Journal of
Electronics Technology Exploration

Homepage: htttps://shmpublisher.com/index.php/joetex

p-ISSN 3025-3470 | e-ISSN 3026-1066

Programming the 8031 Minimum System in Proteus Simulator using the C: Issues and Solutions

Elva Susianti[1], Putut Son Maria[2]

[1]Department of Electronic Systems Engineering Technology, Politeknik Caltex Riau, Indonesia

[2]Department of Electrical Engineering, Sultan Syarif Kasim State Islamic University of Riau, Indonesia

Article Info

Article history:

Received Jun 29, 2025
Revised Jun 30, 2025
Accepted Jul 09, 2025

Keywords:

Assembly code
C programming
Microprocessor
Minimum system
Proteus

ABSTRACT

An essential required course in electrical engineering, computer science, and informatics is the microprocessor. Students may consider using Proteus software in cases wherein microprocessor trainers are unavailable. Yet, the simulation of the 8031 microprocessor-based minimum system circuit that Proteus executes fails to operate correctly, despite the fact that the source code and circuit wiring comply to programming and circuit theory standards. This is in contrast to other microcontroller-based minimum system circuits that it can be simulated successfully and as intended. This research aims to get hints in programming the 8031 minimum system circuit simulated using Proteus. The problem was investigated and analyzed by observing the parameters that become the properties of each element in the circuit, especially the RAM, then comparing them with the specifications of the microprocessor. The experimental results showed that some adjustments on the program code were necessary either written using assembly language or C program code.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Putut Son Maria,
Teknik Elektro,
Universitas Islam Negeri Sultan Syarif Kasim Riau,
Pekanbaru, Riau 28293.
Email: putut.son@uin-suska.ac.id
https://doi.org/10.52465/joetex.v3i1.592

1. INTRODUCTION

Microprocessors are an essential part of various modern electronic devices and are one of the compulsory courses in electrical engineering, computer engineering, and computer science. In Indonesia, Microprocessor curriculum guidelines are written in the electrical engineering undergraduate study program document recommended by Fortei [1]. Through this course, students learn the internal structure of microprocessors, how to program in machine language (assembly), and how microprocessors interact with other hardware through input/output and interrupt systems. These subjects provide and prepare students to create IoT-based devices, build embedded systems, and gain a deeper understanding of computer architecture [2].

Learning microprocessors can be conducted either through direct hardware practice or simulation. [3]; nevertheless, in unfavorable conditions, when the trainer kits are unavailable to perform the exercise, the simulator can serve as a substitute, such as the Proteus [4], [5]. It can simulate minimum system circuits

consisting of microprocessors, memory, and supporting components that can be observed or evaluated to see their performance. Proteus is an interactive and animative computer program that has assisted many students learn how circuits work [6], [7] as well as a tool for designing electronic systems [8]. Students who do simulations using Proteus are proven to have better learning activities compared to students who only read the theory [9], [10].

However, programming a microprocessor necessitates a deep and thorough understanding because assembly language is used by default when programming a microprocessor, which can pose difficulties for students. The issue is that programming assembly is so particular language which takes more time and effort than other high-level languages like C [10]–[13]. Nonetheless utilizing the C language, it has been discovered that the simulation results using Proteus fail to correspond to the expected results when a control program is performed on a minimum system circuit.

The goal of this study is to provide experimental guidance on the best way to program the bare 8031 minimum system in C to ensure the program simulation's outcomes can be viewed as they should.

2. METHOD

In general, two types of minimum system circuits can be built, namely using microprocessors and microcontrollers. Minimum systems built using full-featured microcontroller chips are easy to use and implement. The control program can be stored internally in the chip, eliminating the requirement for external memory. However, such a minimum system is unable to be used as a material for learning to observe the contents of memory, learning about pointers, and learning data management techniques by indirect addressing in memory. Students need to know the role of external memory, especially in understanding the formation of a minimum system and the role of the elements that build it, which is the basis of a computer unit.

Minimum system – memory allocation

The minimum system circuit in this study was designed to consist of an 8031 microcontroller chip, 2732 chip and 6116 chip. The 8031 chip is a microcontroller without internal program memory(half-featured), so it is more appropriate to be called a dedicated microprocessor, therefore it must be wired with external memory as a program memory. Chip 2732 is a ROM (read-only memory) that will serve as program memory, storing codes executed by the 8031 microprocessor. Memory 6116 is a RAM (Random Access Memory) that will be used as a target for the write and read process in the experiments. The address segmentation design for each ROM and RAM is shown in Table 1. The minimum system circuit was sketched and simulated using Proteus with versions 7 and 8. The consistency of the best Proteus performance will be assessed by comparing the simulation outcomes.

Table 1. Memory address

Part Number	Address range	Capacity	Note
ROM 2732	0000 H – 0FFF H	4 KB	Program memory
RAM 6116	1000 H – 17FF H	2 KB	Data memory

Programming language dan compiler

Two (two) programming languages—Assembly and C—were used to program the microprocessor in this study. The identical program plot is implemented using Assembly and C language. Every program's code was simulated, and the output findings will be compared. Both source code were written and compiled using M-IDE Studio for MCS-51 Ver 0.2.6.0 [14]. The plot of the program is to perform a write-read-write task on RAM. The simulation is considered successful if the results show the output matches with the program algorithm. This is accomplished by observing Memory Contents on RAM 6116.

Algorithm

The program plot scenario is to work on the process of writing data to the RAM as much as 100 bytes started from the 1000 H address location, then the data are retrieved back from the RAM and saved in a variable. Data from the same variable were then loaded to a different address location 1090 H, since the display remains in the same window, this address selection is meant to make observation easier. Thus the task performed by the microprocessor is write-read-write on memory RAM. Figure 1 shows the plot diagram of the program tested in this study.

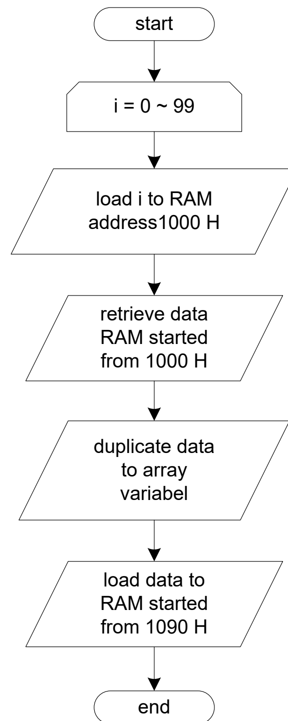

Figure 1. Task flowchart

Source code

Figure 2 presents how the assembly language program code implies the flowchart provided Figure 1. Data were loaded to address 1000H in lines 5 to 15, and then copied from external memory (RAM 6116) to the microprocessor's internal memory in lines 17 to 31. This process is indispensable because the 8031 microprocessor has only one 16 bit register (DPTR) that can be used for external indirect addressing operations, therefore, it is impossible to duplicate a number of data from one external memory location to a different location in the same chip using only one 16 bit register. Assembly language is highly dependent on the role of the DPTR register to access external memory or peripherals, therefore, despite its low capacity, internal memory is used as a buffer to overcome about this restriction. Lines 33 to 44 execute for copies of data from internal memory to external memory (RAM 6116) at address starting 1090 H.

.

```
org 0h
mov dptr, #1000h
mov a, #0

; task 1 menulis ke RAM
loop:
movx @dptr, a
cjne a, #100, up
sjmp task2

up:
inc a
inc dptr
sjmp loop
; end task 1

task2:
; task 2 read and copy
mov dptr, #1000h
mov r0, #10h
loop2 :
movx a, @dptr
mov @r0, a
;
cjne a, #100, next_addr
sjmp task3

next_addr :
inc dptr
inc r0
sjmp loop2

task3 :
mov dptr, #1090h
mov r0, #10h
loop3 :
mov a, @r0
movx @dptr, a
cjne a, #100, next_loc
sjmp fin
next_loc :
inc dptr
inc r0
sjmp loop3

fin:
mov p1, #0Fh
end
```

Figure 2. Assembly source code

```
#include<8051.h>

void main()
{
unsigned char __xdata *lokasi1 = 0x1000;
unsigned char __xdata *lokasi2 = 0x1090;

unsigned char i;
unsigned char array[100];


        //tulis ke RAM
        for(i = 0 ; i <= 100 ; i++)
        {
        *(lokasi1+i) = i;
        }

        //baca dari RAM
        for(i = 0 ; i <= 100 ; i++)
        {
        array[i] = *(lokasi1+i);
        }

        //tulis ke RAM(lokasi beda)
        for(i = 0 ; i <= 100 ; i++)
        {
        *(lokasi2+i) = array[i];
        }
}
```

Figure 3. C source code

Figure 3 illustrates the way the flowchart in Figure 1 is implemented in the C programming language. Data were stored to address 1000H in lines 13 through 16, and then copied to an array variable from external memory (RAM 6116) in lines 19 through 22. The C programming language is simpler since variables can be defined by the programmer as demanded, but the assembly language lacks recognition of variables. Prior to the data being written to the external memory, the array variable serves as a temporary buffer, then data were copied from the array to external memory in lines 25 to 28, beginning at location 1090 H. The wiring schematic for the bare 8031 minimum system used in this study is shown in Figure 4.

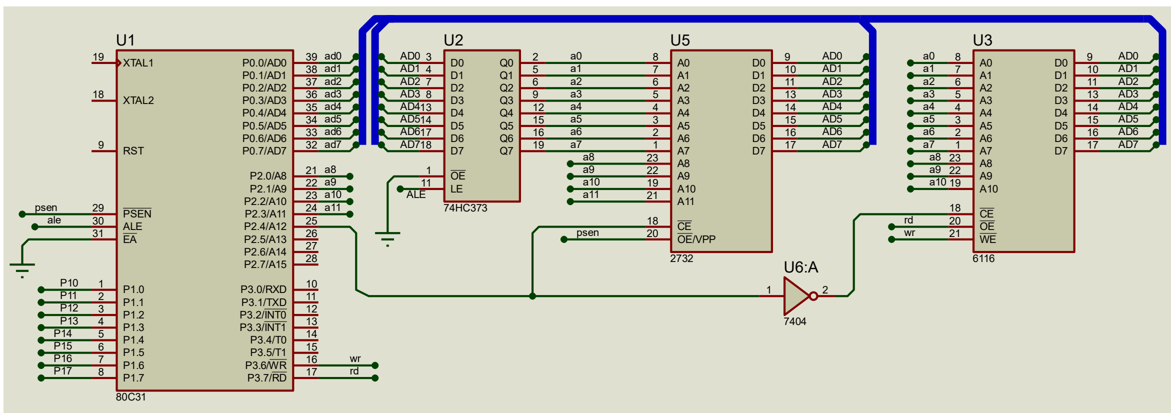

Figure 4. 8031 CPU based minimum system wiring diagram

3. RESULTS AND DISCUSSIONS

The ROM 2732 alternately stores the compiled binary files of the program code (Figure 2 and Figure 3), which are 49 bytes and 84 bytes, respectively, each time the circuit was simulated. The 6116 RAM memory should be loaded with data at address 000 H through 064 H in line with the program plot. In contrast, as Figure 5 illustrates, both assembly and C language applications experience data that fails to remain in the RAM memory as intended.

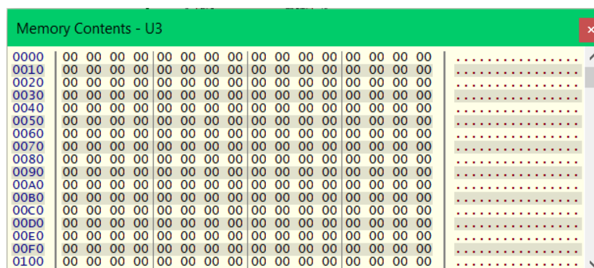

Figure 5. RAM 6116 contents

Algorithm and software compilation issues are not the cause of simulation failures. In order to obtain potential solutions to the problem, it is necessary to assess certain simulation settings. The parameters that need to be considered refer to the RAM 6116 (U3), where the default properties of Proteus consist of address access time (100 ns), minimum write pulse (80 ns), chip enable access time (100 ns), output disable time (35 ns) and output enable time (50 ns). The total time of these properties is around 365 nano seconds.

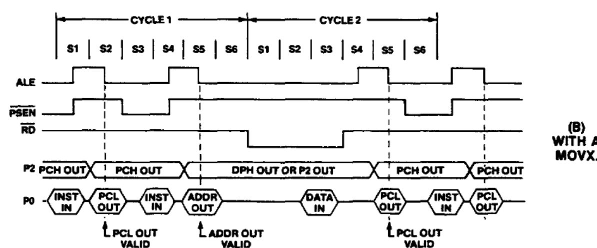

Figure 6. State sequence of MCS-51 CPU family [15]

Conversely, Figure 6 displays the timing diagram for the MCS-51 microcontroller family(includes the 8031 microprocessor). The instruction to access external memory (movx) is a 1 byte instruction, where the effective state only requires 1~2 states to access external memory (cycle 2; states 2-3). One state requires 2 clock oscillator periods or the equivalent of 166 nano seconds, so 2 states are equivalent to 330 nano seconds (with a clock frequency of 12 MHz). This means that the microprocessor's capabilities are faster than the default properties of memory or RAM. Eventhough Proteus offers a dialog to adjust the amount of RAM parameters, they were insufficiently accomplished in obtaining the simulation to function normally; conversely, reducing the microprocessor's clock frequency did not yield favorable outcomes.

The method employed in this investigation is to include instructions that prolong the time for access to RAM. The program language being used determines the technique. Successful assembly language strategies have included inserting one nop instruction and/or repeating the movx command twice. In C, an assembly language instruction must be positioned right after C instruction line in order to access external memory.

The code in Figure 2 has been modified to include the nop and movx instructions on lines 8, 24, and 41, which results in the code shown in Figure 7. The nop and movx instructions can be inserted right on the line after the instruction for external memory access without combining the two. Figure 9 depicts an illustration of the program's execution in Figure 7. Internal memory U1 (microprocessor 8051/8031) and RAM 6116 (U3) observations showed that the data were successfully stored and duplicated without alteration. This result alligns with the design of the program, which declares that the tasks of writing data (write) and reading data (read) should operate as intended. It is advised that the maximum buffer usage for data duplication be limited to 111 bytes, and that one should be careful never to exceed more than 128 bytes of internal memory as a buffer. When the program contains routines involving call and ret instructions, this number will decrease even further. Technically, when a function call instruction occurs, the microprocessor's internal memory will be utilized for

operations involving stack pointers, which store the value of the program counter. The more call instructions there are, the more internal memory space is required.

```
org 0h
mov dptr, #1000h
mov a, #0

; task 1 menulis ke RAM
loop:
movx @dptr, a
nop
cjne a, #100, up
sjmp task2

up:
inc a
inc dptr
sjmp loop
; end task 1

task2:
; task 2 read and copy
mov dptr, #1000h
mov r0, #10h
loop2 :
movx a, @dptr
movx a, @dptr
mov @r0, a
;
cjne a, #100, next_addr
sjmp task3

next_addr :
inc dptr
inc r0
sjmp loop2

task3 :
mov dptr, #1090h
mov r0, #10h
loop3 :
mov a, @r0
movx @dptr, a
movx @dptr, a
cjne a, #100, next_loc
sjmp fin
next_loc :
inc dptr
inc r0
sjmp loop3

fin:
mov p1, #0Fh
end
```

Figure 7. Assembly source code - post refinement

The program in Figure 3 gets modified to yield Figure 8, upon which assembly language instructions are inserted right after the instructions for external memory access, for example, on lines 16–18, 25–27, and 35–37. These instructions are meant to add a single line of movx commands, yet the asm keyword must be applied because the main code uses C language commands. The SDCC (Small Device C Compiler) compiler will retain the assembly language instructions literally as written in the program script. For C language instructions that retrieve data from external memory and transfer it into an array to work correctly, the movx command must be included because the assembling process file only includes one line of the movx command after compilation. The binary code sizes of the programs in Figures 7 and 8 are 52 bytes and 97 bytes, respectively. Using C language for the minimum system always results in a larger binary file size because every line of instruction (in C) involving variables will be transformed into 7 to 8 lines in assembly language. Thus, the binary code compiled from the C language requires more bytes (86-87%), depending on the number of lines and how frequently variables are invoked in the program code.

```
#include<8051.h>

void main()
{
unsigned char __xdata *lokasi1 = 0x1000;
unsigned char __xdata *lokasi2 = 0x1090;

unsigned char i;
unsigned char array[100];


        //WRITE TO MEMORY
        for(i = 0 ; i <= 100 ; i++)
        {
        *(lokasi1+i) = i;
        __asm
        movx @dptr, a
        __endasm;
        }

        //BACA DARI MEMORI
        for(i = 0 ; i <= 100 ; i++)
        {
        array[i] = *(lokasi1+i);
        __asm
        movx a,@dptr
        __endasm;
        array[i] = *(lokasi1+i);
        }

        //TULIS KE MEMORI ALAMAT BERBEDA
        for(i = 0 ; i <= 100 ; i++)
        {
        *(lokasi2+i) = array[i];
        __asm
        movx @dptr, a
        __endasm;
        }
}
```

Figure 8. C source code - post refinement

The distinction between the run results of the program code Figures 9 (assembly) and 10 (C) is that it turns out that the internal memory of the 8031 microprocessor contains duplicated data from RAM 6116 memory but occupies a different address offset. In Figure 9 the address offset of the duplicated data starts from 10 Hex because it had been specified in the program code (Figure 7 line 21), while in Figure 10 the address offset of the duplicated data starts from 0A Hex, this offset appears automatically without previously specified or written in the program. This also proves that variables or arrays in the C language are manifested to the internal memory in the microprocessor. If the space capacity in the internal memory reaches its limit, it is necessary to include external memory; otherwise, the running program will potentially malfunction. Knowing the basic architecture and how variables, pointers, and indirect addressing work are essential for the student to manage temporary memory while maintaining program functionality.

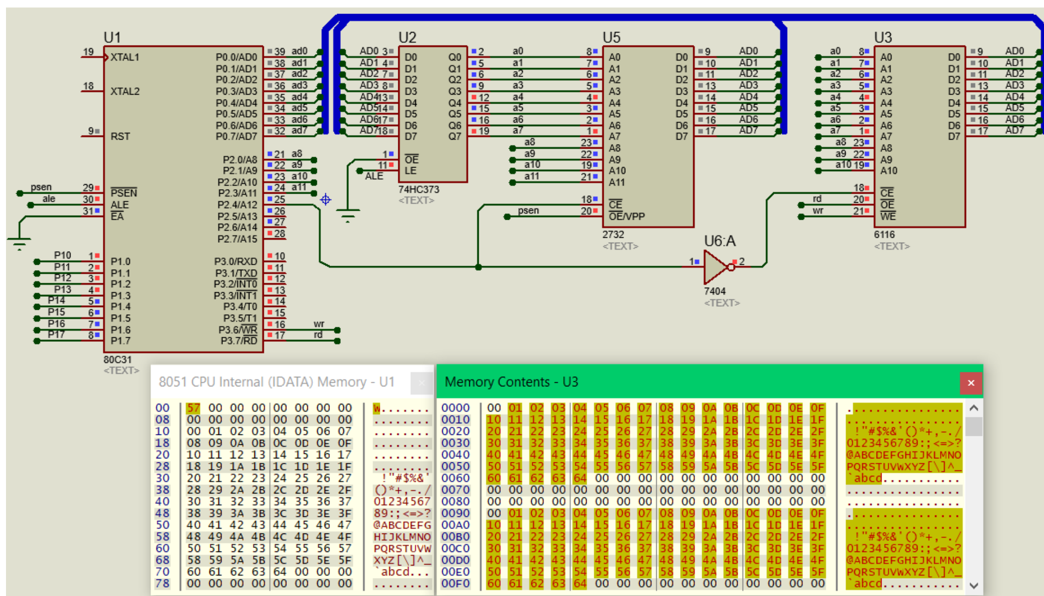

Figure 9. Simulation output – assembly code

.

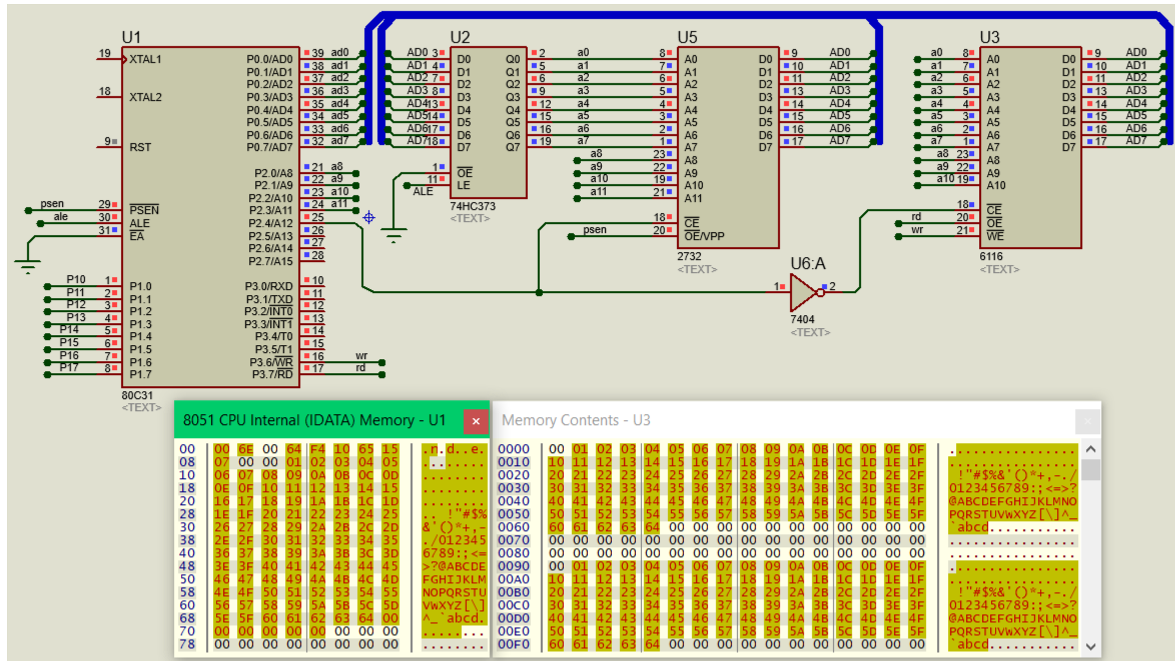

Figure 10. Simulation output – c code

4. CONCLUSION

Proteus can simulate the operational functions of the 8031 microprocessor-based minimum system circuit; however, it turns out that the program code must be adjusted depending on the programming language. If the program code uses assembly language, the 2 (two) tips that can be done are to insert a nop instruction after the movx command or repeat the movx instruction as many as 2 (two) lines in sequence. By comparison, the technique in C-language program code merges the assembly program code (movx instruction) that follows after the C-command. An alteration of 1 to 3 bytes in binary code for assembly language and 3 to 5 bytes for C language resulted from the adjustment, depending on the program plot. C is easy to compose but requires more memory space; compared to assembly language, it occupies memory space fairly effectively but it can be challenging to develop since it lacks structured. It is recommended that the results of this study be examined in order to implement program lines that replicate the 8031-based minimum system circuit, which appears feasible given the characteristics of the two programming languages.

REFERENCES

[1] D. A. dan Publikasi Ilmiah, "Panduan Penyusunan Kurikulum Program Studi Sarjana Teknik Elektro." 2023.

[2] W. A. Muchlis, N. B. A. K, and I. Istikmal, "Implementasi Alat Peraga Interkoneksi Mikrokontroler Dengan Input/output Pada Mata Kuliah Mikroprosesor," *e-Proceeding Eng.*, vol. 8, no. 3, p. 37029, Jun. 2021.

[3] F. M. Putra, M. Aulia, and A. Banjardana, "Perancangan kit trainer sebagai modul praktikum mikroprosesor untuk laboratorium Teknik Elektro Universitas Teknologi Sumbawa," *Renew. Energy Technol. J.*, vol. 1, no. 1, pp. 11–17, Dec. 2023.

[4] J. T. Elektro, F. Teknik, and U. N. Surabaya, "RANCANG BANGUN SIMULATOR MINIMUM SYSTEM MIKROPROSESOR 8088 BERBASIS ARDUINO UNO SEBAGAI MEDIA PEMBELAJARAN MIKROPROSESOR DI JURUSAN TITL SMK RADEN PATAH KOTA MOJOKERTO Randy Ade Anggara Nur Kholis," pp. 1037–1043.

[5] D. Artanto, E. Aris, B. Cahyono, and P. Arbiyanti, "Development of a Remote Three-Phase Motor Wiring Practice Tool Using ESP32, LabVIEW, Wokwi, and Adafruit IO (https://shmpublisher. com/index. php/joetex," *Repository.Usd.Ac.Id*, vol. 2, no. 2, pp. 33–42, 2024, [Online]. Available: https://repository.usd.ac.id/52972/1/12496_Artikel+Development+Development+of+a+Remote+Three-Phase+Motor+Wiring+Practice+Tool.pdf%0Ahttps://shmpublisher.com/index.php/joetex/issue/archive%0Ahttps://shmpublisher.com/index.php/joetex/article/view/434%0Ahttp

[6] T. Elektro *et al.*, "JOURNAL OF APPLIED SMART ELECTRICAL NETWORK AND SYSTEMS (JASENS) Overview Aplikasi Android dan Program Komputer Sebagai Simulator Rangkaian Listrik

Dengan Sumber Arus Bolak Balik,” vol. 4, no. 2, pp. 35–40, 2023.

[7] Ridwan, M. Nurmanita, and N. M. Sangi, “Efektivitas Pembelajaran Simulasi Proteus 8 Professional Berbantuan Virtual Laboratory untuk Meningkatkan Berpikir Kritis Mahasiswa Praktek Instalasi Listrik,” *J. Teach. Educ.*, vol. 3, no. 3, pp. 55–56, 2022.

[8] L. K. Hendinata, “Simulasi Sistem Transfer Daya Nirkabel Berbasis Kopling Magnetik,” *J. Appl. Smart Electr. Netw. Syst.*, vol. 2, no. 2, pp. 71–74, 2021, doi: 10.52158/jasens.v2i2.252.

[9] M. Mukminin and A. B. Santosa, “Pengaruh Media Pembelajaran Software Proteus pada Mata Pelajaran Penerapan Rangkaian Elektronika Terhadap Hasil Belajar Siswa Kelas XI Teknik Audio Video Di SMK Negeri 3 Surabaya,” *J. Pendidik. Tek. Elektro*, vol. 5, no. 1, pp. 147–154, 2016.

[10] Syahminan, “PENGEMBANGAN PEMBELAJARAN TEKNIK DIGITAL DENGAN MEDIA PERANGKAT LUNAK PROTEUS DAN EMULATOR JURUSAN TEKNIK INFORMATIKA UNIVERSITAS KANJURUHAN Syahminan 1),” *Nopember*, vol. 12, no. 2, pp. 41–45, 2020.

[11] S. Syahminan and C. W. Hidayat, “Development of digital engineering learning with proteus software media and emulators department of informatics engineering Kanjuruhan University,” *J. Phys. Conf. Ser.*, vol. 1869, no. 1, pp. 8–12, 2021, doi: 10.1088/1742-6596/1869/1/012076.

[12] P. Modus, P. Dan, A. Panjaitan, and M. Sagala, “Operasi Aritmatika Pada Mikroprosesor,” vol. 03, pp. 144–151, 2018.

[13] Friendly, “Perancangan Mikroprosesor 8 Bit Dengan Menggunakan Bahasa VHDL pada FPGA Xilinx Spartan 3,” *Teknovasi*, vol. 4, no. 1, pp. 10–27, 2017.

[14] OpCUBE, “M-IDE Studio for MCS-51 (versi ~0.2.5 – 0.2.6),” 2025, [Online]. Available: https://archive.org/details/midepack02518

[15] I. Corporation, “MCS-51 Microcontroller Family User’s Manual,” 1981, [Online]. Available: https://web.mit.edu/6.115/www/document/8051.pdf

.