Microcontroller and Microprocessor Selection Guide

Version 1.0
By Fawzi Mudawwar

روابي القابضة
Rawabi Holding

Revision History

Date	Rev	Details
October 15, 2017	1.0	Release

Table of Contents

1. Introduction

Selecting the right device on which to base a new design can be challenging. The need to make the right balance of price, performance and power consumption has many implications. The decision will be based on the immediate technology considerations based on the application. The decision will have long-lasting consequences since it will become the basis of a platform approach for a range of new products.

This document aims to guide the designer to selecting the most suitable platform for their application. It is the designer's responsibility to consider the short term and the long term goals of the platform. They must take into consideration future products that may be based on the platform.

2. Microcontroller or Microprocessor

The following are some primary differences between microcontroller and a microprocessor. Typically a microcontroller uses on-chip embedded Flash memory which is used to store and execute its program. Storing the program internally allows the microcontroller to have a very short start-up period and can execute the code very quickly. The main limitation of using embedded memory is that the total available memory space is finite. Most Flash microcontroller devices available on the market have a maximum of 2 Mbytes of program memory and, depending on the application, this may prove to be a limiting factor.

Microprocessors do not have memory constraints in the same way. They use external memory to provide program and data storage. The program is typically stored in non-volatile memory, such as NAND or serial Flash, and at start-up is loaded into an external DRAM and then commences execution. This means the microprocessor will not be up and running as quickly as a microcontroller but the amount of DRAM and non-volatile memory that can connect to the processor is in the range of hundreds of Mbytes and even Gbytes for NAND. Another difference is power. By embedding its own power supply, a microcontroller needs just one single voltage power rail. By comparison, a microprocessor typically requires several difference voltage rails for core, DDR etc. The developer needs to cater for this with additional power ICs/converters on-board.

3. Selection Criteria

3.1. Understand the application

The first step in selecting a suitable platform is understanding the application that the platform will be used in. One important specification to specify early on is if a real time processing is required. If so, then is hard, firm or soft real time processing required?

3.2. Examine the firmware/software architecture

Depending on the application, the architecture of the firmware/software will have to be based on one of the following setups:

1. Code running in an infinite loop
2. Code that runs based on interrupts
3. Code that runs on a real time operating system (RTOS)
4. Code that runs on a full operating system

3.2.1. Simple loop

The simple loop approach is useful for small applications and applications with flexible timing requirements, however it can become complex, difficult to analyze and difficult to maintain if scaled to larger systems.

Advantages	Disadvantages
Small code size.	Difficult to cater for complex timing requirements.
No reliance on third party source code.	Does not scale well without a large increase in complexity.
No RAM, ROM or processing overhead.	Timing hard to evaluate or maintain due to the interdependencies between the different functions.
	High power consumption.

Microcontrollers are the most suitable for such simple architectures.

3.2.2. Interrupt based

Code that runs based on interrupts is usually used in low power applications. This kind of setup allows hardware components to go to sleep when nothing is happening and to be activated only when an event takes place. Similar to the loop approach, this approach can become complex, difficult to analyze and difficult to maintain if scaled to larger systems.

Advantages	Disadvantages
Small code size.	Difficult to cater for complex timing requirements.
No reliance on third party source code.	Does not scale well without a large increase in complexity.
No RAM, ROM or processing overhead.	Timing hard to evaluate or maintain due to the interdependencies between the different functions.
Low power consumption.	Not suitable for applications that require continuous processing.

Microcontrollers are the most suitable for such architectures.

3.2.3. RTOS

When developing complex firmware to run on a platform, one will often have multiple tasks that need to execute. Manually scheduling these tasks and managing the limited resources amongst all of the running tasks is a nightmare.

An RTOS uses a kernel to run multiple tasks seemingly simultaneously. Within the kernel is a scheduler which is responsible for determining which task should be executed at any particular time. As shown in Figure 1, the designer is able to prioritize the tasks in order for the kernel to know which tasks take precedence over others in certain circumstances.

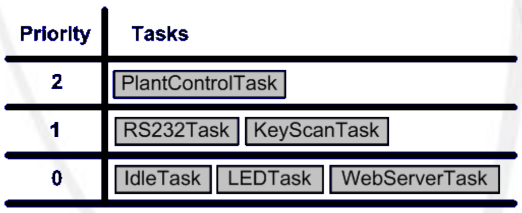

FIGURE 1

In addition to being suspended involuntarily by the kernel a task can choose to suspend itself. It will do this if it either wants to delay for a fixed period, or wait for a resource to become available or an event to occur.

RTOSs are ideal for applications that require multiple tasks running simultaneously. Examples of such tasks may include communicating with devices and modems, triggering alarms, logging data, saving data to an external storage device, Ethernet communication, etc.

Advantages	Disadvantages
Simple, segmented, flexible, maintainable design with few interdependencies.	Each task will require their own stack, and many of which require a queue on which events can be received which will in turn consume a lot of RAM.
Processor utilization is automatically switched from task to task on a most urgent need basis with no explicit action required within the application source code.	The kernel functionality will use processing resources.
The event driven structure ensures that no CPU time is wasted polling for events that have not occurred. Processing is only performed when there is work needing to be done.	Frequent context switching between tasks of the same priority will waste processor cycles.
	High power consumption.

Microprocessors are most suitable for RTOSs since they are typically resource intensive, however some RTOSs are capable of running on microcontrollers.

3.2.4. GPOS

Unlike an RTOS, a GPOS is not real-time capable. This means that it cannot provide scheduling guarantees to ensure deterministic behavior and timely response events and interrupts. A typical GPOS can achieve latencies in the order of tens of hundreds of microseconds at best, where as a typical RTOS can achieve latencies of a few microseconds.

Advantages	Disadvantages
Simple, segmented, flexible, maintainable design with few interdependencies.	Each task will require their own stack, and many of which require a queue on which events can be received which will in turn consume a lot of RAM.
Processor utilization is automatically switched from task to task.	The kernel functionality will use processing resources.
	Frequent context switching between tasks of the same priority will waste processor cycles.
	High power consumption.
	Little control over execution timing.

GPOSs can only run on microprocessors.

3.3. Determine the processing power

Determining the required processing power of a microcontroller or a microprocessor is a crucial step in selecting a suitable platform. Processing power is measured in Dhrystone MIPS (DMIPS). In order to estimate the required DMIPS, the designer needs to identify the processor intensive parts of the application. For example, running a GPOS may require between 300 to 400 DMIPS, while an RTOS may only require 50 DMIPS.

3.4. Determine the code size

It's important to have a general idea of the size of the code in order to identify how much volatile and nonvolatile memory is needed. Development of a hardware platform will go through several cycles. The first cycle, one should select a microcontroller with more RAM than needed. After verifying that the firmware runs well on the given platform, selecting a microcontroller from the same family but without too much excess resources will take place as part of cost reduction. The key to picking the right microcontroller is to use historical data. There will be a record of the other projects developed and know what memory and other resources are required to implement each feature. By knowing what the product is expected to do and having a good feature list, one can quickly and accurately calculate the resources that the microcontroller will need to provide.

3.5. Make a list of required hardware interfaces

There are two types of hardware interfaces:

a. Communication interfaces (USB, I2C, SPI, UART…)
b. I/O's (digital inputs and outputs, analog to digital inputs, PWM's…)

Make a special note if the application requires USB or some form of Ethernet. These interfaces greatly affect how much program space the microcontroller will need to support. Graphically represent the list of hardware interfaces using a block diagram such as the one shown in Figure 2.

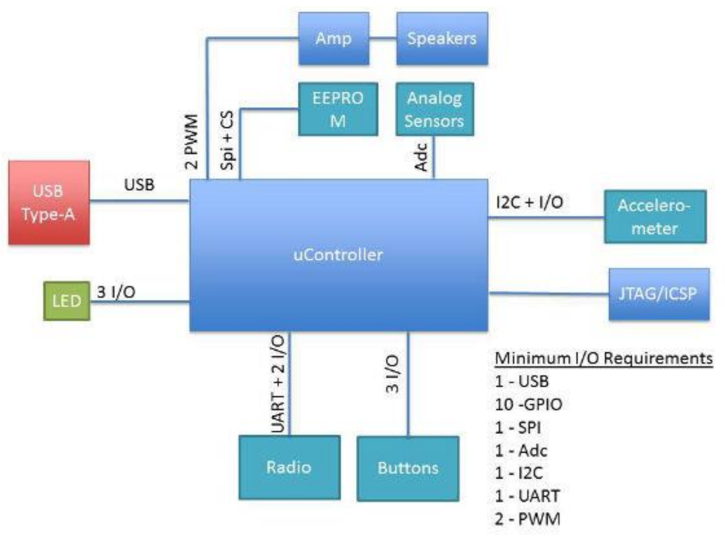

FIGURE 2

3.6. Important considerations

3.6.1. Physical Packaging

As a rule of thumb, try to always use the smallest microcontroller/microprocessor package size as possible. The time when this should be reconsidered is when the type of package will affect the overall cost (possibly by making the PCB unnecessarily complex).

3.6.2. Hardware Tools

Hardware tools are the most important considerations. Hardware is much more difficult to debug than software. A system should never be developed without having JTAG. JTAG is designed to assist with device, board, and system testing, diagnosis, and fault isolation.

A development kit provided by the manufacturer of the microcontroller/microprocessor is crucial for development. Additionally, many manufacturers provide schematic files for the development kit which are useful in accelerating the platform development.

3.6.3. Software Tools

The most important software tool that should be provided by a manufacturer is an IDE with a well-developed GNU toolchain that includes the following:

1. Compiler
2. Assembler
3. Linker
4. Archiver
5. File converter
6. Other file utilities
7. C Library
8. Debugger

The GNU toolchain will simplify the process of programming the platform. The IDE should include a tool that allows you to configure the microcontroller/microprocessor's multiplexed pins and other similar features and a debugging tool.

3.6.4. Community and Resources

Selecting a microcontroller/microprocessor that is popular is useful because it will have an online community. When an online community exists, it's easy to find solutions to issues that one may be facing since others may have suffered from the same issue.

It's important to use a microcontroller/microprocessor from a reputable manufacture with a good support track record. Support directly from the manufacturer may be needed at times.

4. Glossary

Context Switch	A context switch is the process of storing and restoring the state (more specifically, the execution context) of a process or thread so that execution can be resumed from the same point at a later time.
DMIPS	Dhrystone MIPS (Million Instructions per Second), is a measure of computer performance relative to the performance of the DEC VAX 11/780 minicomputer of the 1970s.
Firm real time	Infrequent deadline misses are tolerable, but may degrade the system's quality of service. The usefulness of a result is zero after its deadline.
GPOS	A general purpose operating system is system software that manages computer hardware and software resources and provides common services for computer programs.
Hard real time	Missing a deadline is a total system failure.
Interrupts	An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
Interrupt masking	Processors typically have an internal interrupt mask which allows software to ignore all external hardware interrupts while it is set.
Kernel	The kernel is a computer program that constitutes the central core of a computer's operating system. It has complete control over everything that occurs in the system. As such, it is the first program loaded on startup, and then manages the remainder of the startup, as well as input/output requests from software, translating them into data processing instructions for the central processing unit. It is also responsible for managing memory, and for managing and communicating with computing peripherals, like printers, speakers, etc.
NMI	A non-maskable interrupt is a hardware interrupt that standard interrupt-masking techniques in the system cannot ignore. It typically occurs to signal attention for non-recoverable hardware errors.
Non-volatile memory	Is memory that can get back stored information even when not powered.
RTOS	A real-time operating system is an operating system intended to serve real-time application process data as it comes in, typically without buffering delays.
Soft real time	The usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.

5. References

1. http://www.atmel.com/images/mcu_vs_mpu_article.pdf
2. http://www.freertos.org/