Embedded Programming

Mathias Balling & Mads Thede

Last updated: June 23, 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.

RISC is more power efficient and used in embedded systems because of its cost efficienty and power efficiency.

1.2 ARM Cortex-M4 peripherals

The ARM Cortex-M4 is a processor core that the tiva is designed around. The ARM Coretx-M4 processor provides the core for a high-performance, low-cost platform that meets the needs of a minimal memory implementation, reduced pin count, and low power consumption. 80 MHz, 32-bit processor.

RISC architecture.

Harvard achitecture: Separate memory for data and instructions.

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.

FPU: Floating point unit. Used for floating point calculations.

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.

Mealy 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: are variables that are declared outside of any function.

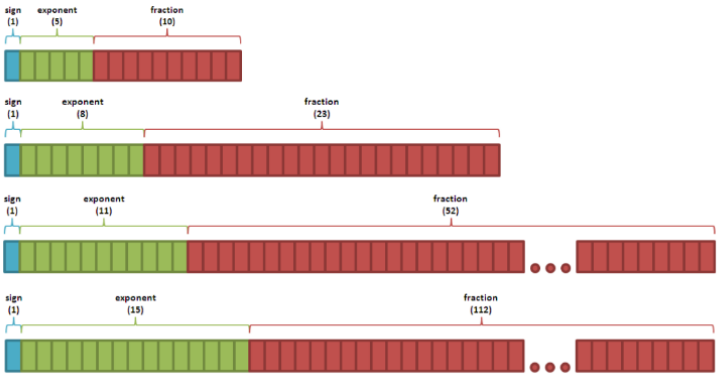
Volatile ints: are used to tell the compiler that the value of the variable is not predictable. This is used when the variable is changed by an ISR.

Extern ints: are used to tell the compiler that the variable is defined elsewhere.

3.1 Floating Point Unit

Floating point is a way to represent real numbers on computers. IEEE floating point formats:

- Half (16-bit)
- Single (32-bit)
- Double (64-bit)
- Quadruple (128-bit)



The fraction part also known as the mantissa, describes the significant digits of the number. Where the exponent tells us where the decimal point is.

The FPU provides floating point computation functionality that is compliant with the IEE 754 standard. It enables conversion between fixed-point and floating-point data formats, and floating-point constant instructions.

3.2 Pointers

Null pointer is a pointer used to initialize a pointer without a memory address. It can be used as a function argument.

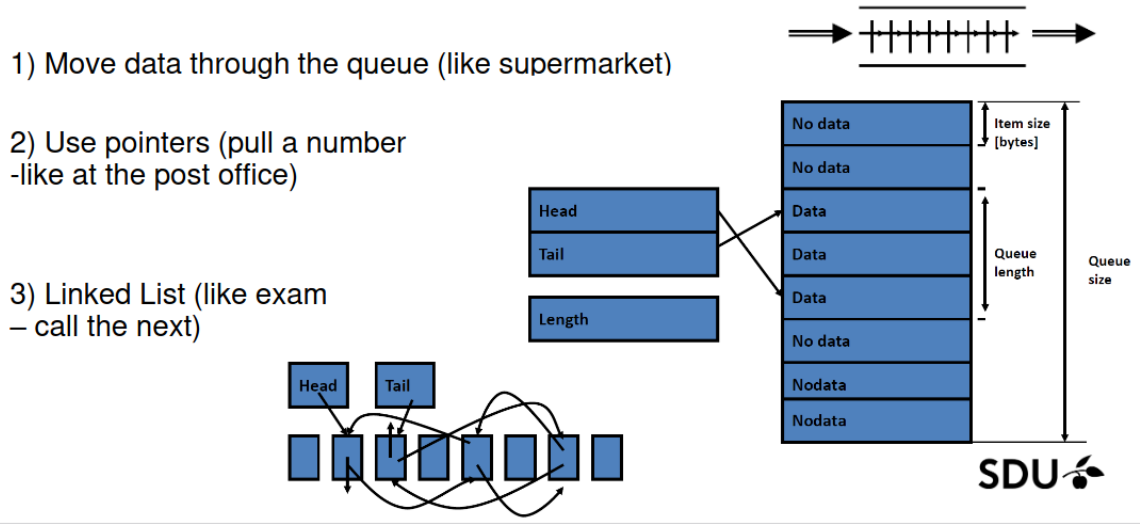
A void pointer is used to receive the address of a variable of any data type.

3.3 Strings

In C a string is a line of characters pointed to by a pointer and continuing until a $'0'$ character is encountered. The string might exist in memory of an array or as a constant.

3.4 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).



If a producer tries to write to a queue that is full, the producer can throw away the data, do something else and try later, overwrite old data or wait(block).

3.5 Semaphores

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

There are two types of semaphores: binary and counting.

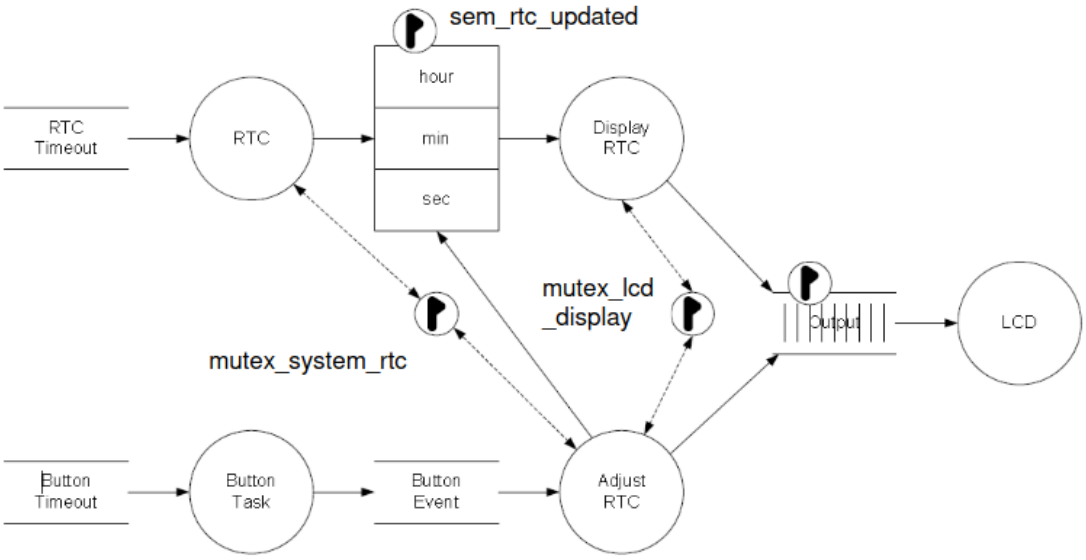
In software, a semaphore is a data structure for synchronization. A semaphore is an integer varaible, but the only allowed operations are increment and decrement. When a task decrements the semaphore, if the results is neagtive, the task blocks itself and cannot continue until another task increments 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().

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.



3.6 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.

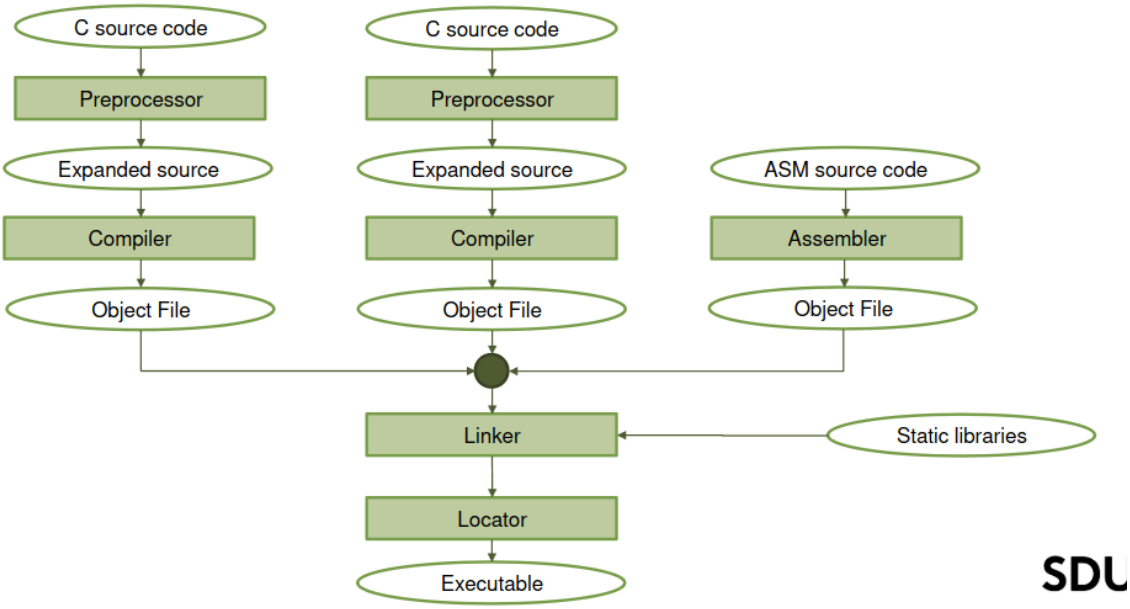
ROM monitors are a cheap solution that uses RAM, ROM and execution time. The good thing about a ROM monitor is that it is always available but it can be problematic when debugging ISRs.

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.

3.7 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.

3.8 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.

3.9 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.

3.10 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.

3.11 Printf

Printf in C is a function that prints formatted output. Using printf one can format the string to contain embedded format tags that are substituted with values.

An example of this is: $printf("The value of x is \%d", x);$

3.12 Deadlock

A deadlock is when a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs.

3.13 Assembler

Low-level programming language. Assembly code is very close to machine code instructions, because of its direct mapping from assembler instructions to procesor instructions.

Assembly language is a human readable programming language that contains mnemonics of machine code (mov,ldr, ldi) etc.

Assember is a computer program that translates assembly source code into machine language object code.

The reason as to why one would use assembly is because of its speed and size. It can be used for strict timing requirements, where the execution time is critical.

Assembly can be used inline in C code, like this:

$$asm("MOV R0, \#0x01");$$

A reccomended way to use assembly is to write the code in a seperate file and then include it in the C code.

3.14 Reentrancy

A function is reentrant or pure if, while it is being executed, can be called again by itself or by another function. i.e. more than one instance of a function can run at the same time.

A function is reentrant if all variables are used atomic or are allocated to the specific instance of the function (not static).

Atomic operations are operations that cannot be interrupted. To ensure this, don't disable interrupts but instead semaphores.

A function is **recursive** if it calls itself. Thus, a recursive funciton must be reentrant.

4 RTCS

Non-preemptive scheduling method. Each task runs until it has finished its job. Scheduling is based on the tick. The tick is the basic unit of time for task scheduling.

RTCS features tasks, task states, task events, timers, semaphores and queues.

API (Application programming interface): A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.

In RTCS this includes Queue API, Semaphore API, Timer API, Task API and Event API.

A superloop is placed within the schedule function.

A task may communicate by reading and writing files. A file can be a data set that you can read and write repeatedly. A stream of bytes generated by a program. Or a stream of bytes received from or sent to a peripheral device.

Files have been used in the lectures by setting up a pool of files consisting of files for uart, LCD and keyboard. Generally I/O interaction is done through files. This does not involve GPIO pins of the controller.

Operating System

An operating system is a computer program that supports the computer's basic functions. It provides services to other programs (applications). The OS manages computer hardware resources including the CPU, memory, storage devices, etc.

The main function of an OS is to provide an interface between the computer hardware and the user. The OS manages the hardware at the lowest level, allowing the user to interact with the computer at a higher level.

Most OS are not real-time, meaning that they cannot guarantee a response time to events. They can be preemptive or non-preemptive. Windows is an example of a non-real-time OS.

5 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:

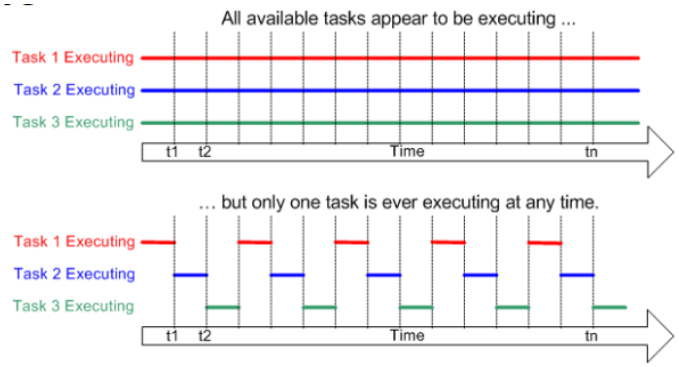
- Real-time scheduling functionality
- Inter-task communication
- Timing and synchronization primitives

FreeRTOS is more a real-time kernel than a full RTOS. It is more like a real-time kernel because it is not a full OS. This means that FreeRTOS does not provide all the services of a full OS. Where a full OS provides file systems, networking, etc. FreeRTOS provides only the basic services needed for a real-time system.

Multi-tasking

Kernel - core of OS. A kernel is responsible for managing the system resources. This includes the CPU, memory, and peripheral devices. The kernel manages processes, thus allowing for multi-tasking.

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

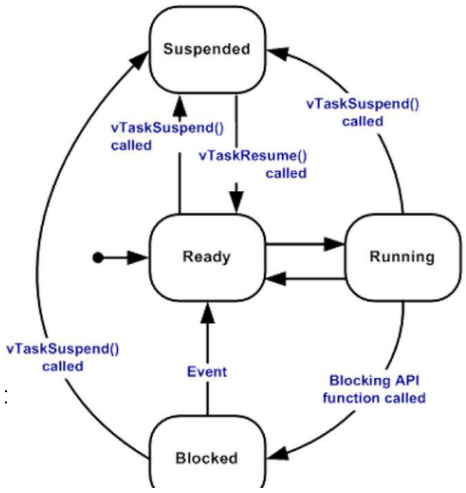
The scheduler is a part of the kernel that decides which task will be executed when. The scheduler can suspend and resume tasks, but tasks can suspend themselves too. When the scheduler switches between tasks, it is called a context switch. Tasks use different resources of the microcontroller, such as registers, stack, and program counter. When the kernel switches tasks, it saves its context, and resumes from it once a task is resumed.

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().



Additional features of FreeRTOS

- Message Queues: Used for inter-task communication. A task can send a message to a queue and another task can receive it.
- Binary and counting semaphores.
- Mutexes.

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.

Queues in FreeRTOS

A queue can hold a finite number of fixed size data items. The maximum number of items a queue can hold is called its length. Writing data to a queue causes a byte for byte copy of the data to be stored in the queue itself. Reading data from a queue causes the copy of the data to be removed from the queue.

Preemption

The general definition of preemption is a prior seizure or apporiation - taking possession before others.

A synonym for preemption is protective, defiensive, precautionary. In computing preemption is the act of temporarily interrupting a task with the intention of resuming it later.

6 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. An example of an embedded system is a pacemaker.

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.

What are the differences between a microcontroller and a microprocessor?

A microcontroller is a single chip that contains a processor (CPU), memory, and I/O peripherals. Where a microprocessor is a single chip that contains only a CPU.