|  |
| --- |
| **External Project Report on**  **Computer Organization and Architecture**  **(EET 2211)** |

**Design a Unit Converter using ARM32 assembly language.**



**Submitted by**

|  |  |
| --- | --- |
| **Name 01- SAI SMRUTI MOHARANA** | **Reg. No.:2241013383** |
| **Name 02- SAURAV SINGH MOHANTA** | **Reg. No.:2241013209** |
| **Name 03- KHUSHI NIGAM** | **Reg. No.:** **2241014154** |
| **Name 04- RATIRANJAN NAYAK** | **Reg. No.:** **2241014112** |

**B. Tech. CSE 4th Semester (Section - 2241029)**

# Declaration

We, the undersigned students of B. Tech. of **Computer Science And Engineering** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “Design a Unit Converter using ARM32 assembly language”, submitted to **Siksha ‘O’ Anusandhan Deemed to be University, Bhubaneswar** for the partial fulfillment of the subject **Computer Organization and Architecture (EET 2211)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

|  |  |
| --- | --- |
| **SAI SMRUTI MOHARANA**  **Registration No.:2241013383** | **SAURAV SINGH MOHANTA**  **Registration No.:2241013209** |
| **KHUSHI NIGAM**  **Registration No.: 2241014154** | **RATIRANJAN NAYAK**  **Registration No.:** **2241014112** |

**DATE:**  21.05.2024

**PLACE:**  BHUBANESHWAR

# Abstract

This project presents the design and implementation of a unit converter using ARM32 assembly language, aiming to demonstrate the practical application of low-level programming in solving real-world problems. The converter supports various units of measurement, including length (meters, kilometers), and temperature (Celsius, Fahrenheit). By leveraging the ARM32 architecture, the program optimizes performance and resource usage, essential for embedded systems and low-power devices. The conversion algorithms are implemented using fundamental assembly instructions . Key features include efficient memory management, modular code structure, and the ability to add additional unit conversions with minimal modifications. The project showcases the strengths of ARM32 assembly in executing precise and efficient calculations, while also highlighting the challenges of assembly language programming, such as debugging and managing low-level hardware interactions.

# 

# Contents

|  |  |  |  |
| --- | --- | --- | --- |
| Serial No. | Chapter No. | Title of the Chapter | Page No. |
|  | 1 | Introduction | 05 |
|  | 2 | Problem Statement | 06 |
|  | 3 | Methodology | 07-09 |
|  | 4 | Implementation | 10-11 |
|  | 5 | Results and interpretation | 12-14 |
|  | 6 | Conclusion | 15 |
|  |  | References | 16 |
|  |  | Appendices | 17-19 |

# **1.Introduction**

Designing a unit converter using ARM32 assembly language involves leveraging the capabilities of ARM architecture to create an efficient and functional program for converting between different units of measurement. This project aims to develop a versatile and user-friendly tool that can handle various conversion tasks, such as converting between temperature units, length units, and more. The unit converter will be designed to run on ARM-32, offering portability and flexibility.

The unit converter will consist of several key components, including input handling, conversion logic, user interface, and output display. The design process will focus on optimizing code efficiency, ensuring accuracy in conversions, and providing a seamless user experience.

The input handling module will be responsible for receiving user input, including the value to be converted and the units of measurement. This module will include functionalities for input validation to ensure that the user provides valid input data.

The conversion logic module will contain the algorithms and formulas necessary to perform unit conversions. This module will utilize ARM assembly instructions to execute arithmetic operations efficiently. Conversion factors for different units of measurement will be defined within this module, allowing for seamless and accurate conversions between different units.

The user interface module will provide an intuitive interface for interacting with the unit converter.

Here the output will be displayed in the registers whose description is stated in comments, the output is sequential and could be used to make further conversions.

# **2. Problem Statement**

Design and implement a Unit Converter program using ARM32 assembly language. The program should be able to convert between different units of measurement in various categories such as length and temperature.

The converter should support common units within each category, such as meters, Celsius, Fahrenheit. The user should be able to input a value to be converted and select the source and target units. The program should then perform the conversion and display the result onto the registers.

The Unit Converter should be modular and extensible, allowing for easy addition of new units and categories in the future. It should utilize efficient algorithms for conversion calculations to ensure optimal performance on ARM32 architecture.

The user interface should be user-friendly and intuitive, providing clear instructions and options for inputting values and selecting units with suitable comments.

Additionally, the program should adhere to ARM32 assembly language conventions and best practices, utilizing appropriate data structures and control flow mechanisms for efficient and maintainable code.

The Unit Converter should be thoroughly tested to ensure correctness and reliability in all supported scenarios. Unit tests should cover various input values, unit combinations, and edge cases to validate the accuracy of the conversion logic.

Overall, the Unit Converter should provide a convenient tool for users to perform quick and accurate conversions between different units of measurement, enhancing their productivity and usability of ARM32-based systems.

# **3. Methodology**

Designing a unit converter using ARM32 assembly language involves several key steps to ensure a robust and efficient implementation. In this 800-word guide, I'll outline a comprehensive methodology for designing such a unit converter.

Step 1: Requirements Gathering

Begin by clearly defining the requirements of your unit converter. Identify the units you want to support conversion between (e.g., length, temperature), and determine the desired precision and range for each conversion. Consider the user interface and input/output methods as well.

Step 2: Architectural Design

Define the overall architecture of your unit converter. Decide whether it will be a console-based application, a graphical user interface (GUI), or a web application. Determine the input and output formats, as well as any necessary data structures for storing conversion factors and user input.

Step 3: Unit Conversion Logic

For each pair of units you want to convert between, research and define the conversion formulas. This may involve simple arithmetic operations or more complex mathematical functions. Implement these conversion formulas using ARM32 assembly language instructions, ensuring accuracy and efficiency.

Step 4: Input Handling

Design a mechanism for accepting user input. This could be through a command-line interface, text input fields in a GUI, or HTML forms in a web application. Implement input validation to handle invalid input gracefully and prevent runtime errors.

Step 5: Output Display

Decide how you will present the converted output to the user. This could be through console output, text displayed in a GUI window, or dynamically generated HTML in a web application. Implement the necessary code to format and display the output appropriately.

Step 6: Error Handling

Plan for error handling and edge cases. Consider scenarios such as division by zero, out-of-range input values, or unsupported unit conversions. Implement error-checking mechanisms to detect and handle these situations, providing informative error messages to the user.

Step 7: Testing and Validation

Thoroughly test your unit converter to ensure correctness and reliability. Test various input scenarios, including boundary cases and edge conditions. Validate the accuracy of the conversions against known reference values. Address any bugs or issues discovered during testing.

Step 8: Optimization

Optimize your ARM32 assembly code for performance and efficiency. Look for opportunities to reduce the number of instructions, minimize memory access, and eliminate unnecessary calculations. Profile your code to identify bottlenecks and areas for improvement.

Step 9: Documentation

Document your design and implementation thoroughly. Provide clear explanations of the conversion formulas used, the input and output formats supported, and any limitations or constraints of the unit converter. Include code comments to aid understanding and maintenance.

Step 10: Maintenance and Updates

Plan for future maintenance and updates to your unit converter. Consider how you will handle additions of new units or conversion types, improvements to the user interface, and bug fixes. Maintain a version-controlled repository of your code for easy collaboration and tracking of changes.

By following this methodology, you can design a robust and efficient unit converter using ARM32 assembly language. Paying careful attention to each step ensures that your unit converter meets the requirements, performs accurately, and provides a seamless user experience.

1. **Implementation**

**.text**

**.global main**

**main:**

**@ Convert kilometers to meters**

**MOV R0,#0x10000000**

**LDRB r0, [R0] @ Example: 5 kilometers**

**ldr r1, =#0x00001000**

**mul r2, r0, r1 @ (1 kilometer = 1000 meters)**

**@ Convert meters to centimeters**

**mov r3,r1@ (1 meter = 100 centimeters)**

**lsl r3,r3,#8@When we shift a binary number left by 8 bits, we are effectively**

**@multiplying it by 2^8=256. In decimal terms, this is equivalent to multiplying by 256.**

**mul r3,r3,r0**

**@ Convert centimeters to milimeters**

**mov r4,r1@ (1 centimeters = 100 milimeters)**

**lsl r4,r4,#12 @When we shift a binary number left by 12 bits, we are effectively**

**@multiplying it by 2^12=4096. In decimal terms, this is equivalent to multiplying by 4096.**

**mul r4,r4,r0**

**@ Convert milimeters back to kilometers**

**mov R5,R4**

**lsr r5,r5,#24 @When we shift a binary number left by 24 bits, we are effectively**

**@DIVIDING it by 2^24. In decimal terms, this is equivalent to DIVIDING by 2^24.**

**@ Convert celsius to fahrenheit**

**ldr r6,=#2@(F = (9/5)C + 32.),taking 9/5 as approx 2,storing it in r6**

**mul r6,r6,r0 @(performing (9/5)C ),storing it in r6**

**add r6,r6,#32@(performing +32),storing it in r6**

**@ End of program**

**MY\_EXIT: B MY\_EXIT**

**@here we are converting km to m to cm to mm and then back to km**

**@register0=value to be converted**

**@register1=1000**

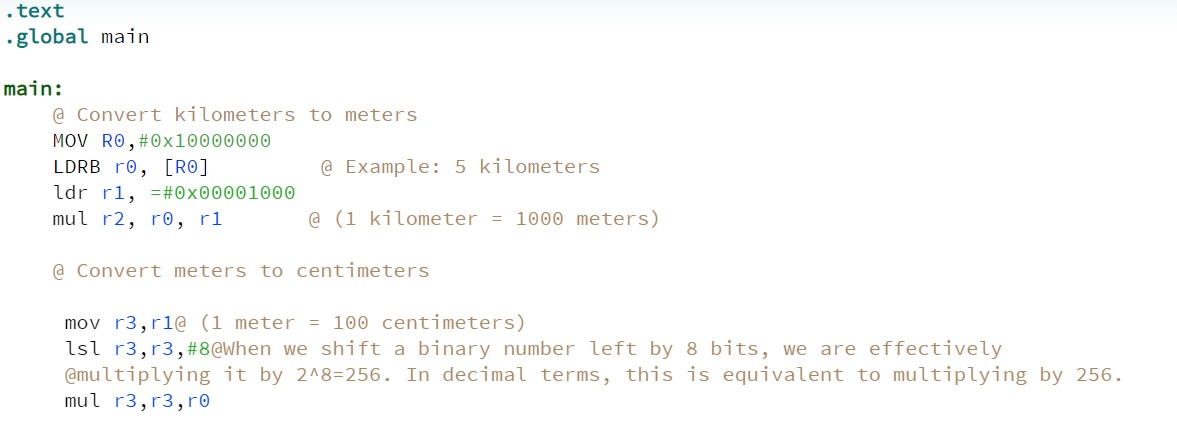
**@register2=initial value in m**

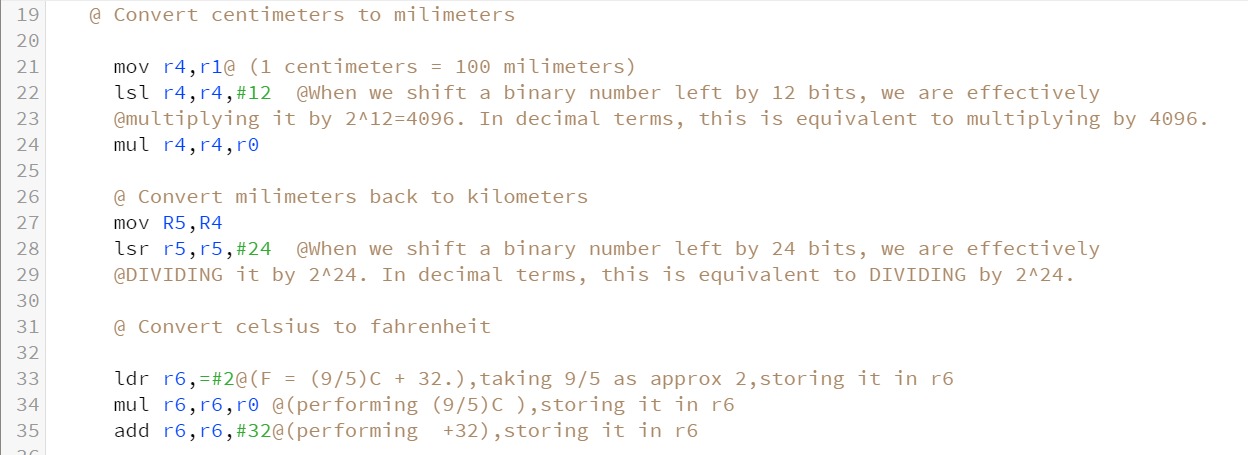
**@register3=initial value in cm**

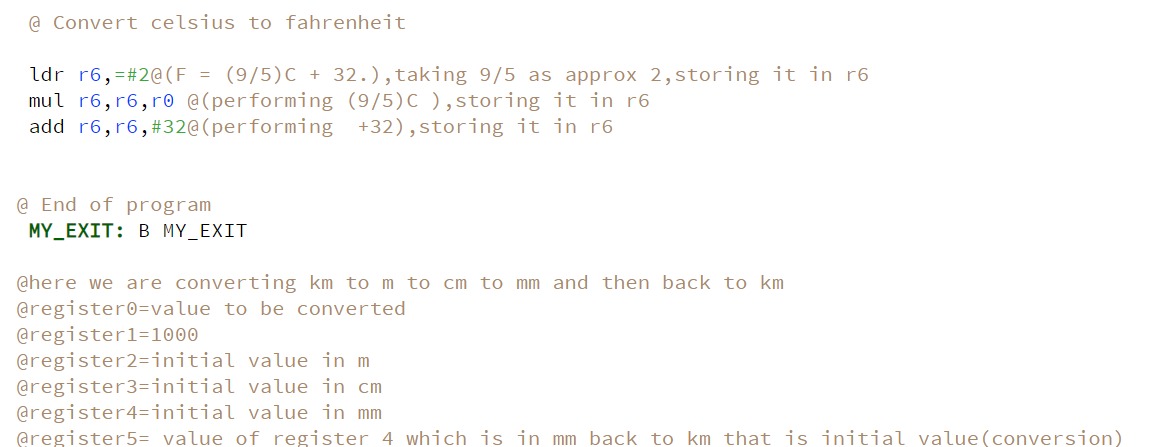
**@register4=initial value in mm**

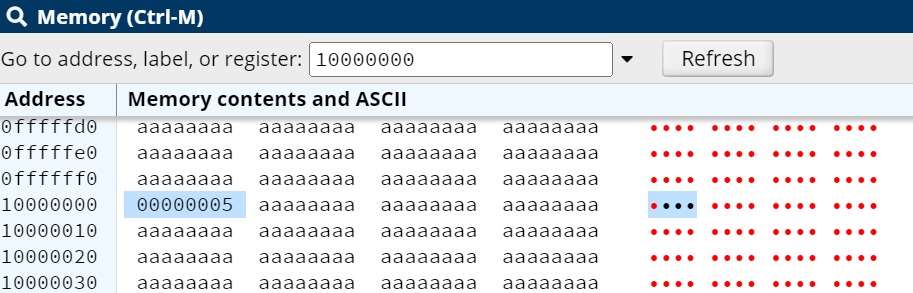
**@register5= value of register 4 which is in mm back to km that is initial value(conversion)**

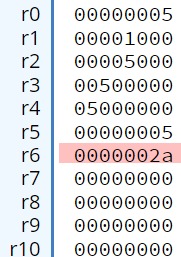
# **5. Results & Interpretation**

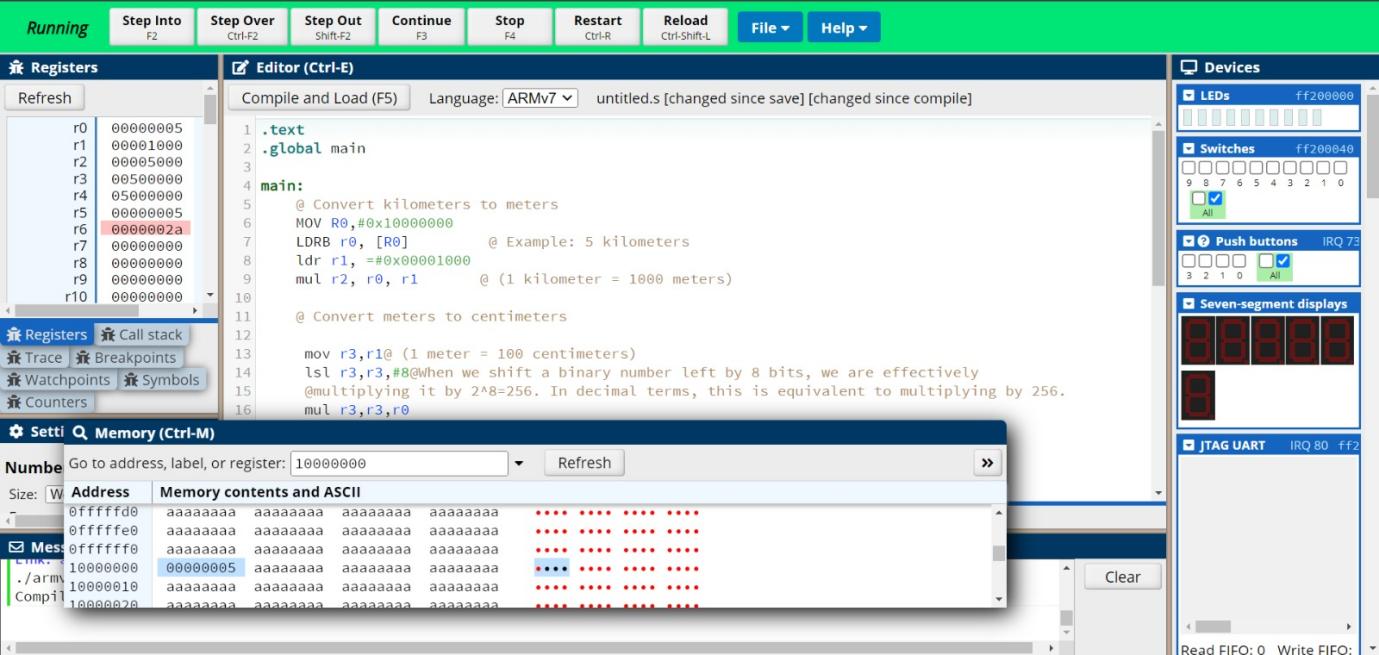
****

****

****

****

****

****

**6. Conclusion**

Designing a unit converter using ARM32 assembly language involves a structured approach that integrates input handling, conversion logic, and output display. Through a systematic methodology, the converter can efficiently process user requests and provide accurate results. The conclusion of such a design encapsulates the significance of the project, its implementation challenges, and potential future enhancements.

Unit converters serve as invaluable tools in various domains, facilitating seamless conversion between different measurement units. Developing a unit converter using ARM32 assembly language offers several advantages, including direct hardware access, optimized performance, and low-level control over system resources. By harnessing the power of assembly language, developers can create efficient and lightweight converters suitable for resource-constrained environments.

During the design process, careful consideration is given to user interaction, ensuring intuitive input mechanisms and informative output displays. Input validation mechanisms are implemented to handle erroneous inputs gracefully, enhancing the converter's robustness and user experience. The core conversion logic is meticulously crafted using ARM assembly instructions, leveraging arithmetic operations to apply conversion formulas accurately.

One of the primary challenges in designing a unit converter using ARM32 assembly is managing the complexity of conversion formulas and ensuring precision in arithmetic calculations. Additionally, optimizing code efficiency and minimizing memory footprint are crucial considerations, especially in embedded systems where resources are limited. Thorough testing and debugging procedures are essential to validate the correctness and reliability of the converter across a diverse range of input scenarios.

Despite the inherent challenges, the flexibility of ARM assembly language enables developers to extend the unit converter's functionality with ease. Future enhancements may include support for additional measurement units, customizable conversion settings, and integration with graphical user interfaces for enhanced usability. Furthermore, optimizing code performance through algorithmic improvements and leveraging advanced ARM architecture features can further enhance the converter's efficiency and versatility.

# 

# **References**

1. ARM Limited. "ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition." ARM Holdings, 2011.

2. Irvine, Kip R. "Assembly Language for x86 Processors." Pearson, 2018.

3. Brey, Barry. "The Intel Microprocessors." Pearson, 2014.

4. "IEEE Standard for Binary Floating-Point Arithmetic." IEEE Std 754-2019, IEEE, 2019.

5. Maguire, Steve. "Computer Organization and Design Fundamentals." Springer, 2006.

6. SoC Design with ARM (Electrical Engineering and Computer Science)." Springer, 2007.

7. Patterson, David A., and John L. Hennessy. "Computer Organization and Design: The Hardware/Software

Interface." Morgan Kaufmann, 2017.

# **Appendices**

***DE1-SoC***

***Development Kit***

The DE1-SoC Development Kit presents a robust hardware design platform built around the Altera System-on-Chip (SoC) FPGA, which combines the latest dual-core Cortex-A9 embedded cores with industry-leading programmable logic for ultimate design flexibility. Users can now leverage the power of tremendous re-configurability paired with a high-performance, low-power processor system. Altera’s SoC integrates an ARM-based hard processor system (HPS) consisting of processor, peripherals and memory interfaces tied seamlessly with the FPGA fabric using a high-bandwidth interconnect backbone. The DE1-SoC development board includes hardware such as high-speed DDR3 memory, video and audio capabilities, Ethernet networking, and much more.

The DE1-SoC Development Kit contains all components needed to use the board in conjunction with a computer that runs the Microsoft Windows XP or later.

## 

## The DE1-SoC package contents

The DE1-SoC package includes:

* The DE1-SoC development board
* DE1-SoC Quick Start Guide
* USB Cable (Type A to B) for FPGA programming and control
* USB Cable (Type A to Mini-B) for UART control
* 12V DC power adapter

### **1.2 DE1-SoC System CD**

The DE1-SoC System CD contains all DE1-SoC documentation and supporting materials, including the User Manual, System Builder, reference designs and device datasheets. User can download this System CD from the link: [http://de1-soc.terasic.com.](http://de1-soc.terasic.com/)

**1.3 Getting Help**

Here are the addresses where you can get help if you encounter any problems:

* Altera Corporation
* 101 Innovation Drive San Jose, California, 95134 USA

Email: university@altera.com

* Terasic Technologies
* 9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan

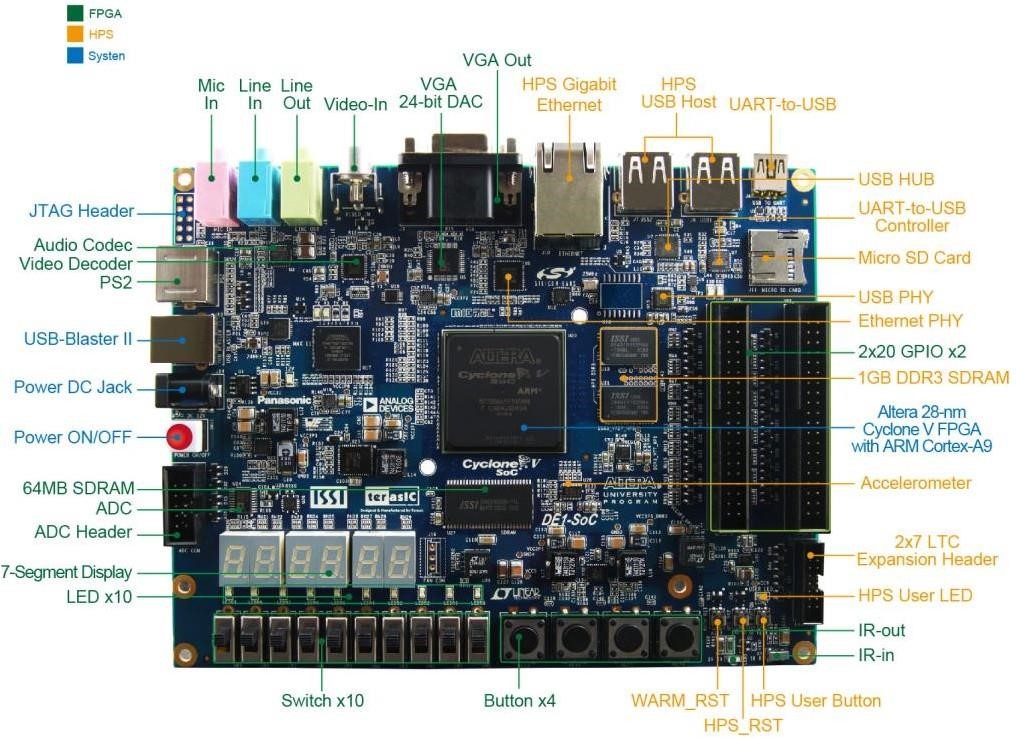
Email: support@terasic.com

Tel.: +886-3-575-0880

Web: [de1-soc.terasic.com](http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836)

**Layout And Components**

A photograph of the board is shown in **Figure 2-1.** It depicts the layout of the board and indicates the location of the connectors and key components.



**Figure 2-1 Development Board (top view)**