LPP process was utilized, and using the memory compiler provided by the foundry, a synchronous SRAM with a size of 16384X32 bits was created. This memory is used commonly as both ROM and RAM. The memory is configured to operate on the falling edge, and the SRAM used in the ROM area had most of its pins directly connected to the AHB bus and Cortex-M0 CPU core, while it was shared with the FIFO & UART for interface. The operational scenario is as follows. First, prepare the instruction code and then transmit it from the PC to the on-chip area using the HyperTerminal program. The code entering the on-chip area via UART communication is stored in the SRAM replacing ROM through the FIFO. Until the write process is completed, the entire system, including the CPU, remains in a reset state and the main clock is gated, waiting for the write to finish. Once the write is complete, the clock going to the FIFO and UART is gated in the opposite direction, booting up the main system. Verifying the normal operation of the SoC system involves checking messages through the UART communication port directed to the outside of the chip. In this study, the SoC system operating in the sequence described above was implemented from RTL to GDS, and ultimately, it was implemented as a Hard Macro form after being reduced to a small area for use as an SoC platform. The process used for implementation is Samsung 28nm LPP process, which has the following characteristics: standard cells can operate in the GHz range, IO cells include ESD functionality, supporting input-output bi-directional operation, and signals up to 200MHz, and up to 160 IO cells and pad cells can be used on an area of 4x4 mm. Memory includes ROM and various types of SRAM and register files, providing support for various sizes. The package proceeded with the LQFP 208 type, and the design using EDA tools could be conducted within a cloud server with IDEC support. Synopsys tools were used for implementation up to the Second Sign-off, and for tasks like merging, Physical DRC and LVS, Cadence and Siemens EDA tools were utilized using the final GDS file. All licenses were obtained through IDEC. Opportunities in Samsung 28nm process were accessible through the IDEC MPW program, and comprehensive technical support throughout the design period was provided by IDEC engineers.

A. Function simulation

The tool for function simulation is Synopsys VCS and Verdi tool. Fig. 8 shows the initial operation status of the SoC. The FIFO displays messages through the UART module and prompts the user to press numbers 1-4. Option 1 stores the compiled hex file in the ROM area of the SoC. Option 2, after saving the hex file, displays the instruction code stored at the specified address. Option 3 reads the entire instruction code stored in the ROM area of the SoC and transmits it through the UART channel. It can be read by the HyperTerminal program on the PC and saved as a file, allowing comparison with the original Hex code to assess the write and read operation of the FIFO and memory. If there are no abnormalities in options 1-3, option 4 boots the Cortex-M0 system. Option 4 releases the reset on the Cortex-M0, Amba bus, and peripherals, and enables CLK normally to verify the overall operation of the SoC system, while disconnecting the CLK to the FIFO and UART to reduce power consumption. Fig. 9 shows the messages during the operation of option 4 and the normal operation of Cortex-M0 and Fig. 10 shows the output of the Gated CLK at the time.

The CLK period for simulation was set to 5ns. These conditions were described in constraints and applied to the synthesis stage.

```
ucli% run
====================================
 Fifo System based on 50MHz
 1. Store ROM code to On Chip SRAM
 2. Verify
 3. Save memory contents as a file
 4. Main System Reboot
 ====================================
 Type the Number [1-4] :  1
Send ROM code via MTTTY and Press q
 20000368
00000165
0000016d
0000016f
00000000
```

Fig. 8. Storing ROM code to on-chip SRAM.

```
====================================
 Fifo System based on 50MHz
 1. Store ROM code to On Chip SRAM
 2. Verify
 3. Save memory contents as a file
 4. Main System Reboot
 ====================================
 Type the Number [1-4] :  4
START! IDEC Cortex-M0
  306917118 ns UART: Hello world
306919683 ns UART: smkcow
306925163 ns UART: ** TEST PASSED **
tb_cmsdk_mcu.v, 213 :    #32700 $stop;
```

Fig. 9. Normal operation of Cortex-M0.

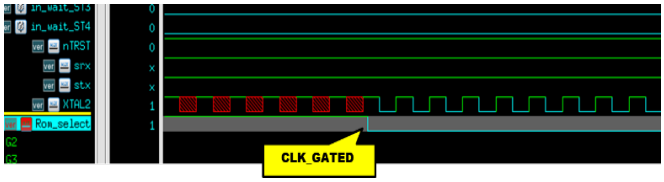

Fig. 10. The result of gated clock of Cortex-M0.

B. Synthesis

The tool for synthesis is Synopsys Design Compiler [4]. According to the process guide from Samsung, 80% of the clock period is applied when performing topographical synthesis. Because the maximum available frequency of the IO cell provided by Samsung's 28nm process is 200 MHz, 5ns was set as the target and 4ns was applied as a constraint. The maximum synthesizable clock frequency is also 4ns.

Synthesis of the Cortex-M0 Core and extraction of the DDC were performed in advance and the entire implementation of the memory wrapper is completed and both DDC and NDM were extracted. Therefore, during the synthesis of the TOP design's Hierarchical Design, pre-created DDCs were used as is, and all Verilog source codes for the remaining modules were read and synthesized at once. For this purpose, the ***set_dont_touch*** command is applied to the pre-created modules. After synthesis, the ***remove_design*** command is used to remove the mapped DDC and gate-level netlist of the Cortex-M0 and the memory wrapper from the entire design. The resulting black box is replaced with the pre-created NDM during the Auto PNR stage using ICC2.

When synthesizing using the Samsung 28nm LPP process, it is characteristic to operate in Topographical mode, select .lib with the option to use the Composite Current Source (CCS) model, and apply Multi Corner Multi Mode (MCMM), and On Chip Variation (OCV). The tool operation proceeds with the command `**dc_shell -topo**`, and setup includes recognizing .tf, .tluplus, .map files, etc. The process for selecting operating conditions is shown in Fig. 11. Create a table and organize the features. Rows for 10% higher and lower voltages than the reference voltage of 1.1V are created, and columns for operating temperatures ranging from -40°C to 125°C are made. Then, after fixing the temperatures at -40°C and 125°C, the highest voltage at -40°C and the lowest voltage at 125°C are selected to define two operating PVT conditions. It is advantageous for SoC implementation to apply the same criteria for selecting .lib for STD Cell, IO Cell, and Memory. If Temperature Inversion (TIV) operating conditions must be selected, then choices completely opposite to the above conditions should be chosen. Also, referring to the Samsung 28nm process guide document, if some corners are marked as Dominant, the designer should choose and adopt [4].

Corner	Temperature	STD	IO	Memory
ff	m40	1.10V	1.95V	1.10V
ss	125	0.90V	1.65V	0.95V

Fig. 11. Operating conditions for synthesis.

```
set scenario mode_norm.OC_rvt_ss_0p900v_125c

create_scenario ${scenario}

# Read in scenario-specific constraints file

puts "RM-Info: Sourcing script file [which [dcrm_mcmm_filename ${DCRM_CONSTRAINT
S_INPUT_FILE} ${scenario}]]\n"

#puts "RM-Info: Reading SDC file [which [dcrm_mcmm_filename ${DCRM_SDC_INPUT_FIL
E} ${scenario}]]\n"
# constraint syntax is not based on SDC style.... so at first, just source and d
o read_sdc after write_sdc
source [dcrm_mcmm_filename ${DCRM_CONSTRAINTS_INPUT_FILE} ${scenario}]
##read_sdc [dcrm_mcmm_filename ${DCRM_SDC_INPUT_FILE} ${scenario}]

#set_operating_conditions -max_library <OC_WC_LIB_NAME> -max <OC_WC>
set_operating_conditions ss_0p900v_125c -library sc9_cmos28lpp_base_rvt_ss_nomin
al_max_0p900v_125c_sadhm -analysis_type on_chip_variation
# OCV variation from voltage IR-drop 3.6%
set_timing_derate -early 0.964
```

```
set scenario mode_norm.OC_rvt_ff_1p100v_m40c

create_scenario ${scenario}

# Read in scenario-specific constraints file

puts "RM-Info: Sourcing script file [which [dcrm_mcmm_filename ${DCRM_CONSTRAINT
S_INPUT_FILE} ${scenario}]]\n"

#puts "RM-Info: Reading SDC file [which [dcrm_mcmm_filename ${DCRM_SDC_INPUT_FIL
E} ${scenario}]]\n"
# constraint syntax is not based on SDC style.... so at first, just source and d
o read_sdc after write_sdc
source [dcrm_mcmm_filename ${DCRM_CONSTRAINTS_INPUT_FILE} ${scenario}]
##read_sdc [dcrm_mcmm_filename ${DCRM_SDC_INPUT_FILE} ${scenario}]

#set_operating_conditions -max_library <OC_WC_LIB_NAME> -max <OC_WC>
set_operating_conditions ff_1p100v_m40c -library sc9_cmos28lpp_base_rvt_ff_nomin
al_min_1p100v_m40c_sadhm -analysis_type on_chip_variation
# OCV variation from voltage IR-drop 3.6%
set_timing_derate -late 1.036
```

Fig. 12. Scenarios setup for synthesis.

Fig. 12 shows the results of selecting operating conditions for STD, IO, and Memory. In this study, two scenarios were created: one composed of Function Mode and the worst corner, and the other composed of Function Mode and the best corner. On-Chip Variation was also applied, with derate values set to 1.036 and 0.964 respectively, to implement a more pessimistic design. Fig. 13 shows the results of the ***report_scenarios*** command, highlighting the confirmed derate values.

```
Scenario #1: mode_norm.OC_rvt_ff_1p100v_m40c is active.
Scenario options: Setup Leakage_power
Cts_corner: none
Has timing derate: Yes

Library(s) Used:
 sc9_cmos28lpp_base_rvt_ff_nominal_min_1p100v_m40c_sadhm (File: /home/smkcow/QnA/dig
/PDK/sc/FE/LIBERTY/sc9_cmos28lpp_base_rvt_ff_nominal_min_1p100v_m40c_sadhm.db)
 io_gppr_cmos28lpp_t18_ff_1p155v_1p950v_m40c (File: /home/smkcow/QnA/digital/ARM/cor
/FE/LIBERTY/io_gppr_cmos28lpp_t18_ff_1p155v_1p950v_m40c.db)

Operating condition(s) Used:
  Analysis Type    : on_chip_variation
  Max Operating Condition: sc9_cmos28lpp_base_rvt_ff_nominal_min_1p100v_m40c_sadhm: f
  Max Process    : 1.00
  Max Voltage    : 1.10
  Max Temperature: -40.00
  Min Operating Condition: sc9_cmos28lpp_base_rvt_ff_nominal_min_1p100v_m40c_sadhm: f
  Min Process    : 1.00
  Min Voltage    : 1.10
  Min Temperature: -40.00
```

Fig. 13. Details of a scenario.

Fig. 14 shows the result of synthesis and mapped design. The synthesized design has no constraint errors and setup timing errors.

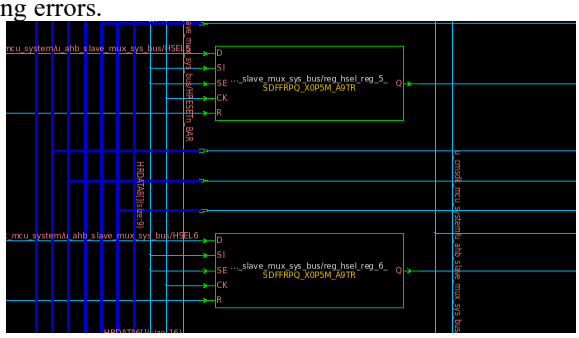

Fig. 14. Mapped design after Synthesis.

C. Pre-Layout Equivalence Check & STA

The tools for equivalence check and STA are Synopsys Formality and Primetime [5][6]. Figs. 15 and 16 show the results of Formality and Primetime execution.

In the stage of equivalence check, the verilog sources and the ddc files of the pre-implemented Cortex-M0 core and the memory_wrapper are read as references, and matching and verification are carried out after reading the mapped.ddc file as an implementation.

In the stage of STA, there are no errors in setup and recovery. Therefore, an SDF was extracted to proceed with Pre-layout Simulation.

```
****************************** Verification Results ******************************
Verification SUCCEEDED
   ATTENTION: synopsys_auto_setup mode was enabled.
              See Synopsys Auto Setup Summary for details.
--------------------------------------------------------------
 Reference design: r:/WORK/cmsdk_mcu
 Implementation design: i:/WORK/cmsdk_mcu
 6903 Passing compare points
--------------------------------------------------------------------------------
Matched Compare Points      BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
--------------------------------------------------------------------------------
Passing (equivalent)          696       0      32       0      36    6138       1    6903
Failing (not equivalent)        0       0       0       0       0       0       0       0
********************************************************************************
```

Fig. 15. Results of Equivalence Check.

```
Type of Check         Total              Met         Violated         Untested
-------------------------------------------------------------------------------
setup                  6804      6246 ( 92%)       0 (  0%)       558 (  8%)
hold                   6804      6246 ( 92%)       0 (  0%)       558 (  8%)
recovery               5934      2355 ( 40%)       0 (  0%)      3579 ( 60%)
removal                5934      2355 ( 40%)       0 (  0%)      3579 ( 60%)
min_period               65         3 (  5%)       0 (  0%)        62 ( 95%)
min_pulse_width       48298     24556 ( 51%)       0 (  0%)     23742 ( 49%)
clock_gating_setup        2         1 ( 50%)       0 (  0%)         1 ( 50%)
clock_gating_hold         2         1 ( 50%)       0 (  0%)         1 ( 50%)
out_setup                36        35 ( 97%)       0 (  0%)         1 (  3%)
out_hold                 36        35 ( 97%)       0 (  0%)         1 (  3%)
-------------------------------------------------------------------------------
All Checks            73915     41833 ( 57%)       0 (  0%)     32082 ( 43%)

1
write_sdf -enabled_arcs_only -context verilog cmsdk_mcu.sdf_PT
```

Fig. 16. Results of Pre-layout STA.

D. Pre-Layout Simulation & Power Estimation

The tools for simulation and power estimation are Synopsys VCS, Verdi and PrimePower [7].

Fig. 17 shows the results of Pre-layout Simulation. The result is identical to the functional simulation result and it can be observed that the system operates normally even with the annotation of sdf files incorporating cell delays.

```
ucli% run
=====================================
 Fifo System based on 50MHz
 1. Store ROM code to On Chip SRAM
 2. Verify
 3. Save memory contents as a file
 4. Main System Reboot
 =====================================
 Type the Number [1-4] :  1
Send ROM code via MTTTY and Press q
 20000368
00000165
0000016d
```

```
=====================================
 Fifo System based on 50MHz
 1. Store ROM code to On Chip SRAM
 2. Verify
 3. Save memory contents as a file
 4. Main System Reboot
 =====================================
 Type the Number [1-4] :  4
START! IDEC Cortex-M0
  306917118 ns UART: Hello world
306919683 ns UART: smkcow
306925163 ns UART: ** TEST PASSED **
tb_cmsdk_mcu.v,  213 :    #32700 $stop;
```

Fig. 17. Results of Pre-layout Simulation.

Using the `**fsdb2saif**` command, a saif file with only transition data in text format was generated and fed into the PrimePower tool for power estimation. Fig. 18 shows the contents of saif file and Fig. 19 shows the power consumption of standard cells, IO cells, and Memory. It is the result of average voltage estimation.

```
(VENDOR "Synopsys, Inc.")
(PROGRAM_NAME "fsdb2saif")
(VERSION "Verdi_M-2017.03-SP2")
(DIVIDER / )
(TIMESCALE 1 ps)
(DURATION 298835433)
(INSTANCE tb_cmsdk_mcu
 (NET
  (XTAL1
   (T0 149417923) (T1 149417500) (TX 10)
   (TC 119534) (IG 0)
  )
  (XTAL2
   (T0 150074697) (T1 148760053) (TX 683)
   (TC 119533) (IG 0)
  )
  (IN_fifo_CLK_50
   (T0 149419990) (T1 149415433) (TX 10)
   (TC 29883) (IG 0)
  )
```

Fig. 18. Contents of the extracted saif file.

```
                        Internal  Switching  Leakage    Total
Power Group             Power     Power      Power      Power   (     %)  Attrs
-------------------------------------------------------------------------------
clock_network           1.807e-03 2.064e-04 1.773e-07 2.013e-03 (36.80%)  i
register                1.769e-05 2.562e-07 2.711e-06 2.065e-05 ( 0.38%)
combinational           6.561e-06 1.130e-04 3.651e-06 1.233e-04 ( 2.25%)
sequential                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                  7.113e-04    0.0000 3.561e-05 7.469e-04 (13.65%)
io_pad                  2.036e-03 1.435e-06 5.292e-04 2.567e-03 (46.92%)
black_box                  0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 3.211e-04   ( 5.87%)
  Cell Internal Power  = 4.578e-03   (83.69%)
  Cell Leakage Power   = 5.713e-04   (10.44%)
                         ---------
Total Power            = 5.471e-03  (100.00%)
```

Fig. 19. Result of Power Estimation.

E. Auto PnR

The tool for Auto PnR is Synopsys ICC2 [8].

The process used in this study is the Samsung 28nm LPP process. Since the foundry provides the standard cell, IO cell, and memory in LEF format, they were converted to NDM format for ICC2 work before proceeding with PnR [3][8]. When MPW is conducted through IDEC, a size of 4000x4000um can be used, and 160 IO cells can be utilized. Therefore, in this study, while implementing the Cortex-M0 SoC, IO cells were used and the core area was minimized using a hierarchical design approach.

First, the pre-made NDM file of the memory wrapper module was reused for the hierarchical design. As shown in Fig. 20, since the area of the memory is the largest, the memory pins were symmetrically placed facing each other, and the floorplan was designed to accommodate most of the standard cells between the memories [9]. The ICC2 tool execution and the scripts used for the design implementation in this study were referenced from the Reference Methodology (RM) provided by the Synopsys SolvNet site [10]. This script set can be executed using the **make** method and is composed of detailed steps shown in Figure 21. At each step, verification of timing, power/ground connectivity, DRC, and LVS was conducted.

Fig. 22 shows the results of the auto PNR from this study. If the platformization is implemented, the left side of the chip could be used as a hard macro, and the right side of the chip could be designed differently.

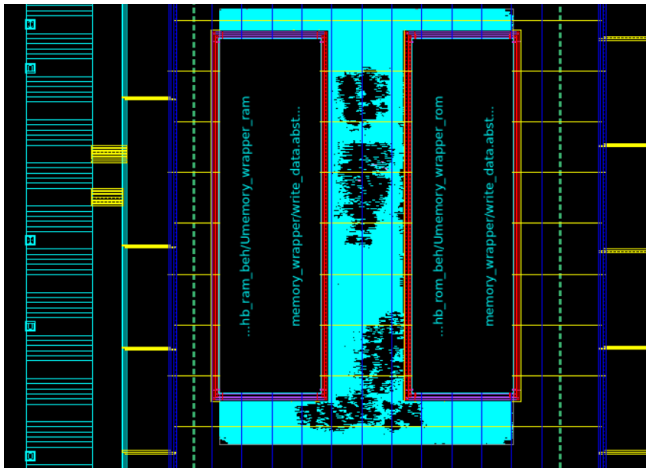

Fig. 20. Floorplan and Powerplan of Cortex-M0 SoC.

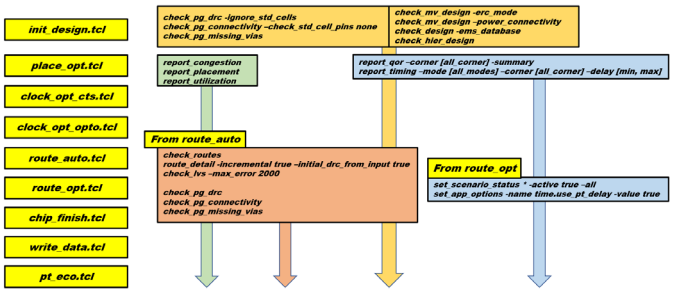

Fig. 21. Internal steps of Auto PnR and Verification Command.

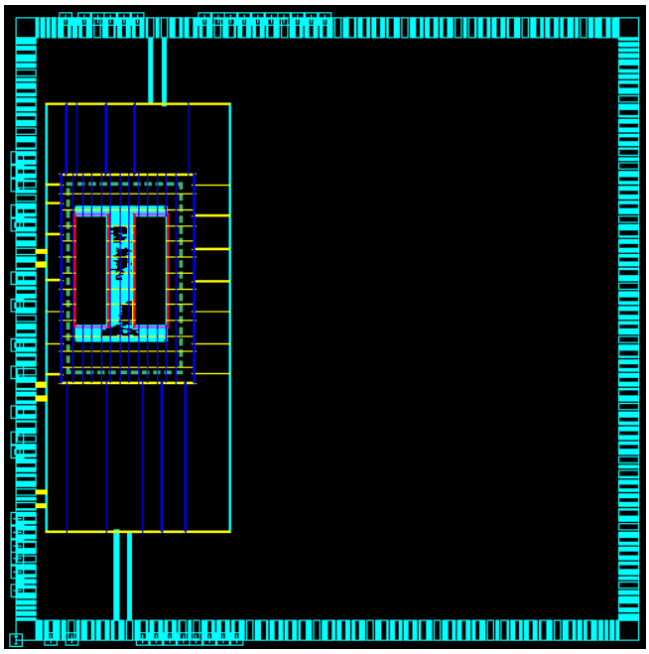

Fig. 22. Full Chip Implementation.

F. Post-layout Equivalence Check and Parasitic Extraction

The tool used for Equivalence Check is Formality, and for Parasitic Extraction, Synopsys StarRCXT was utilized [11].

When using the Formality tool, the gate-level netlist after synthesis is designated as the reference, and the gate-level netlist after Auto PNR is set as the implementation, and then the process is carried out. The RM for ICC2 can be conducted using the make method, with the Makefile containing an item for fm, which enables execution of the process with the **make fm** command. Although there is no GUI, the Equivalence Check can be completed by confirming the SUCCEED message.

For StarRCXT, the location of the AutoPNR working folder and the design name are provided, along with the PVT conditions for extracting the parasitic RC. In this study, two scenarios were applied for AutoPNR. Since there are two PVT conditions, two separate folders were prepared to run StarRCXT, extracting spef files for the best operating condition and the worst operating condition, respectively. Figure 23 shows the results of the post-layout equivalence check and Figure 24 shows the contents of spef which results of the parasitic extraction.

```
************************************ Verification Results ************************************
Verification SUCCEEDED
   ATTENTION:  synopsys_auto_setup mode was enabled.
               See Synopsys Auto Setup Summary for details.
--------------------------------------------------------------
 Reference design:  r:/WORK/cmsdk_mcu
 Implementation design:  i:/WORK/cmsdk_mcu
 7137 Passing compare points
--------------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop    BBNet     Cut     Port     DFF      LAT    TOTAL
--------------------------------------------------------------------------------------------
Passing (equivalent)         926       0       36       0       36     6138        1     7137
Failing (not equivalent)       0       0        0       0        0        0        0        0
********************************************************************************************
```

Fig. 23. Result of the Post-layout Equivalence Check.

```
*SPEF "IEEE 1481-1999"
*DESIGN "cmsdk_mcu"
*DATE "Fri Apr 29 13:26:49 2022"
*VENDOR "Synopsys Inc."
*PROGRAM "StarRC"
*VERSION "N-2017.12-SP3"
*DESIGN_FLOW "PIN_CAP NONE" "NAME_SCOPE
 LOCAL"
*DIVIDER /
*DELIMITER :
*BUS_DELIMITER []
*T_UNIT 1.0 NS
*C_UNIT 1.0 FF
*R_UNIT 1.0 OHM
*L_UNIT 1.0 HENRY

// COMMENTS

// OPERATING_TEMPERATURE -40
// GLOBAL_TEMPERATURE 25
@
@
results/RC_MIN.spef  1,7
```

```
*D_NET *27791 4.8821

*CONN
*I *56440:Y O *D INV_X0P5B_A9TR *
590 2727.300
*I *56441:A I *C 502.777 2716.758

*CAP
1 *56440:Y 0.396015
2 *56441:A 0.120616
3 *27791:4 2.54938
4 *27791:5 0.392791
5 *27791:6 1.4233

*RES
1 *56440:Y *27791:2 0.001
2 *56440:Y *27791:4 157.33
3 *56441:A *27791:3 0.001
4 *56441:A *27791:5 36.0246
5 *27791:4 *27791:6 57.4926
6 *27791:5 *27791:6 11.03
*END
꼭 대기  results/RC_MIN.spef  1012043,1
```

Fig. 24. Contents of the SPEF file which results of Parasitic Extraction.

G. Post-layout STA and ECO

The tool used for Static Timing Analysis (STA) was Synopsys PrimeTime.

To perform post-layout STA, the design, SDC, and SPEF files are required. Since the SDC and SPEF files were extracted based on different operating conditions, separate folders were created for each scenario, and STA was performed in both folders. An important point is that even if no setup or hold violations were detected in the AutoPNR tool, such violations might still be found in PrimeTime.

If a hold violation is detected during this process, it is crucial to output an ECO file that can insert buffer cells using commands like **insert_buffer**, which is an ICC2-type command. In this study, hold violations were found in both STA folders, leading to the generation of two **eco_changes.tcl** files. After this, the AutoPNR environment was revisited to apply the ECO to the cells where route optimization had been completed.

After applying the ECO, equivalence check and parasitic

extraction need to be repeated, followed by another round of STA. It's important to note that this process may not be completed in one iteration, and excessive buffer insertion to resolve hold violations could result in setup time violations, requiring careful attention.

The final results of the post-layout STA are shown in Figure 25. In this study, since two scenarios were used, both results had to be satisfied.

```
Type of Check         Total              Met          Violated         Untested
-------------------------------------------------------------------------------
setup                  6804      6246 ( 92%)        0 (  0%)      558 (  8%)
hold                   6804      6246 ( 92%)        0 (  0%)      558 (  8%)
recovery               5934      2355 ( 40%)        0 (  0%)     3579 ( 60%)
removal                5934      2355 ( 40%)        0 (  0%)     3579 ( 60%)
min_period               65         3 (  5%)        0 (  0%)       62 ( 95%)
min_pulse_width       48298     24556 ( 51%)        0 (  0%)    23742 ( 49%)
clock_gating_setup        2         1 ( 50%)        0 (  0%)        1 ( 50%)
clock_gating_hold         2         1 ( 50%)        0 (  0%)        1 ( 50%)
out_setup                36        35 ( 97%)        0 (  0%)        1 (  3%)
out_hold                 36        35 ( 97%)        0 (  0%)        1 (  3%)
-------------------------------------------------------------------------------
All Checks            73915     41833 ( 57%)        0 (  0%)    32082 ( 43%)

(tal/ARM/cortex-m0/SS28/smkcow_make_S28/TOP/STA/POST/MAX/pt.log 25640,1          9
****************************************

Type of Check         Total              Met          Violated         Untested
-------------------------------------------------------------------------------
setup                  6804      6246 ( 92%)        0 (  0%)      558 (  8%)
hold                   6804      6246 ( 92%)        0 (  0%)      558 (  8%)
recovery               5934      2355 ( 40%)        0 (  0%)     3579 ( 60%)
removal                5934      2355 ( 40%)        0 (  0%)     3579 ( 60%)
min_period               65         3 (  5%)        0 (  0%)       62 ( 95%)
min_pulse_width       48298     24556 ( 51%)        0 (  0%)    23742 ( 49%)
clock_gating_setup        2         1 ( 50%)        0 (  0%)        1 ( 50%)
clock_gating_hold         2         1 ( 50%)        0 (  0%)        1 ( 50%)
out_setup                36        35 ( 97%)        0 (  0%)        1 (  3%)
out_hold                 36        35 ( 97%)        0 (  0%)        1 (  3%)
-------------------------------------------------------------------------------
All Checks            73915     41833 ( 57%)        0 (  0%)    32082 ( 43%)
```

Fig. 25. Result of 2nd sign off STA.

H. Post-layout Simulation and Power Estimation

The tools for simulation and power estimation are Synopsys VCS, Verdi and PrimePower.

The same testbench is used for function simulation, pre-layout simulation, and post-layout simulation. When the same test scenario is applied, the results should be identical across all simulations. For pre-layout simulation and the post-layout simulation, Standard Delay Format (SDF) annotation is required, which results in the annotation of cell and net delays for each signal. This can be observed when comparing the simulation waveforms, where delays are applied in the post-layout simulation compared to the function simulation.

Although delays are annotated, the design should function correctly without issues because it has passed STA.

Fig. 26 shows the results of the post-layout simulation of 2 scenarios, and Fig. 27 illustrates the synchronization between the function simulation and post-layout simulation, displaying how cell delay and net delay are applied at the same timing.

```
511667318 ns UART: Hello world
511669883 ns UART: smkcow
511675363 ns UART: ** TEST PASSED **
tb_cmsdk_mcu.v, 213 :    #32700 $stop;
ucli% exit
            V C S   S i m u l a t i o n   R e p o r t
Time: 514282700000 ps
CPU Time:   53548.400 seconds;       Data structure size:   14.5Mb
~
~
~
~/QnA/digital/ARM/cortex-m0/SS28/smkcow_make_S28/TOP/SIM/POST/max/
511667318 ns UART: Hello world
511669883 ns UART: smkcow
511675363 ns UART: ** TEST PASSED **
tb_cmsdk_mcu.v, 213 :    #32700 $stop;
ucli% exit
            V C S   S i m u l a t i o n   R e p o r t
Time: 514282700000 ps
CPU Time:   48826.620 seconds;       Data structure size:   14.5Mb
```

Fig. 26. Messages of the post-layout simulation.

Fig. 28 shows the results of power estimation, which represent average voltage estimates.

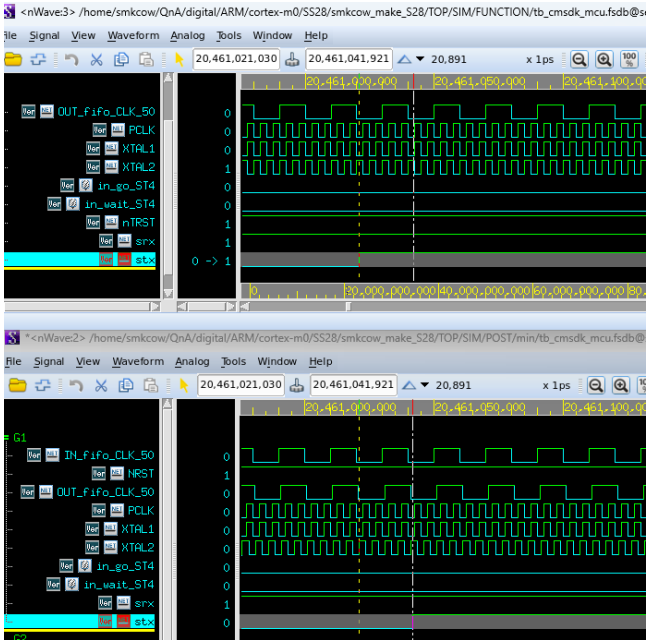

Fig. 27. Delays between function and post-layout simulation.

```
                          Internal  Switching  Leakage    Total
Power Group               Power     Power      Power      Power   (     %  Attrs
-------------------------------------------------------------------------------
clock_network          3.118e-03 5.863e-04 3.986e-08 3.704e-03 (57.55%)  i
register               6.538e-07 2.006e-07 2.454e-06 3.309e-06 ( 0.05%)
combinational          1.970e-06 2.240e-06 2.757e-06 6.967e-06 ( 0.11%)
sequential                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                 1.917e-05 4.033e-09 3.561e-05 5.479e-05 ( 0.85%)
io_pad                 2.081e-03 9.556e-06 5.774e-04 2.668e-03 (41.44%)
black_box                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 5.983e-04   ( 9.29%)
  Cell Internal Power  = 5.221e-03   (81.10%)
  Cell Leakage Power   = 6.183e-04   ( 9.60%)
                         ---------
Total Power            = 6.437e-03  (100.00%)

1
(M/cortex-m0/SS28/smkcow_make_S28/TOP/PrimePower/POST/MAX/reports/cmsdk_mcu_repor

                          Internal  Switching  Leakage    Total
Power Group               Power     Power      Power      Power   (     %  Attrs
-------------------------------------------------------------------------------
clock_network          3.116e-03 8.758e-04 3.986e-08 3.992e-03 (59.20%)  i
register               6.525e-07 2.997e-07 2.454e-06 3.407e-06 ( 0.05%)
combinational          1.966e-06 3.333e-06 2.757e-06 8.057e-06 ( 0.12%)
sequential                0.0000    0.0000    0.0000    0.0000 ( 0.00%)
memory                 1.918e-05 6.024e-09 3.561e-05 5.479e-05 ( 0.81%)
io_pad                 2.092e-03 1.435e-05 5.775e-04 2.684e-03 (39.81%)
black_box                 0.0000    0.0000    0.0000    0.0000 ( 0.00%)

  Net Switching Power  = 8.938e-04   (13.26%)
  Cell Internal Power  = 5.230e-03   (77.57%)
  Cell Leakage Power   = 6.184e-04   ( 9.17%)
                         ---------
Total Power            = 6.742e-03  (100.00%)
```

Fig. 28. Power consumption of 2 scenarios.

I. Merge and Physical DRC, LVS

The merge process involves replacing digital cells, which were previously worked on as phantom cells, with digital cells that possess real patterns. The detailed procedure is as follows:

After extracting the final design from ICC2 into a GDS file, it is streamed into Cadence Virtuoso tool. A Cadence library will be created during this process, into which the GDS files provided by the foundry are sequentially streamed in. Cells with the same names as the phantom cells will be overwritten with the real GDS file.

Once the standard cells, IO cells, and memory cells have been sequentially streamed in, the layout shows a design

where all the actual patterns can be verified. Following this, the OUTLINE and a team logo are added, and Physical DRC and LVS are performed. According to the rules provided by the foundry, Siemens Calibre is used. After the merge, various DRC errors, including antenna errors, may appear. Depending on the situation, these errors might need to be manually corrected through layout work. If there are too many errors, it may be necessary to return to ICC2 for further adjustments. It is important to be cautious during DRC corrections, as failing LVS could occur if the original patterns are not maintained. Figure 29 shows the result after merging performed by the Cadence Virtuoso tool [12].

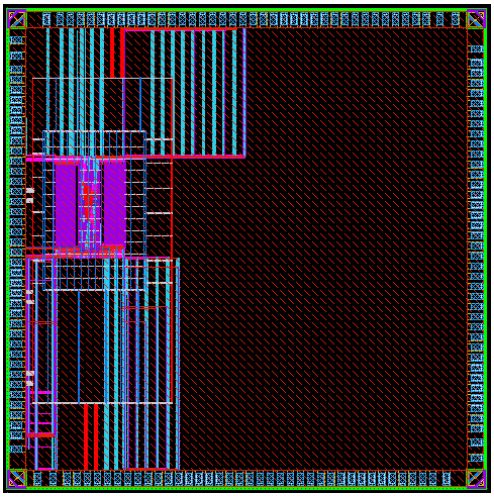

Fig. 29. The final design results of Physical DRC, LVS.

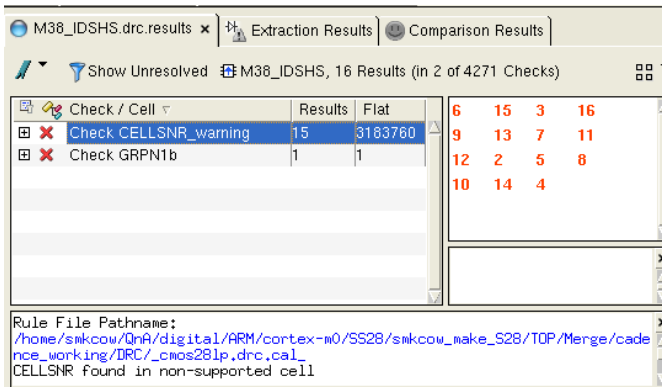

Fig. 30. The physical result of DRC.

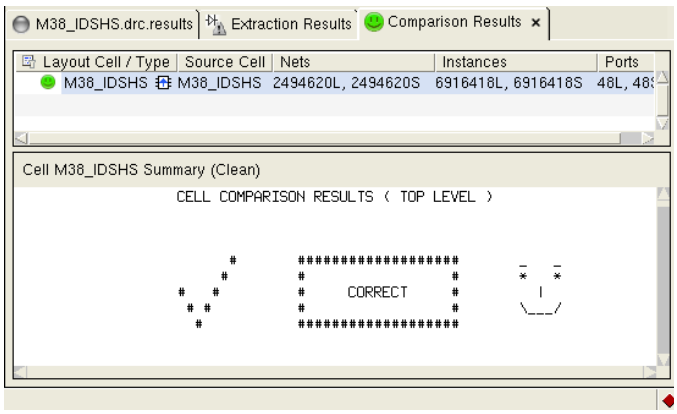

Fig. 31. The physical result of LVS.

Figs. 30 and 31 shows the final DRC and LVS results obtained using the Siemens Calibre [13]. The DRC results show errors that occur within the memory provided by the foundry and errors arising from the insertion of the team logo. Since these have been confirmed by the Samsung foundry, they were waived.

IV. RESULTS AND DISCUSSIONS

Fig. 32 shows the fabricated chip, which was packaged in an LQFP 208-pin package. It was mounted on a pre-designed test board with a socket, and power and signal connections were made before linking it to the measurement equipment. Various software codes were written to the ROM area via a UART cable. Fig. 33 shows the packaged chip and the test board and Fig. 34 shows the results of the cooperation with the software.

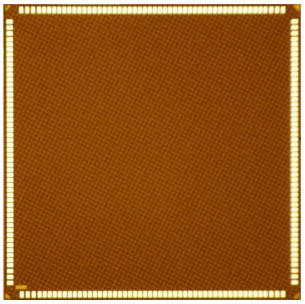

Fig. 32. The fabricated chip.

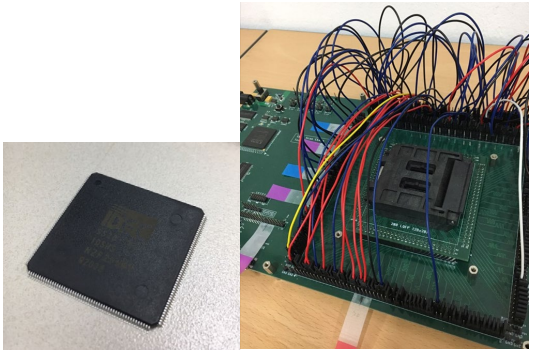

Fig. 33. The packaged chip and the test board.

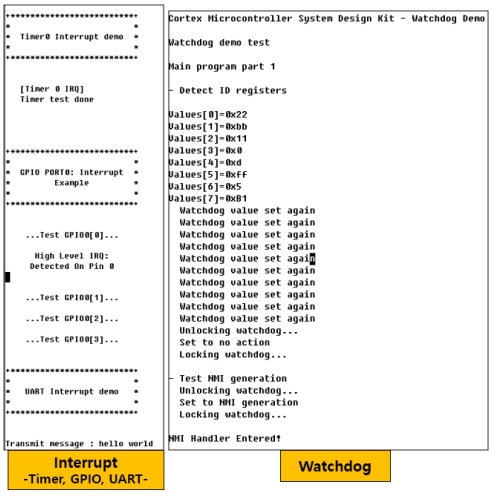

Fig. 34. The messages of co-working the software and the chip.

The specifications and results of this study are summarized in the following table.

TABLE I. The specifications and the results of this study

Parameters		This Work
Process		Samsung 28 LPP process
Main clock period		4ns
On chip memory		SRAM 16384x32
Core Area		600x800 um
Power Consumption		Averaged 6.55 mW (Includes leakage 0.6 mW)
Tool	Synopsys	VCS, VERDI, Library Compiler, Design Compiler, Formality, PrimeTime, ICC2, StarRCXT, PrimePower
	Cadence	Virtuoso
	Siemens	Calibre
	Samsung	MemoryCompiler

V . CONCLUSION

The implementation of the Cortex-M0 in this study was aimed at achieving the following objectives.

- Replace the behavioral model-based memory with the foundry's memory.
- Convert the ROM area into RAM and create a UART channel to enable software code to be written from a PC at any time.
- Although the Cortex-M0 provided by ARM targeted TSMC's 180nm process, implement a faster SoC using Samsung's 28nm process.
- Mount the implemented Cortex-M0 chip on a test board and demonstrate it along with the software.
- Implement the chip using as many tools as possible, following the guidelines provided by IDEC.

This study presents the above content with numerous figures. By implementing a chip using all the available tools, as demonstrated in this research, it is ensured that the chip will operate properly. It is hoped that all designers who read this paper will manage their schedules effectively, produce well-functioning chips, and achieve significant success in chip testing.

Through this study, the basic SoC has been verified and its operation confirmed. For future research, it would be appropriate to propose methods for easily integrating analog and digital IPs to facilitate platform development and to suggest ways to simplify the creation of device drivers within the software development environment.

ACKNOWLEDGMENT

This research was made possible through IDEC MPW chip manufacturing opportunity and EDA license support. Thanks to IDEC and its staff members.

REFERENCES

[1] ARM, Arm University Program. [Online]. Available : https://www.arm.com/resources/education/education-kits.

[2] ARM Tools and Software. [Online]. Available : https://developer.arm.com/Tools%20and%20Software/Keil%20PK166

[3] Synopsys, VCS and Verdi User guide. [Online]. Available : https://www.synopsys.com.

[4] Synopsys, Design Compiler User guide. [Online]. Available : https://www.synopsys.com.

[5] Synopsys, Formality User guide. [Online]. Available : https://www.synopsys.com.

[6] Synopsys, PrimeTime User guide. [Online]. Available : https://www.synopsys.com.

[7] Synopsys, PrimePower User guide. [Online]. Available : https://www.synopsys.com.

[8] Synopsys, IC Compiler 2 User guide. [Online]. Available : https://www.synopsys.com.

[9] Park, J. W., & Jeon, D. S. (2020). Designing Neuromorphic Processor with On-Chip Learning. *Journal of Integrated Circuits and Systems*, *6*(2).

[10] Synopsys, Solvnet. [Online]. Available : https://solvnet.synopsys.com.

[11] Synopsys, StarRCXT User guide. [Online]. Available : https://www.synopsys.com.

[12] Cadence, Virtuoso Layyout Editor User guide. [Online]. Available : https://www.cadence.com.

[13] Siemens, Calibre User guide. [Online]. Available : https://www.sw.siemens.com/en-US/

Hye-Seung Sun received his B.S. degree in information and communication engineering from Hanbat National University, Daejeon, Korea, in 2007 and received the M.S. degrees in computer engineering in from Hanbat National University, Daejeon, Korea, in 2009.

His research interest includes digital ASIC design flow. In particular, he is currently conducting the research on low power IC design for ARM Cortex SoC Platform.

In-Shin Cho received his B.S. degree in information and communication engineering from Hanbat National University, Daejeon, Korea, in 2005 and received the M.S. degrees in computer engineering in from Hanbat National University, Daejeon, Korea, in 2007.

His research interest includes analog ASIC design flow. In particular, he is currently conducting the research on low-power IC design.