Embedded Programming

Mathias Balling & Mads Thede

Last updated: June 20, 2024

Contents

1 ARM architecture

Acorn RISC Machine. Reduced Instruction Set Computer, later Advanced RISC Machine. Key feature is that ARM processors have good performance per Watt.

1.1 RISC

RISC is a small set of simple and general instructions. Advantages compared to CISC:

- Instructions take one clock cycle
- Performance is better due to simplified instruction sets
- Less chip space is used due to reduced instruction set
- Can easily be designed as compared to CISC
- Reduced per chip cost, as it uses smaller chips
- Performance of the processor will vary according to the code being executed
- RISC processors require very fast memory systems to feed various instructions

Risc is used in Qualcomm snapdragon which is in most smartphones. In addition to this apple silicon is also based on ARM.

1.2 ARM Cortex-M4 peripherals

SysTick timer is a 24-bit timer that counts down to zero. Systick is used for precise timing. Whenever the timer runs out it will trigger an interrup.

Registers: A processor register is one of a small set of data holding places that are part of the computer processor. A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual chracters). Some instructions specify registers as part of the instruction.

Data types: 32-bit words, 16-bit half words, 8-bit bytes.

Interrupt vector table: Contains the addresses of the interrupt service routines (ISR). Using the addresses you can jump to the ISR when an interrupt is triggered.

2 State machines

A state machine is a computational model that is used as an abstract machine which can be in only one state at a time. There are two representations of state machines, Mealy and Moore.

Moore machines have outputs that depend only on the current state.

Meal machines have outputs that depend on the current state and the current input.

3 Notes

Static variables: Keep their values between unrelated calls to the function.

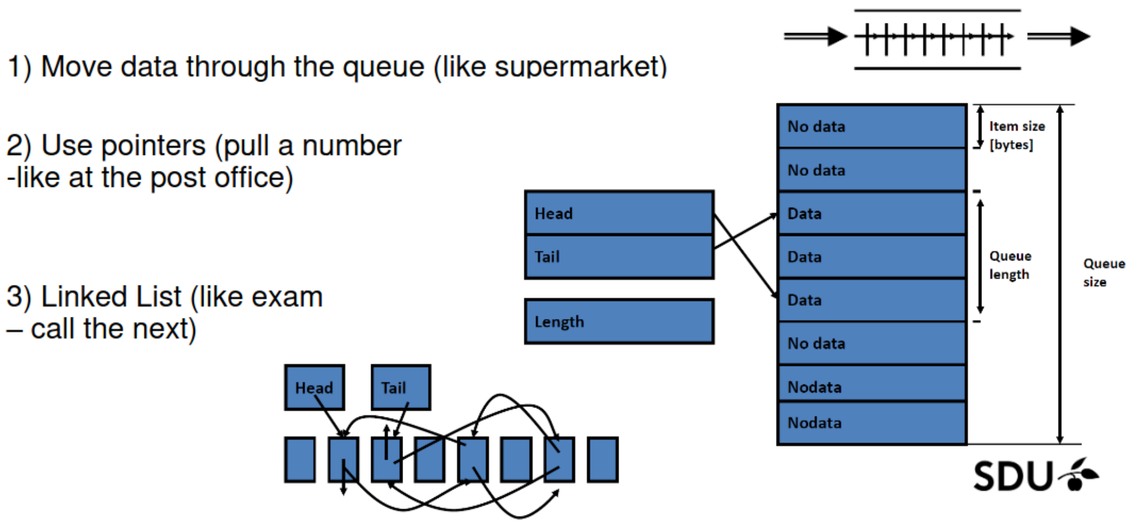
Global variables

Volatile

extern

3.1 Queues

A queue is an abstract data type that contains a collection of elements. A queue object is an individual object in the task diagram. As an object it has a name, number, id, key or handle and implement a FIFO strategy. In addition to this it offers an API for creating, inserting data, fetching data, show the state of the queue and maybe read data without removing it from the queue (peek).



3.2 Semaphores

Real life: Visual signaling with arms and flags. A different example can be parking access control.

In software, a semaphore is a data strcture for synchronization. A semaphore is an integer varaible, but the only allowed operations are increment and decrement. When a task decrements the semaphore, if the results is neagtive, the task blocks itself and cannot continue until another task increments the semaphore. When a task increments the semaphore, if there are other tasks waiting, one of the waiting tasks gets unblocked.

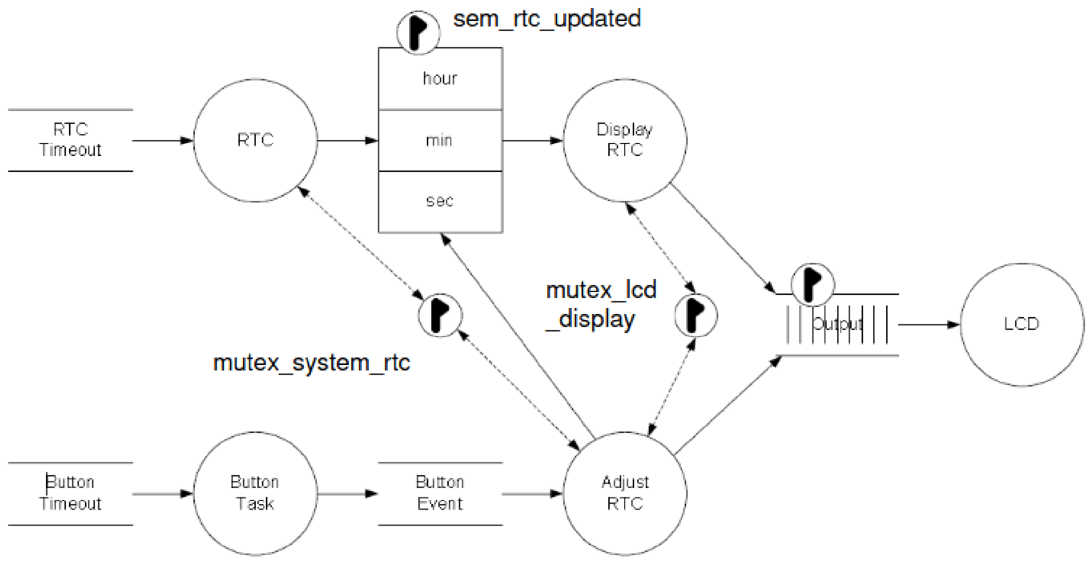
Incrementing a semaphore is done using Signal() and decrementing is called wait().

There are two types of semaphores: binary and counting.

The reason for using semaphores is that it helps prevent bugs. In addition to this the purposes of semaphores are to protect shared variables, protect shared resources and critical sections. Thus it helps with signaling (serialization) by making sure that statements in different tasks/threads execute in a specific order.

When using a semaphore for signaling you can have one task decrement the semaphore and then a different task increment the semaphore.

Semaphore mutex is a type of semaphore that secures mutually exclusive blocks to prevent concurrent access to the same resource. A mutex is a binary semaphore that creates critical sections.



3.3 Debugging

Simplest form of debugging is to toggle an LED pin to check execution.

Another form of debugging is to use an assert statement. This is a macro that prints a message and halts the program if the argument is false. The message contains line number and source file.

Otherwise you can use the serial port and write messages to a terminal. However, this uses extra RAM and program memory for the printf routine.

PC simulation can be used but not for real-time testing.

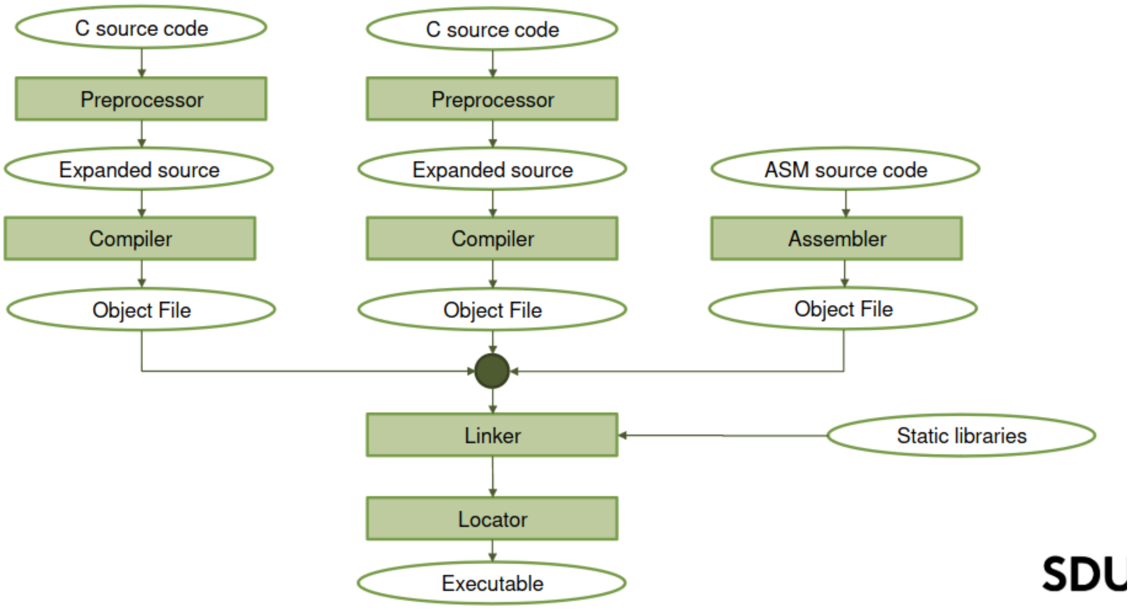
ROM monitors are a cheap solution that uses RAM, ROM and execution time. The good thing about a ROM monitor is that it is always available but it can be problematic when debugging ISRs.

In Circuit Emulators are expensive but very powerful. This replaces the microprocessor on the board. But they are problematic at high frequencies because of the long wires.

In circuit debuggers or On-Chip Debuggers are the most common debuggers. These consume no RAM or ROM and are always available. They are used for debugging the hardware kernel.

3.4 Build process

A build process is hardware speficic. When using the TIVA controller the compilation happens on the PC but for the TIVA controller, this is called cross compilation. The compilation is handled by the IDE (code composer studio)



The **preprocessor** handles preprocessor directives. This is what starts with a # in the code. This is for including libraries and defines. In addition to this it removes comments and combines split lines. The expanded code you get from copying header files into the source code is then passed on to the compiler.

The **compiler** translates human readable code into machine code. The output of a compiler is an object file, which is not executable. Instead it is a binary file that contains instructions and data.

A **linker** linkes the object files together and resolves any dependencies between them. It also links the libraries used in the code. The output of the linker is relocatable and has no memory addresses assigned to the code. A relocatable program is a program that can be loaded into memory at any address.

The **Locator** then transforms relocatable program into executable binary image. Addresses are specified for each code line.

3.5 Coding standards

- Think in abstraction levels
- limit module size
- Limit function size
- Comment your code
- Use proper indentation
- Use expressive file names
- Avoid: GOTO and CONTINUE
- Avoid code repetition

If a function contains more than 5 or 6 levels of indentation or occupy more lines than presented on the screen, it should be considered to break the function into sub functions.

RETURN should only be placed at the end of the function. This can be ensured by using state variables instead of multiple returns.

Avoid using abreveations in variable names, unless they are obvious. Use underscores to seperate words.

3.6 Logical vs bitwise

Logical operators operate on bytes and words. An expression is true if it not equal to zero. An expression is false if it is equal to 0.

Bitwise operators perform logical functions by comparing bits one by one.

3.7 Interrupts

An interrupt is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution.

The hardware event is called a trigger. When this happens, an ISR is called. This pauses the execution of the regular code, until the ISR returns or a higher priority interrupt is triggered.

4 Real Time Operating System

Responds to events within a strictly defined time. The scheduler is deterministic, meaning that the time it takes to switch between tasks is predictable. Priority based scheduling is used to determine which task to run next. Thread=task. Data is processed as it comes in, typically without buffer delays.

Application of RTOS could be airbag deployment, anti-lock braking system, etc.

FreeRTOS

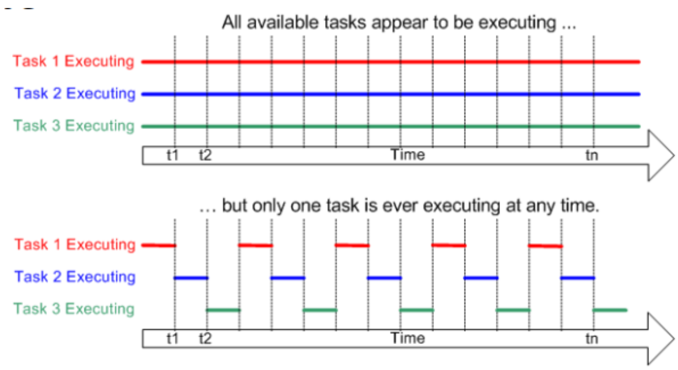
RTOS small enough to run on a microcontroller. Embedded controllers don't warrant full RTOS because of their hardware limitation. FreeROTS provides only:

- Real-time scheduling functionality
- Inter-task communication
- Timing and synchronization primitives

Multi-tasking

Kernel - core of OS.

Multi-tasking in OS is if the OS can run multiple tasks at the same time. This allows for complex applications to be subdivided into small tasks.



Scheuler

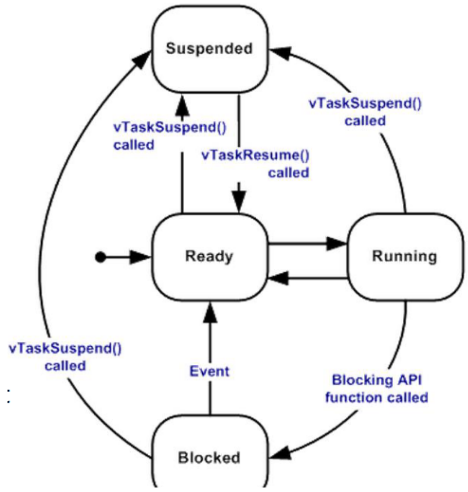
The scheduler is a part of the kernel that decides which task will be executed when. The scheduler can suspend and resume tasks, but tasks can suspend themselves too.

FreeRTOS choice of scheduling policy

Two choices: Preemptive and Cooperative. Pre-emptive always runs the highest priority task that is ready to run. Having multiple tasks with the same priority, they will run using round robin scheduling. Co-operative scheduling is where context switches only occur if a task blocks or explicitly calls task YIELD().

FreeRTOS Task States

There are four states in FreeRTOS: Running, Ready, Blocked, Suspended. Running is the task that is currently executing. Ready is where the task is able to execute but is not currently executing due to another task with equal or higher priority running. Blocked is where the task waits for a temporal or external event. It can block waiting for a queue or semaphore event. Suspended is where the task is not available for scheduling. Tasks only enter or exit suspend by vTaskSuspend() and xTaskResume().



Sempahores

Semaphores are used to synchronize tasks and protect shared resources. A semaphore can be a signaling semaphore or a mutex. Meaning that a semaphore can be used to signal between tasks or to lock a resource. Semaphores can be binary or counting.

Resource Management

There is a potential risk for a conflict in a multitasking system. If one task starts to access a resource but does not complete its access before being transitioned out of the running state. If the task left the resource in an inconsistent state, then access to the same resource by any other task or interrupt could result in data corruption.

Mutual exclusion

Mutual exclusion can be implemented in FreeRTOS by: Disabling interrupts, using a mutex or disabling the scheduler.

Mutex

A mutex is a special type of binary semaphore that is used to control access to a resource. Mutex vs. Semaphore: Mutexes can be released only by the task which took them, while binary semaphores can be released by any task. Mutexes are for protecting resources - binary semaphores are for serialization and signaling.

5 Learning objectives

- Explain the special demands that embedded systems place on software, and how to meet those demands by using the programming language: C.
- Analyze specifications of I/O-devices, and design high performance, hardware-near programs for these
- Evaluate real-time conditions in an embedded system
- Explain principles and algorithms for central parts of operating systems
- Design functions to a real-time operating system

What is embedded programming?

Programming of embedded systems. An embedded system is a computer system - a combination of a computer processor, computer memory, and input/output peripheral devices - that has a dedicated function within a larger mechanical or electrical system.

Why is embedded programming important?

It is important because it is used in many devices that we use in our everyday life. For instance in cars, airplanes, household appliances, and medical equipment.