EasySTEP: A Programming Platform for the PIC16F648A Microcontroller with Applications in Educational Robotics and Automation

Vitor Amadeu Souza [1]

[1] Department of Electrical Engineering
University Center of Volta Redonda (UniFOA)
Volta Redonda, RJ, Brazil
vitor.amadeu@foa.org.br
https://orcid.org/0009-0002-1857-6799

Abstract. *EasySTEP is an educational programming platform for the PIC16F648A microcontroller that allows programs written in a BASIC-like high-level language to be compiled, transmitted via RS232, and stored in the device's internal EEPROM for interpretation by an Assembly firmware. The AutoEasy IDE supports program creation, compilation, and uploading. Nine test programs validated core functionalities including digital I/O control, iterative and conditional structures, subroutines, serial communication, PWM control, and LCD interfacing. The system targets educational robotics and low-cost automation contexts.*

1. Introduction

Engineering and technology education has, over the past decades, sought tools that enable students to practically understand the principles of embedded systems, programming, and automatic control. In this context, microcontrollers play a central role, as they allow the direct integration of software and hardware within a single low-cost and highly versatile device [Mazidi et al. 2006]. The PIC family, developed by Microchip Technology, is one of the most widely used in educational and industrial applications worldwide. Among its devices, the PIC16F648A stands out as particularly suitable for low-complexity projects due to its compact 18-pin architecture and the availability of 256 bytes of internal EEPROM memory [Microchip 2007].

Direct programming of microcontrollers in low-level languages such as Assembly, although essential for the development of optimized firmware, presents a steep learning curve for beginners. Several educational platforms have been developed to mitigate this difficulty. The BASIC Stamp, produced by Parallax, pioneered this approach by employing a PBASIC interpreter resident in ROM, enabling programming through a simplified syntax and becoming an important reference in microcontroller-based education [Parallax 2023]. Similarly, the Arduino platform has greatly popularized educational programming through a simplified interface and a broad user community, being widely adopted in introductory courses on robotics and embedded systems [Arduino 2024, Plaza et al. 2018]. However, both BASIC Stamp and Arduino rely on microcontrollers with distinct architectures and instruction sets, and the availability of native simplified-language platforms for the PIC16F family remains limited.

Low-cost home automation projects also demonstrate the relevance of solutions based on accessible microcontrollers. Several studies in the literature describe home automation systems using platforms such as Arduino and other low-cost hardware, emphasizing affordability and ease of programming [Lopes et al. 2024, Da Silva and Perez 2013]. Nevertheless, these systems generally assume wireless connectivity and IoT protocols, which are not required in basic educational applications focused on local control.

Interpreter-based approaches for embedded systems have also been explored in different contexts, including lightweight scripting languages for constrained devices and bytecode interpreters for 8-bit architectures [Dunkels 2006, Khuntaweetep and Somkuarnpanit 2010, Zilli et al. 2015]. These works demonstrate the feasibility of implementing compact interpreters in low-resource environments, reinforcing the technical viability of the proposed solution.

EasySTEP is a system that combines an interpreter firmware, developed in Assembly for the PIC16F648A, with the AutoEasy IDE development environment. Through this integrated approach, it is possible to write programs in a high-level language with BASIC-like syntax, compile them, and transmit them to the microcontroller via RS232, where they are stored in EEPROM and executed by the interpreter. The objective of this paper is to present the system architecture, describe the development methodology, and demonstrate the results obtained from nine test programs covering the main functionalities provided by the platform.

2. Theoretical Background

The 8-bit microcontrollers of the PIC16 family, manufactured by Microchip Technology, are among the devices most widely used in low-complexity embedded systems. The PIC16F648A model features 4096 words of Flash memory for program storage, 256 bytes of RAM, 256 bytes of EEPROM dedicated to non-volatile data storage, and an internal 4 MHz oscillator. In addition, the device integrates a USART module, two analog comparators, and a Capture/Compare/PWM (CCP) module [Microchip 2007]. Taken together, these characteristics make the PIC16F648A particularly suitable for applications that do not require high processing power but demand flexibility and reliability in input and output interfaces.

EEPROM memory is a type of non-volatile memory that allows read and write operations through the program residing in the microcontroller itself. In the PIC16F648A, the EEPROM comprises 256 addressable bytes and can be accessed using the special function registers EEADR, EEDAT, and EECON1 [Microchip 2007]. This feature is fundamental to the operation of EasySTEP, as it enables programs to be permanently stored in the device without requiring reprogramming of the Flash memory.

The concept of resident interpreters in microcontrollers is not new in the literature. The BASIC Stamp, produced by Parallax, employs a PBASIC interpreter that translates source code into tokens and executes them directly, providing a simplified programming interface for beginners [Parallax 2023]. Similarly, the SIMPL project implements a minimalist serial interpreter for Arduino microcontrollers, in which each ASCII character functions as a bytecode command, enabling peripheral control through an extremely compact serial interface [Monsonite 2009]. These initiatives demonstrate that implementing

interpreters on resource-constrained devices is a well-established approach to democratizing access to embedded systems programming.

Asynchronous serial communication, implemented through the RS232 protocol, is one of the simplest and most universal interfaces for data transfer between a computer and a microcontroller. In the PIC16F648A, serial communication can be implemented using the internal USART module or by means of a software-based UART, employing timer interrupts to ensure the temporal precision required for the desired baud rate. The 9600 bps rate is commonly used in educational applications, as it provides an appropriate balance between speed and reliability [Mazidi et al. 2006].

The generation of PWM (Pulse Width Modulation) signals is a fundamental technique for controlling devices such as DC motors and LEDs. In the PIC16F648A, the CCP module enables hardware-controlled PWM signal generation, with frequency and duty cycle configurable via software. In educational robotics applications, motor speed control through PWM represents one of the most instructive demonstrations of hardware–software integration [Uzam 2014].

LCD displays based on the HD44780 controller, manufactured by Hitachi, are widely used in microcontroller-based projects due to their low cost and standardized interface. The 4-bit mode allows communication using only four data lines, thereby reducing the number of microcontroller pins required. The initialization sequence must be followed precisely to ensure proper display operation [Zhou and Yang 2015]. In the educational context, interfacing with LCD displays constitutes a valuable exercise for students, as it involves understanding parallel communication protocols and critical timing requirements.

3. Methodology

The development of EasySTEP was conducted in three main stages. The first stage consisted of the design and implementation of the interpreter firmware on the PIC16F648A, using the Assembly language and the MPLAB development environment provided by Microchip Technology. The firmware was structured into two distinct functional blocks: a bootloader responsible for receiving data via RS232 and storing it in EEPROM, and an interpreter engine responsible for sequentially reading the stored commands and executing them.

The bootloader implements a software-based UART, using interrupts to receive bytes at a rate of 9600 bps. When a program is sent from the AutoEasy IDE environment, the bootloader receives the bytes, writes them to the EEPROM memory, and signals the start of execution to the interpreter engine. The interpreter engine reads the commands from EEPROM byte by byte, decodes each opcode, and executes the corresponding instruction, using the PIC registers as working variables and the program counter as a virtual instruction counter. The instruction set supports more than one hundred opcodes, organized into categories including input and output control, timing, PWM, serial communication, LCD display interfacing, and flow control structures such as iterative loops (FOR/NEXT, WHILE/WEND, DO/LOOP UNTIL), conditionals (IF/THEN/ELSE), and subroutines (GOSUB/RETURN). Figure 1 shows the typical appearance of the PIC16F648A microcontroller in DIP package form.

The second stage involved the development of the AutoEasy IDE environment, a

Figure 1. PIC16F648A MCU [Microchip 2026].

graphical interface tool that allows users to write programs in the simplified EasySTEP language, compile them into tokens, and transmit them to the microcontroller via the serial port. The environment provides a set of commands that abstracts the low-level operations inherent to Assembly, enabling beginner programmers to interact intuitively with the PIC16F648A hardware. Figure 2 presents the integrated development environment (IDE), which, in addition to supporting code editing, also enables code simulation.

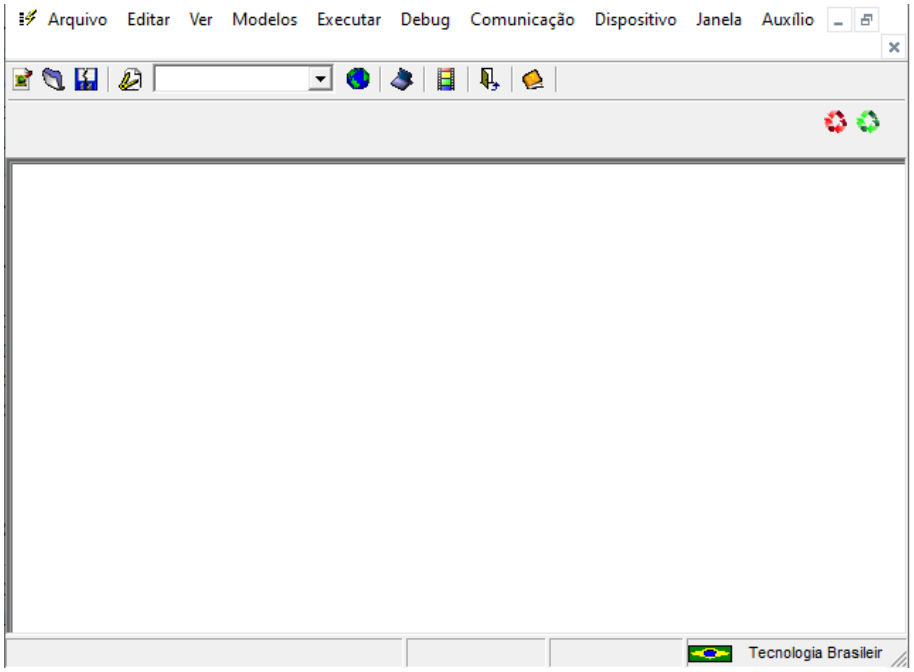

Figure 2. AutoEasy IDE Interface.

The third stage consisted of system validation through the execution of nine test programs, each designed to exercise specific functionalities of the platform. The programs were executed on a test circuit equipped with the PIC16F648A, LEDs, a 16×2 LCD display, a serial receiver, and input peripherals, with the microcontroller powered within the 3 to 5.5 V range, as specified in the device datasheet [Microchip 2007]. The results of each execution were observed and recorded based on the physical output of the peripherals connected to the system.

Algorithm 1 EasySTEP Interpreter Engine

Input: Program stored in EEPROM (bytes 0 to N-1)
Output: Execution of commands corresponding to the opcodes

$PC \leftarrow 0$ // virtual program counter (EEPROM index)
$stack\_top \leftarrow 0$ // top of the software stack
$stack[] \leftarrow$ software stack for return addresses $b[1..8], w[1..8] \leftarrow$ byte and word variables
while $PC < program\_size$ **do**
  $opcode \leftarrow EEPROM\_read(PC)$ $PC \leftarrow PC + 1$
  switch *opcode* **do**
    case *HIGH* **do**
      $pin \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ set_pin_high(pin)
    end
    case *LOW* **do**
      $pin \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ set_pin_low(pin)
    end
    case *DELAY_MS* **do**
      $value \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ wait_milliseconds(value)
    end
    case *FOR* **do**
      $limit \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ $counter \leftarrow 0$ $loop\_start\_address \leftarrow PC$
    end
    case *NEXT* **do**
      $counter \leftarrow counter + 1$ **if** $counter < limit$ **then**
        $PC \leftarrow loop\_start\_address$
      end
    end
    case *GOSUB* **do**
      $target\_address \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ $stack[stack\_top] \leftarrow PC$ $stack\_top \leftarrow stack\_top + 1$ $PC \leftarrow target\_address$
    end
    case *RETURN* **do**
      $stack\_top \leftarrow stack\_top - 1$ $PC \leftarrow stack[stack\_top]$
    end
    case *IF* **do**
      $condition \leftarrow evaluate\_condition\_from\_EEPROM(PC)$ $PC \leftarrow PC + condition\_size$ **if** $condition = FALSE$ **then**
        $PC \leftarrow jump\_to\_END\_IF(PC)$
      end
    end
    case *GOTO* **do**
      $target\_address \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ $PC \leftarrow target\_address$
    end
    case *PWM* **do**
      $value \leftarrow EEPROM\_read(PC); PC \leftarrow PC + 1$ configure_PWM(value)
    end
    case *DISPLAY* **do**
      $text \leftarrow EEPROM\_read\_string(PC)$ $PC \leftarrow PC + length(text) + 1$ send_to_LCD(text)
    end
    case *END_PROGRAM* **do**
      stop
    end
  end
end

4. Architecture and Firmware Algorithm of the Microcontroller

The general philosophy of the EasySTEP interpreter aligns with classical bytecode-based virtual machines, operating via a fetch–decode–execute cycle. Similar approaches have been explored in resource-constrained systems, such as *SCript* for MSP430 microcontrollers [Dunkels 2006] and bytecode firmware for 8-bit devices [Khuntaweetep and Somkuarnpanit 2010]. In EasySTEP, the 256-byte EEPROM stores the program, with the interpreter executing instructions directly from it. Serial reception is implemented via a software UART using TMR0 interrupts at 9600 bps, with each received byte written to EEPROM. A control byte signals the end of loading, triggering execution.

EEPROM interfacing follows the PIC16F648A specifications, with read/write operations using EEADR, EEDAT, and EECON1, and a mandatory unlock sequence (0x55, 0xAA) for writes [Microchip 2007]. The interpreter engine performs fetch, decode, and execute stages: opcodes are matched using `BTFSC`/`GOTO` sequences or `RETLW` tables, and operands for commands like `HIGH`, `LOW`, or `DELAY_MS` are read from subsequent EEPROM bytes. Byte (`b1--b8`) and word (`w1--w8`) variables are stored in RAM, enabling efficient arithmetic and logical operations. Flow control structures (`FOR/NEXT`, `WHILE/WEND`, `DO/LOOP UNTIL`, `IF/THEN/ELSE`) update the program pointer directly, and the `GOSUB/RETURN` mechanism uses a software stack due to the limited hardware stack [Mazidi et al. 2006].

Algorithm 1 presents a simplified pseudocode of the interpreter cycle, abstracting PIC-specific details such as FSR/INDF addressing and TMR0 interrupt management. EEPROM read speed (3 µs per byte at 4 MHz) and instruction cycles make execution of single-byte commands (`HIGH`, `LOW`) feasible within tens of microseconds, suitable for educational robotics applications [Valvano 2018, Zilli et al. 2015].

5. Results and Discussion

The nine test programs were executed successfully, confirming the correct operation of the different aspects of the EasySTEP platform. In order to enable the reader to understand the syntax and functionalities provided by the language, all programs are presented in full throughout this section, accompanied by comments on their structure and on the results observed during execution.

The first program, Code 01, was designed to demonstrate basic control of the microcontroller's digital outputs. As shown in Table 1, the program consists of only two instructions.

Table 1. Program Code 01

```
dirpin = %00000000  ; configures the pins as outputs
ios = %10101010  ; activates the I/O outputs
```

The instruction `dirpin` defines the direction of all port pins using an eight-bit binary value, in which each bit corresponds to one pin and the value zero indicates an output. The instruction `ios` directly assigns a binary pattern to the digital outputs, resulting in the alternating activation of the LEDs connected to the even-numbered pins.

Execution confirmed that the pin direction configuration system and output commands operate correctly, validating the most fundamental interface between the interpreter and the PIC16F648A input/output hardware.

The second program, Code 02, tested timing functionality by alternately switching all LEDs using the `DELAY_SEG` command and the unconditional jump structure `GOTO`, as shown in Table 2.

Table 2. Program Code 02

```
dirpin = %00000000  ; configures the pin direction as output
repete_:
  ios = %11111111  ; sets all I/Os to high level
  delay_seg(1)  ; waits 1 second
  ios = %00000000  ; sets all I/Os to low level
  delay_seg(1)  ; waits 1 second
  goto repete_  ; returns to the beginning of the loop
```

The program operates in a continuous cycle defined by the label `repete_` and the `goto` instruction, which performs an unconditional jump back to the beginning of the block. Between jumps, the `delay_seg` command suspends execution for one second, allowing visual observation of the LED alternation. Execution confirmed that the interpreter engine performs delay commands with sufficient accuracy for educational applications. This result is particularly relevant, since timing is one of the most fundamental operations in automatic control systems, and its correct implementation is essential for robotics applications [Uzam 2014].

The third program, Code 03, exercised the iterative control structure `FOR/NEXT`, representing the first example to present a more elaborate syntactic construction, as shown in Table 3.

Table 3. Program Code 03

```
dirpin = %00000000  ; configures the pin direction as output
for b1 = 1 to 10 step 1  ; loop from 1 to 10
  ios = %11111111  ; loads a value to the I/O output
  delay_seg(1)  ; waits 1 second
  ios = %00000000  ; loads a value to the I/O output
  delay_seg(1)  ; waits 1 second
next
```

The `FOR/NEXT` structure uses the control variable `b1`, whose value is initialized at 1, defined with an upper limit of 10, and incremented at each iteration by the step parameter. The loop body is delimited by the keywords `for` and `next` and contains the instructions that are executed repeatedly until the termination condition is reached. Observation confirmed that exactly ten cycles were completed, validating the implementation of the loop counter and the termination condition within the interpreter. Iterative structures such as `FOR/NEXT` are central to embedded systems programming, and their availability in the simplified EasySTEP language represents a significant improvement over direct Assembly programming.

Programs Code 04 and Code 05 tested, respectively, the `DO/LOOP UNTIL` and `WHILE/WEND` structures. Since both implement conditional iteration and produce similar external behavior, they are presented sequentially in Tables 4 and 5 to allow direct comparison of their syntax.

Table 4. Program Code 04

```
dirpin = %00000000  ; configures the pins as outputs
b1 = 0  ; initializes variable b1
do
  toggle  ; toggles the state of the I/O pins
  delay_seg(1)  ; waits 1 second
  toggle  ; toggles the state of the I/O pins
  delay_seg(1)  ; waits 1 second
  inc(b1)  ; increments variable b1
loop until b1 = 10  ; loop until b1 equals 10
```

Table 5. Program Code 05

```
dirpin = %00000000  ; configures the pins as outputs
b1 = 0  ; initializes variable b1
while b1 <> 10
  toggle  ; toggles the state of the I/O pins
  delay_seg(1)  ; waits 1 second
  toggle  ; toggles the state of the I/O pins
  delay_seg(1)  ; waits 1 second
  inc(b1)  ; increments variable b1
wend
```

Both programs use the variable `b1` as a counter and the `toggle` command to invert the state of the output pins at each iteration. The fundamental difference lies in the moment at which the condition is evaluated: in Code 04, the condition `b1 = 10` is verified at the end of the loop body, after the `inc` instruction, so the internal block is executed at least once before any evaluation occurs. In Code 05, the condition `b1 <> 10` is evaluated before executing the body, meaning that if the condition were false from the outset, the block would not be executed at all. This contrast represents a valuable didactic point for beginner students, as it highlights the semantic differences between iteration structures in BASIC languages [Parallax 2023].

The sixth program, Code 06, validated the subroutine mechanism through the `GOSUB` and `RETURN` commands, as shown in Table 6.

The main program defines a continuous cycle at the label `loop` and invokes three subroutines through the `gosub` command: `liga_led`, `desliga_led`, and `aguarda_tempo`. Each subroutine is defined by a label and terminated by the `return` command, which transfers control back to the calling point in the main program. The instruction `high 8` activates output 8, and the instruction `low 8` deactivates it, while the subroutine `aguarda_tempo` encapsulates the delay instruction. Execution confirmed that the subroutine addressing and return mechanism operates correctly, enabling program modularization in the EasySTEP language. The ability to use subroutines is essential for

Table 6. Program Code 06

```
dirpin = %00000000  ; configures the pins as outputs
loop:
  gosub liga_led  ; calls the liga_led routine
  gosub aguarda_tempo  ; calls the wait routine
  gosub desliga_led  ; calls the desliga_led routine
  gosub aguarda_tempo  ; calls the wait routine
  goto loop  ; returns to loop
liga_led:
  high 8  ; sets output 8 high
  return  ; returns
desliga_led:
  low 8  ; sets output 8 low
  return  ; returns
aguarda_tempo:
  delay_seg(1)  ; waits 1 second
  return  ; returns
```

the development of more complex and organized programs, and its implementation in the interpreter represents an intermediate-level feature that distinguishes EasySTEP from purely sequential tools.

The seventh program, Code 07, shown in Table 7, combines serial reception and PWM control through conditional structures.

Table 7. Program Code 07

```
dirpin = %00000000  ; configures the port direction
teste:
  if rxdata = "a" then  ; if a character was received...
    pwm(50)  ; configures PWM
  end if
  if rxdata = "b" then
    pwm(100)
  end if
  if rxdata = "c" then
    pwm(150)
  end if
  if rxdata = "d" then
    pwm(200)
  end if
  if rxdata = "e" then
    pwm(250)
  end if
  goto teste  ; returns to teste
```

The program operates in a continuous loop and, at each iteration, checks whether a character has been received via the serial port through the special variable `rxdata`. Each `IF/THEN/END IF` structure evaluates whether the received character corresponds to

one of the five predefined levels, from ”a” to ”e”, and, if so, configures the PWM duty cycle using the `pwm` command, whose parameter assumes the values 50, 100, 150, 200, and 250. Five distinct intensity levels were observed on an LED connected to the PWM output, confirming the feasibility of EasySTEP as a tool for projects involving integration between serial communication and analog control. This configuration is typical in educational robotics systems, where motor speed control is performed through commands sent by a host computer.

The eighth program, Code 08, tested the interface with a 16×2 LCD display based on the HD44780 controller, operating in 4-bit mode, as shown in Table 8.

Table 8. Program Code 08

```
dirpin = %00000001  ; configures the pin direction
inic display  ; initializes the display
display(1,1,"***Counter***")  ; shows a message on the display
loop:
  if io1 = 1 then  ; if the input is pressed
    inc(w1)  ; increments the variable
    display(2,5,w1)  ; shows the value on the display
    waitio1(0)  ; waits for the button to be released
  end if
  goto loop  ; returns to the loop
```

The program begins with the instruction `inic display`, which executes the initialization sequence required by the HD44780 controller, followed by the `display` instruction, which prints a string at a specific LCD position defined by the first two parameters, line and column. In the main loop, the input variable `io1` is evaluated through an `IF/THEN` structure: when the button connected to pin 1 is pressed, the word variable `w1` is incremented using the `inc` command, and its value is shown on the second line of the display. The command `waitio1` waits for the button to be released, preventing multiple increments per press. Execution confirmed that the display initialization sequence was performed correctly, that characters were shown in the expected positions, and that digital input reading operated properly. LCD interfacing is one of the most visually impactful demonstrations in educational microcontroller projects, and its implementation in the EasySTEP language reduces the programming complexity associated with the HD44780 protocol [Zhou and Yang 2015].

The ninth program, Code 09, tested the `PULSOUT` command, responsible for generating digital pulses with configurable duration. The program consists of a single instruction, as presented in Table 9.

Table 9. Program Code 09

```
pulsout(255,10)  ; generates 10 pulses of 255 ms each
```

The `pulsout` command receives two parameters: the first defines the duration of each pulse in milliseconds, and the second specifies the total number of pulses to be generated. In the tested program, ten consecutive pulses were generated, each with a duration of 255 ms, enabling observation of periodic pulse generation on an oscilloscope

connected to the corresponding output. The syntactic economy of this instruction exemplifies the EasySTEP design philosophy: an operation that would require dozens of lines in Assembly is expressed in a single line, without sacrificing the configurability required for robotics applications [Uzam 2014]. Pulse generation is a fundamental operation in the control of servomotors and in interfaces with devices that use pulse-width-based protocols.

The overall analysis of the results demonstrates that EasySTEP provides comprehensive coverage of the functionalities required for educational robotics and automation projects. The system allows students to progress from basic digital output control operations to more complex applications involving control structures, subroutines, serial communication, and interfacing with output peripherals such as LCD displays and PWM-controlled actuators. The interpreter-based architecture, with programs stored in EEPROM, offers the additional advantage of enabling microcontroller reprogramming without the need for a dedicated PIC programmer, thereby facilitating the development and debugging workflow in educational environments.

The source code for the microcontroller firmware, the AutoEasy IDE development environment, as well as the system manual and the programs developed in the AutoEasy language used in this work, are publicly available for access and download in a GitHub repository at the following address: `https://github.com/vitor-souza-ime/easystep`.

6. Conclusions

This article presented the development and validation of EasySTEP, an educational platform for programming the PIC16F648A microcontroller. The system uses an Assembly-based interpreter to execute programs written in a high-level BASIC-like language, stored in EEPROM via RS232. Results from nine test programs confirmed correct operation of key functionalities, including digital I/O control, iterative and conditional structures, subroutines, PWM generation, serial communication, and LCD interfacing.

EasySTEP shares similarities with solutions such as the BASIC Stamp and projects like SIMPL, while specifically leveraging the widely available PIC16F648A. Integration with the AutoEasy IDE provides an accessible programming environment, enabling beginners to focus on control logic without mastering microcontroller architecture or Assembly.

Future work may extend the command set to support analog sensors via ADC, I2C communication, arrays, and more complex data structures. A more advanced graphical interface in AutoEasy, with real-time debugging and variable visualization, could further enhance its educational utility.

References

PLAZA, Pedro et al. Arduino as an educational tool to introduce robotics. In: *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, IEEE, 2018. p. 1–8.

ARDUINO. Arduino Education, 2024. Available at: `https://www.arduino.cc/education/`. Accessed on: Feb. 1, 2026.

DA SILVA, E. G.; PEREZ, A. L. F. Aplicação de hardware de baixo custo na automação residencial. *Revista Técnico-Científica do IFSC*, p. 171, 2013.

ZHOU, Qing Fang; YANG, Jun. Design and Implementation of LCD Based on FPGA. *Applied Mechanics and Materials*, v. 738, p. 184–187, 2015.

LOPES, R. de J.; SOUZA, G. J. de; FERRAZ, D. de O.; SOUSA, A. L. de; MELO JÚNIOR, G. de. Desenvolvimento de bancada didática de automação residencial. *Cuadernos de Educación y Desarrollo*, v. 16, n. 6, e4587, 2024. Available at: https://doi.org/10.55905/cuadv16n6-158. Accessed on: Feb. 1, 2026.

MAZIDI, M. A.; CAUSEY, D.; McKINLAY, R. D. *PIC Microcontroller and Embedded Systems: Using Assembly and C for PIC18*. Upper Saddle River, NJ: Pearson Prentice Hall, 2006.

MICROCHIP TECHNOLOGY. *PIC16F627A/628A/648A Data Sheet*. DS40044F, 2007. Available at: https://ww1.microchip.com/downloads/en/devicedoc/40044e.pdf. Accessed on: Feb. 1, 2026.

MONSONITE. SIMPL: Serial Interpreted Microcontroller Programming Language. GitHub, 2009. Available at: https://github.com/monsonite/SIMPL. Accessed on: Feb. 1, 2026.

PARALLAX. PBASIC Programming with the BASIC Stamp, 2023. Available at: https://learn.parallax.com/programs/pbasic-programming-basic-stamp/. Accessed on: Feb. 1, 2026.

UZAM, M. *Building a Programmable Logic Controller with a PIC16F648A Microcontroller*. Boca Raton, FL: CRC Press, 2014. Available at: https://doi.org/10.1201/b15408. Accessed on: Feb. 1, 2026.

DUNKELS, Adam. *A low-overhead script language for tiny networked embedded systems*. Swedish Institute of Computer Science, 2006.

KHUNTAWEETEP, N. Jeenjun S.; SOMKUARNPANIT, S. Byte code Interpreter for 8051 Microcontroller. In: *Proceeding of the International Multi Conference of Engineers and Computer Scientists*, 2010. p. 17–19.

VALVANO, J. W. *Embedded Systems: Real-Time Interfacing to ARM Cortex-M Microcontrollers*. 4th ed. CreateSpace Independent Publishing Platform, 2018. Available at: http://users.ece.utexas.edu/~valvano/. Accessed on: Feb. 1, 2026.

ZILLI, Massimiliano et al. Hardware/software co-design for a high-performance Java Card interpreter in low-end embedded systems. *Microprocessors and Microsystems*, v. 39, n. 8, p. 1076–1086, 2015.

MICROCHIP TECHNOLOGY. PIC16F648A [imagem do microcontrolador]. Available at: https://www.microchip.com/en-us/product/pic16f648a. Accessed on: Feb. 1, 2026.