CEC Continuing Education Center

DesignNews

Expert C Techniques to Master Baremetal Programming

DAY 4 : Assertions and Validation Techniques for Baremetal Programming

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.

THE SPEAKER

Jacob Beningo

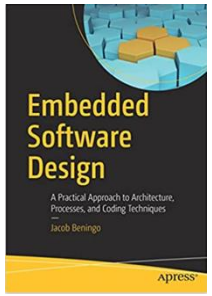
Jacob@beningo.com

jacobbeningo

Beningo Embedded Group – CEO / Founder

Focus: Software Architecture, Processes, and Dev Skills

At Beningo Embedded Group, we believe everyone deserves the skills to confidently advance their careers, meet deadlines, and deliver quality embedded systems. We provide modern strategies, insights, and hands-on training to equip developers and teams with the tools they need to succeed.



EMBEDDED SOFTWARE ACADEMY BY BENINGO

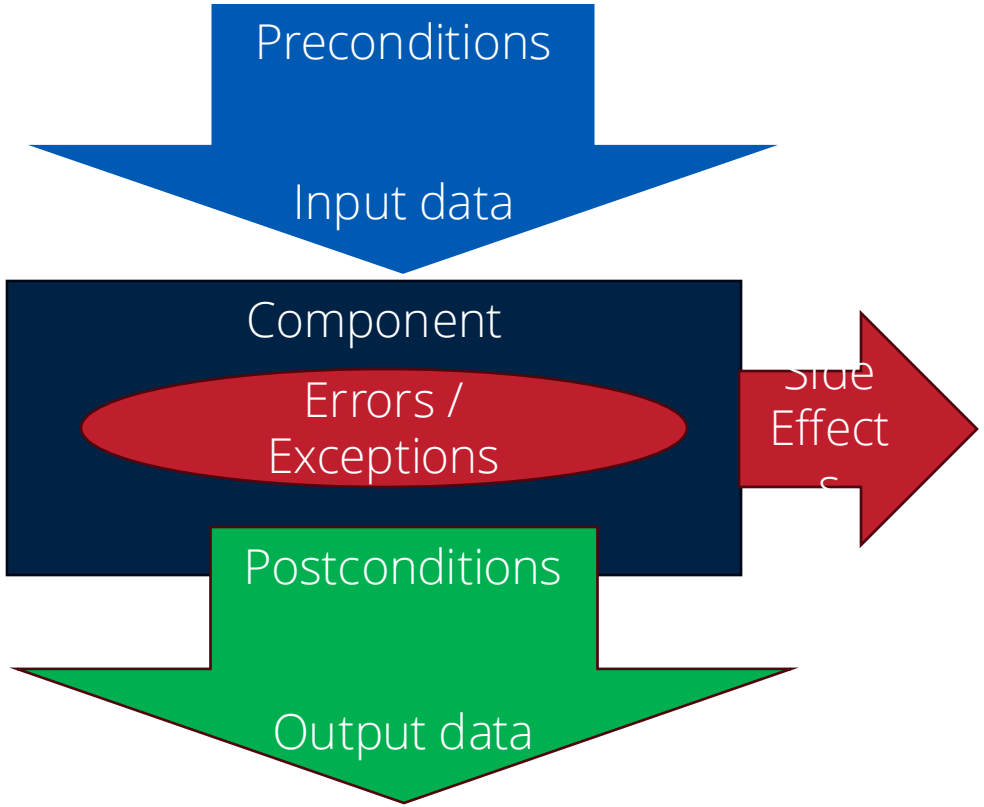
Visit **www.beningo.com** to learn more

Design-by-Contract

01

Design-by-Contract

Definition



Design-by-Contract (DbC) is a software development methodology that defines clear, formalized agreements (or "contracts") between different parts of a program to ensure correct and predictable behavior. These contracts specify the obligations, benefits, and constraints for software components, such as functions, classes, or modules, promoting robust and reliable software.

Design-by-Contract

More Definitions

Preconditions:

Conditions that must be true before a method or function is executed. These are obligations the caller must fulfill for the function to execute correctly.

- Example: A function that calculates the square root might require the precondition that the input must be non-negative.

Postconditions:

Conditions that are guaranteed to be true after a method or function executes, provided the preconditions are met. These are the guarantees the function offers to the caller.

- Example: After a withdraw(amount) method is called on a bank account object, the account balance must be reduced by amount.

Invariants:

Conditions that must always hold true during the lifetime of an object or while a program is running. These often define the consistency rules for a class or system.

- Example: In a BankAccount class, the invariant might ensure that the balance is always non-negative.

Design-by-Contract

Example

```
/* Invariant Condition: DataBuffer must remain in a consistent state and not be accessed
* concurrently without proper synchronization to prevent race conditions */
int ProcessSensorData(float * restrict DataBuffer, size_t DataLength)
{
/* PreCondition: DataBuffer must not be NULL */
/* PreCondition: DataLength must be greater than 0 and less than MAX_BUFFER_SIZE */
/* PreCondition: DataBuffer elements must contain valid floating-point values */

/* PostCondition: Return value is 0 if all data processed successfully */
/* PostCondition: Return value is -1 if invalid data is encountered */
/* PostCondition: DataBuffer contents are normalized to the range [0.0, 1.0] */
}
```

Audience POLL Question

What is an invariant?
A) A condition that must hold true before a function is executed.
B) A condition that must hold true after a function is executed.
C) A condition that must always hold true throughout the lifetime of a program or object.
D) A condition that determines whether a loop terminates correctly.

Assertions

02

Assertions

Assertion Fundamentals

Definition: An assertion is a Boolean expression at a specific point in a program that will be true unless there is a bug in the program.

Pros for assert()

- improve testing
- bugs are easier to detect
- execution stops at them
- can serve as executable comments
- improve code quality
- can be turned on and off

Cons for assert()

- slow down code execution
- commonly misunderstood
- used improperly for error handling
- use string that require RAM/ROM
- require printf and a terminal
- can be turned on and off

What is the difference between a bug and an error condition?

Assertions

Example Uses

Assertion conditions:

- If the expression is true, execution continues normally
- If the expression is false, whatever happens is “undefined”

Proper Use:

```
void CalculateDistance (uint32_t Velocity)
{
  assert (Velocity < 10);
}
```

Improper Use:

```
int result = Open("MyFile.txt", 'r');
assert (result != NULL);
```

Side Effects (Mistakes!):

```
assert (result = 14);
assert (VelocitySet(100) < VELOCITY_MAX);
```

Assertions

Implementation

Must match the function definition in assert.h!

```
288  void __aeabi_assert(const char *expr, const char *file, int line)
289  {
290    Uart_printf(UART1, "Assertion failed in %s at line %d\n", file, line);
291
292    for(;;);
293  }
```

Wait forever?

What should happen?

- Print to terminal
- Entry in log file
- etc

Audience POLL Question

What are the potential problems with assertions?

A) They immediately halt the system if an assertion is hit
B) Compiling them out can change real-time performance
C) They use CPU cycles to evaluate the expression
D) All the above
E) None of the above

Real-time Assertions

03

Real-time Assertions

Definition

Definition: A real-time assertion is a Boolean expression at a specific point in a program that will be true unless there is a bug in the program and will have minimal impact on the real-time performance if triggered.

```
void __aeabi_assert(const char * expr, const char * file, int line)
{
  // Log the assertion file and line number

  // Handle Real-time Behaviors?

  // Safe Restart Sequence?
}
```

No wait forever!

Real-time Assertions

Where to Log the Bug?

- Print to a terminal
  - IDE Terminal (ITM, etc)
  - External (UART, etc)
- Store in a RAM buffer
  - Printed at start-up or when full
- Write to non-volatile memory
  - SD card
  - Flash
  - EEPROM

Real-time Assertions

Defining Real-time Assertions

```
typedef enum
{
  ASSERT_SEVERITY_MINOR,
  ASSERT_SEVERITY_MODERATE,
  ASSERT_SEVERITY_CRITICAL,

ASSERT_SEVERITY_MAX_COUNT
}AssertSeverity_t
```

```
void assert_failed(const char expr, const char *file, int line, AssertSeverity_t Severity)
{
  #if ASSERT_UART == TRUE
    Uart_printf(UART1, "Assertion failed in %s at line %d\n", file, line);
  #elif
    Log_Append(ASSERT, Assert_String, file, line);
  #endif

  # Pull the I/O line Low to turn on the LED and signal an assertion.
  Dio_ChannelWrite(LED_Assert, LOW);

  App_Notify(Severity);
}
```

Real-time Assertions

Defining Real-time Assertions

```
void Distance_Calculate (uint32_t Velocity)
{
   assert (Velocity < 10, ASSERT_SEVERITY_MINOR);
   assert (Velocity > 300, ASSERT_SEVERITY_CRITICAL);

   // Functional code goes here ...
}
```

Audience POLL Question

What is the primary purpose of real-time assertions in embedded systems?
A) To document the purpose of a function for future developers.
B) To dynamically verify that critical conditions are met during the execution of real-time code.
C) To ensure a program compiles successfully before deployment.
D) To optimize the performance of time-sensitive algorithms.

Next Steps

04

Going Further

https://beningo.mykajabi.com/

Download Function Pointer Example Code:

- Additional Video on Baremetal Scheduling
- Scheduling Lab Instructions
- Scheduler Source Code

Additional Resources

Please consider the resources below:

- Jacob's Blogs
- Jacob's CEC courses
- Embedded Software Academy

- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

Consulting | Coaching | Training

www.beningo.com

Next Steps

- Mastering Function Pointers for Baremetal Systems
- Building Cooperative Schedulers and Command Parsers
- Baremetal Performance Analysis and Architecture Design
- Assertions and Validation Techniques for Baremetal
- Managing Interrupts in Baremetal Systems

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

DigiKey

BENINGO EMBEDDED GROUP

informa markets