3 Embedded System

3.1 Features of target system

--Embedded system, which is opposite to host system, will be ultimate destination of our work.

In our point of view, we expect to build a little and versatile platform, even though the target products are often single purpose. This kind of system has extreme flexibility so that we can change its usage by modifying any application program and peripheral devices. According to this feature, our target system likes mankind that can have multi-usage with one body.

Like human with 5 circulatory apparatus, which are blood circulation, digestive circulation, immune system, central nervous system and autonomic nervous system, the target system also has to own similar subsystem.

Vitality	Blood circulation	Power & Clock
Convert	Digestive circulation	Peripheral Devices and Ports
Diversity	Immune system	Authentication & Opposite
Spirit	Central nervous system	Instruction Set & Bus Matrix
Coordination	Autonomic nervous system	SNS & PNS

The system will be fit to be a platform for all of our expectations because these five features are complete.

REMARK:

Unlike the Linux, we do not intend to support many kinds of CPU. We only concern the compacted MCUs, what are we want.

In this chapter, we will describe the most commonalities of compacted MCU, such as in STM32 series and TI's series chip. These kinds of MCU (with lower power and smaller size) will be our target hardware platform in the future.

https://github.com/fluctuations00/Program-Linux-Code-in-Embedded-Chips

3.2 CPU and MCU

We should know which we are working for. Before this, it is better to make a distinction between MCU and CPU.

CPU is the abbreviation of **C**entral **P**rocessing **U**nit, and MCU is the abbreviation of **M**icro **C**ontroller **U**nit, sometimes is called as single chip microcomputer.

Generally, a computer comprises of a CPU, considerable RAM, some buses, ports and customized peripheral devices. It used to do that about data analysis and decision, teaching and entertainment as a host system. But nowadays, more and more application field need to introduce automation and programmable control. In many cases, the computer can act as host system no longer. It should be small size enough that can be embedded in another system. So we must compact the system continually until we are able to encapsulate all of these units into one chip.

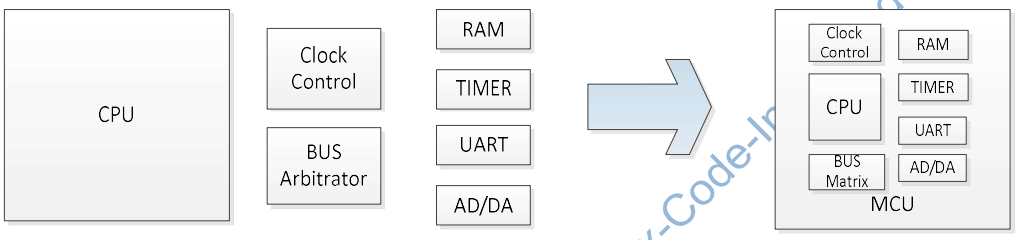

Figure 3-1: Skeleton view of the kernel

https://github.com/fluctuations00/Program-Linux-Code-In...dded-Chips

3.3 Embedded System

From mentioned before, we got two features of embedded system.

1. It can be embedded into a larger system, such as host computer or Robot.
2. It can encapsulate some general purpose ports and peripheral devices.

These two features are come from hardware and physics architecture. Now, we will expand them logically and build an important conception of embedded system.

3. It is subsystem embedded in host system.
4. It is system compacted with some smaller embedded components.

In short, it is better either the embedded system can be embedded into other system or is embedded in by other systems.

In this sense, it is our ultimate destination to design a compact and ultralow power consumption system. In this idea, we rearrange the embedded system as the following,

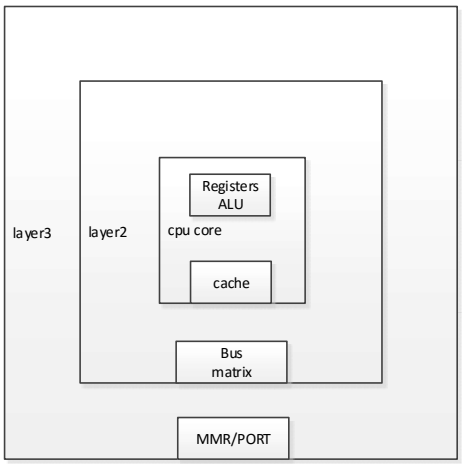

Figure 3-2: Skeleton view of the kernel

From the figure, modern embedded system or MCU indeed, has 3 layer generally. The innermost layer is processor core that consists of registers and logical unit which are related to *instruction set* and cache memory. The middle layer is bus arbitrator and bus matrix. The outer layer consists of bus interface with outside, dynamic memory, IO port with MMRs, clock generator and power supplier.

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

3.4 Processor Core

-- Matrix: instruction matrix, bus matrix and signal matrix

The processor core can be CPU in common sense (in host computer system). Actually there is a few difference between them. There are some CPUs perhaps without definition of bus and clock proxy.

Now, in order to create a clear outline of conception, we give an anthropomorphic redefinition for processor core through comparing with central nervous system.

A processor core is composed of some general registers, cache memory, inner bus and three operator matrices which are instruction decoder matrix, bus matrix and signal matrix. From point of view of software, the register and cache belong to data structure, and the three matrices can be considered as algorithm.

Generally we can consider instruction decoder matrix as instruction set. If the CPU is the core of MCU, then the instruction set exactly be the core of CPU as well.

3.4.1 Instruction Set – Instruction decoder matrix

-- *The instruction set is about algorithm of instruction decoding.*

All of CPU is indeed a circuit machine, so that definitely only machine code can be recognized by those circuits. As a central nervous subsystem can do, the process core, or called as CPU, does convert program to corresponding switch signal to guide data exchanging or calculation.

Sequentially, any program written by high level language will be compiled into assembly language sentence. Then these assembly code will be convert into machine code, called as instruction code, which will be decoded through instruction decoder and transmit control signal of gate circuit to everywhere. The signal will control the circuit switch, such as tri-state gate, to complete data transfer between register and logic operation in ALU.

https://github.com/fluctuations00/Prog...nux-Code-In-Embedded-chips

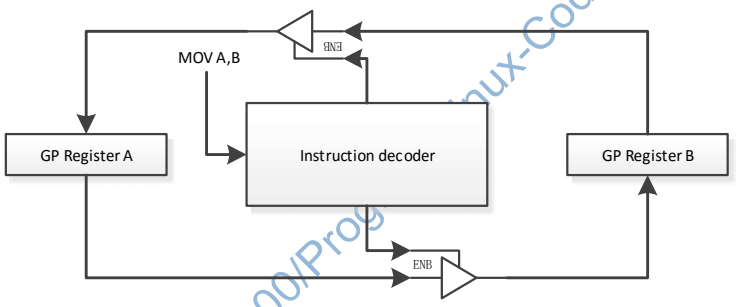

Figure 3-3 (a): data transfer between 2 registers

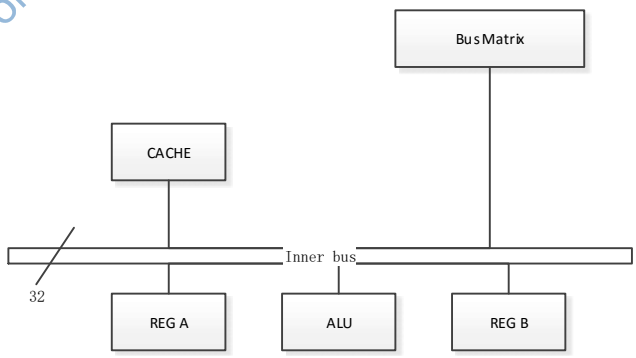

Figure 3-3 (b): Inner bus to interconnect every components of core

In figure 3-3 (b), we can see that, inside one core, all of registers and cache are interconnected by inner bus. All of accesses and operations between registers can be considered as circuit gate switching on a bus. In one clock cycle, only one data can pass

through the bus. The execution of each instruction is just the action of switch on or off of the gate.

Instruction decoder matrix – core operator

An instruction decoder matrix convert instruction code, usually 16/32/64 bits width, to switch signal of circuit gate and trigger.

This matrix is the core operator, and a serial of cascade similar operators compose an entire operating system.

Complete instruction set

From system analysis in chapter 1, each instruction decoder matrix has a set of eigenvector, by which we can span a complete instruction space. So a complete instruction set denotes instruction decoder matrix indeed.

Generally, a matrix of N order has N eigenvectors and assemble 2^N instructions. Thus a 16 bits matrix can provide 65536 instructions totally. Certainly, not all instructions are useful.

Essential instruction set

By comparing with ARM-cortex 4, which only provides dozens of instructions, it is obvious that those instructions provided by CPU designer only be a subset of the complete instruction set.

But this subset includes frequently used instructions, or in other point of view, they are necessary.

Illegal instruction

Every programmer can achieve all of their purpose through only the essential instructions. Moreover the rest instructions of complete instruction set will cause some irregular problems probably.

Perhaps someone will find some high efficient or special instruction. As well somebody want to achieve some ulterior purpose with them.

Almost all of the extra instructions are irregular and illegal except for a few.

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

UnDocument instructions

Sometime the CPU designer will reserve some instructions for debug or other else. They don't want to publish these instructions for a variety reasons.

Instruction checking

Some programmer with high-level language perhaps think that they has no chance to use or generate any illegal instructions because compiler does not support them.

But programmed codes are stored in memory zone, then this zone can be modified intentionally or unintentionally. Sometimes there exists error within program transfer or zip/unzip procedure.

In this case, CPU should arrange a checking procedure when instruction prefetching. Any error or mismatch will result in a CPU exception event.

https://github.com/fluctuations90/Program-Linux-Code-In-Embedded-Chips

3.4.2 Bus matrix

-- high performance v.s. high flexibility

The algorithm of bus matrix design is important and is second only to instruction set. Now, the CPU IP supplier such as ARM, only think about an opened architecture in order to build an industrial ecosphere. Then their IP will be accepted wildly.

The bus matrix is use to build a bridge joint between inner bus and outer bus. It can be considered as extension of instruction decoder matrix, because it isn't necessary to take care bus switch when an instruction is controlling memory mapped registers.

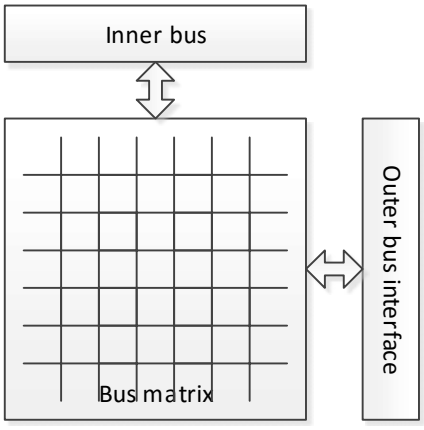

Figure 3-4: Bus matrix between inner bus and outer bus

For example, there is a *bus matrix,* named AHB-Lite in STM32F40X, which acts as bus interface as well.

Inner bus

Referencing figure 3-3(b) in the early, the inner bus links general registers, ACC/ALU and Multiplier together. But it can't link DRAM and memory mapped registers directly. It should be done by transfer of bus matrix.

The inner bus is derived from instruction set absolutely.

Outer bus

The outer bus is used to connects CACHE/SRAM, MMR and extend bus controller with CPU core. The outer bus decide what functionality we can do except pure numerical calculating and logical analysis, such as temperature and pressure detecting or grasping and throwing an object.

From this point of view, only the outer bus is closely related to us, the system developers (not CPU designer). The help directly from instruction set and its derived inner bus is far less than outer bus.

Outer bus interface

Besides a few of CPU used for specific application, the most of CPU designer who can't know the purpose in the future. So who should elaborately design an outer bus interface to meet the demand as many as possible. Therefore the key of outer bus is bus interface which provide the convenience to make people learning and studying easily. At the same time, a perfect bus interface definition can bring us the compatible for the future.

The evaluating standard of instruction set design is whether it can take with high performance to the CPU. On the contrary, for CPU design, the standard is the popularity degree.

Exactly, neither performance nor advance, but convenience and customizability decide whether the CPU will success or failure. Thus system developers are more interested in outer bus than inner bus or instruction set.

Not only CPU/MCU designer but also device and APP developer will focus on the bus interface. In fact, they can only see the bus interface. Thus it can be seen, the bus is the most important in the field of MCU application and development. More specifically, we need an opened and hierarchy bus interface.

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

~~Furthermore, the more clear hierarchy, the more reliable will be taken with the bus. Because a clear and hierarchy statement is more convincing than a mysterious confidence. It is the reason why so many programmer like Linux more than other.~~
~~条理清晰的阐述比神秘的自信更让人信服~~

A successful example of open-definition bus is ARM, whose AHB-Lite bus-matrix rather than anything else is the key to influence people's preference.

3.4.3 Signal Dispatch Matrix

The third matrix of core is the signal dispatch matrix. For central nervous subsystem, the signal dispatch matrix be a very important component absolutely. It will help system determine the event reason and source.

As a system that wants to provide service for all of applications, it must be able to answer as many outer event as possible. But there is limit in kernel's resource so as to we should employ the limitary resource to serve for the unlimited purpose. The signal dispatch matrix is created for this requirement. The MCU and platform developers will delimit a service area and provide a route for all outer event in this area.

Because of limit of resource, always there are some situations which can't be cared for, so that almost every operating system will rearrange event dispatch by themselves. In this case, we will discuss signal dispatch detail later.

3.5 Feed, Digest and Convert Subsystem

-- Bus and devices

Ignoring the differences in form of all kinds of computers or auto-controllers, all of them can be considered as an operator $\hat{0}$ acting on an input information and producing an output signal. Thus the input and output components be very important certainly for any system.

In this document, the IO components that we use usually involve SPI, UART, ADC and GPIO and so on. These ports seize the data, and store them in memory. At last the subsystem transfer and reorganize the data to formatted structure that can be recognized by processor core.

https://github.com/fluctuations00/Program-Linux-Code-in-Embedded-Chips

3.5.1 Bus interface – DMA bus and Non-DMA bus

All of these ports, memory and processor core can be considered as devices which can be connected by bus (outer bus in MCU).

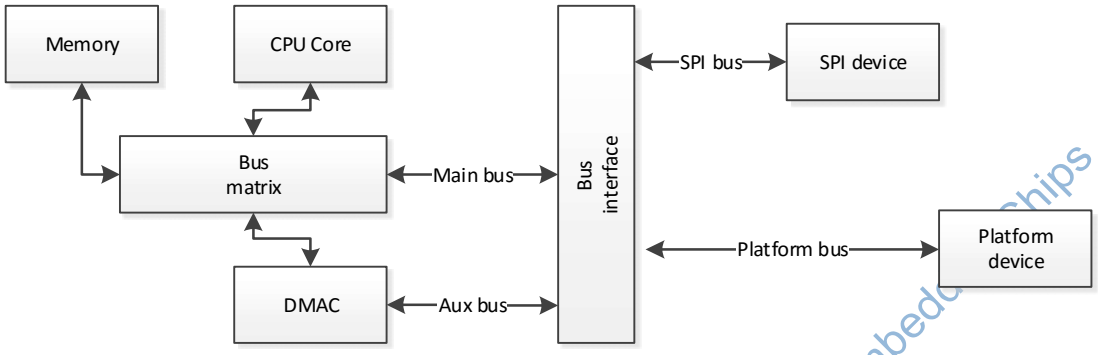

Figure 3-5: Every devices interconnected with bus

The memory like a stomach where the data must be stored firstly. The buses used to carry and transfer data in or from memory can be divided into two types, DMA and Non-DMA. In Non-DMA type, the data will be exchanged by processor core (registers in it), and in DMA type, the data will carried on DMA controller (DMAC, like a reduced CPU core, only a switchers bridge).

In some CPU, there are two bus interfaces so that DMA and Non-DMA channels can work independently. It bring us the benefit that we can separate autonomic subsystem from the central subsystem like a human. But some chip producer can't split bus interface totally to support two independent pathways, such as STM MCU. They pull out two pathways from bus interface, main bus and auxiliary bus, to complete Non-DMA and DMA exchanging. Thus the CPU core can't enter into deep sleep even when DMA transmission.

3.5.2 Relationship of Devices

It must be the most important problem to understand how to organize the devices in system. How can we share a device between multitasks and avoid confliction between each other.

In MCU, any physical devices whatever CPU core or port, can be split up into two conceptual parts logically, bus and device. The bus part is assigned the property about interconnection aspect, and device is corresponding to peripheral platform aspect.

As showed in figure 3-6(a), we draw the two conceptions with vertical and horizontal, different two kinds of description. The vertical line link a series of devices of each port form bottom to top. The horizontal lines connect devices of each port with themselves buses, at last to core side through bus interface. The line of deep color represents the

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

transfer pathway of user data stream.

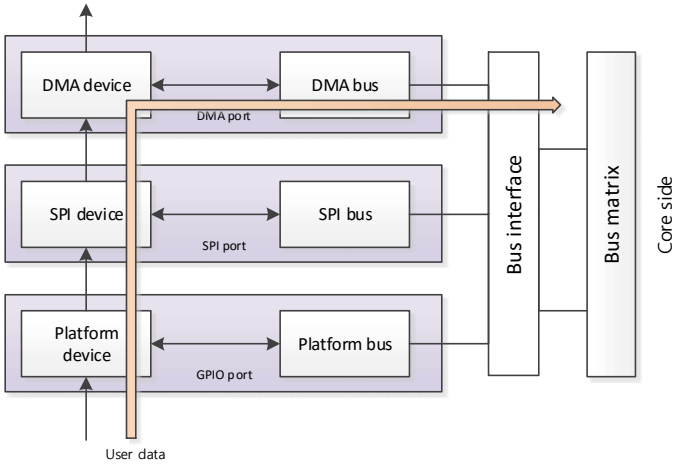

Figure 3-6(a): 2 Dimension Relationship of Devices

Through the net of latitude and longitude, we could connect all the devices and ports which will play a role in communicating with a peripheral device. Any changes occupied in a device will effect the associated bus.

There is a more important reason to split the port abstractly is that one port most probably be shared by more than one task. For each port, we should prepare an indivisible physical entity which is bus, and several logical instances which are devices. Each of the devices may be considered as an agent of a task which will access peripheral device through the port. All tasks can share the port in this kind of method, we redraw the enhanced structure as the following picture.

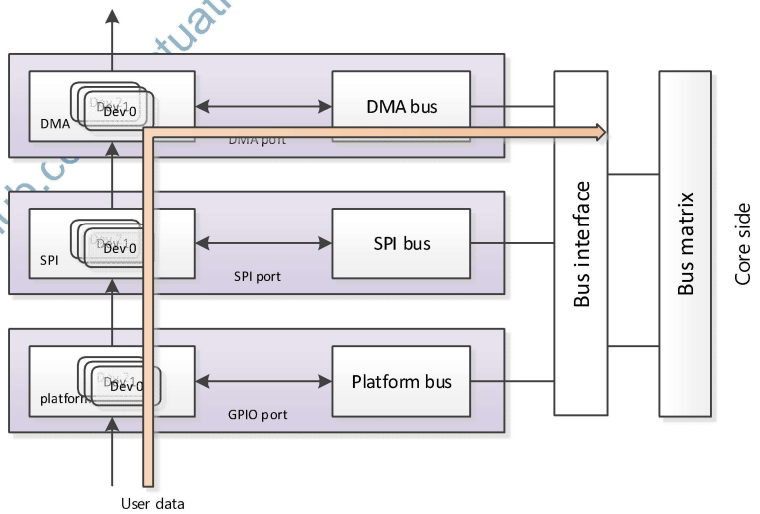

Figure 3-6(b): Shared port of multiple devices

The multi instances of device as agent help us to solve the sharing problem. Meanwhile, the uniqueness of bus may help us to avoid conflict between multi tasks.

https://github.com/tuations00/Program-Linux-Code-In-Embedded-Chips

For example, when task A is reading data from a peripheral devices through SPI and DMA port, at the same time task B need access another peripheral device through the same SPI and DMA port. Task B will ask the read/write authority of SPI and DMA port from the system, and system will deny the request of task B by the usage information of these ports. The usage information can be obtained from by the net of latitude and longitude.

REMARK:

In Linux system, everything is file, so that all of the devices can be managed by file system. Linux file system for devices provides 4 pathways to manage a device (create, look up and delete etc.). Yet we only focus on the bus pathways of (bus_kset) and devices (devices_kset). The two pathways reveal the relationship of bus to devices and relationship between devices and devices else. The bus_kset is the representation corresponding to longitude, and the devices_ket is corresponding to latitude.

3.6 Life Support Subsystem

-- Time is the source of diversity of everything, and cycle is the nature of time. Time's essence and its relationship with diversity of everything have involved the grand world view of the philosophy field and already over that our concerned.

Our target system is an electronic organism with a life period. The life support subsystem must keep on working for its whole life period. In the case of ignoring special applications, the system only has two kinds of life support components, which are power line for energy and clock line for heart rhythm.

We will study clock pipeline structure early because the power line usually works according to clock.

3.6.1 Clock Tree

In our system or work platform, all of components can't depart from power and clock support for a moment. The conveying pipeline of power and clock densely cover the chip everywhere like a tree structure. Based on this situation, MCU designers build a clock interface isolate inside clock from variety outside clock source.

The clock system has only one root to guarantee clock synchronization absolutely, meanwhile the root can select different clock source.

https://github.com/fluctuations06/Program-Linux-Code-In-Embedded-Chips

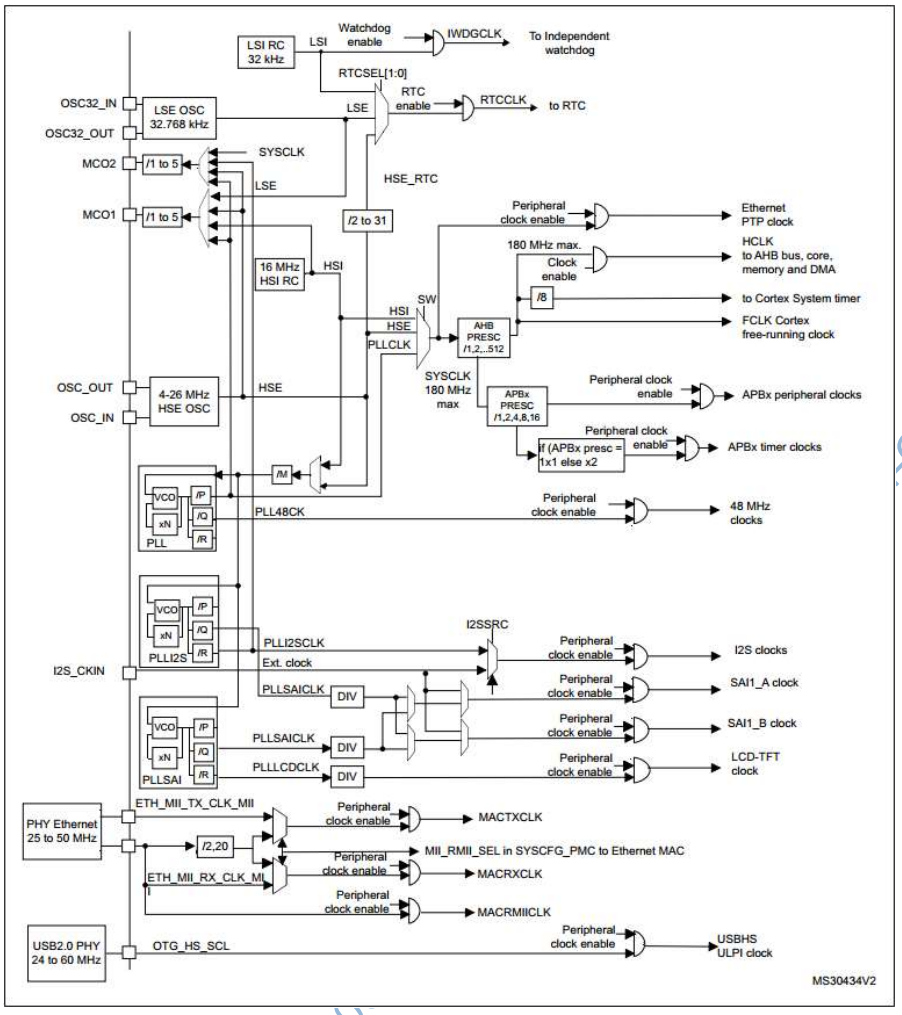

Figure 3-7: Power and Clock Pipeline with Tree Structure

Figure 3-7 presents the clock tree in STM32F4xx serial MCU. The clock diagram can be divided into 3 parts, clock source, frequency's synth and clock dispatcher.

REMARK:

In the beginning, the chip core (IP) designers doesn't provide a rigorous recommendation for clock and power pipeline like data bus interface. But almost every MCU designer employ the similar design scheme spontaneously. It lets us to apply a unified mechanism to manage the clock tree.

3.6.2 Clock Adjust

Because our system is a compacted and low power consumption humanlike system, each component of system can alternate between working with high speed or low speed. Meanwhile, every components' clock and power should to be able to be controlled independently. MCU designers build a dam independently for every component to adjust flow rate, and who can do the following works,

1. May select different clock source with or without PLL for each PORT.
2. Can change power consume of CORE and PORTs by switching frequency rate.
3. Dynamically modify clock frequency of a PORT which multiplex with different devices for different tasks.

Generally, the power supply voltage will modified automatically with working clock adjusting. Thus almost every components only provide clock controller while voltage varying passively.

3.6.3 Clock Controlling Policy

We will adjust flow rate of each component for the following reasons,
Request of peripheral device interconnecting with this port.
Request of communication protocol.
Request of low power consumption.
Request of hibernation or deep sleep.

All of this will encounter two problems that are rationality and correlativity. Rationality will make work complex, and yet correlativity will double the complexity.

Rationality involves rate matching in interconnection and power consumption to clock rate ratio. And correlativity focuses on what else components will be effected when a port switches off.

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

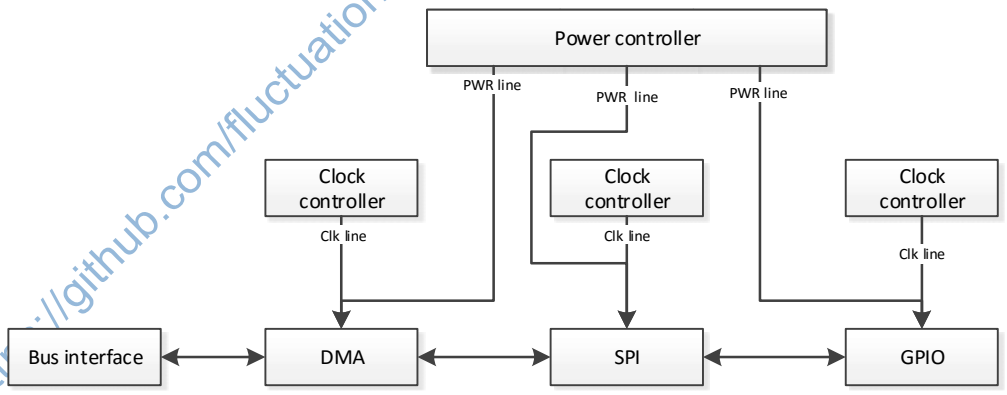

Figure 3-8 power and clock pipeline relationship between DMA, SPI and GPIO

As showed in figure 3-8, a SPI usually works with DMA and GPIO, thus when SPI is in working state, the power and clock of DMA and GPIO must be held on. Additional, if the SPI work finish and be switch off, the DMA and GPIO should keep on until there is no other ports need them.

File system structure

Early the engineers should remember the switch status by themself, after that Linux provides a *pm* structure, by which we can reorganize the power and clock pipeline.

Because in Linux system, everything is file, every problems can be treated with file system. Thus we suggest putting the clock and power controlling into file operation. We can find every associated PORTs along branches of file system, then do any necessary dependent operations for them.

3.7 Autonomic Subsystem

> *-- As we mentioned before, it is MCU not CPU that we focus on. It is more like an artificial life.*

The system looks like a human being more and more if we introduce the autonomic subsystem and immune subsystem. If some people don't understand what the autonomic subsystem is and what it can do, they can read the associate knowledge, autonomic nerve system of medicine.

Not only human but also computer system has to have a rest or deep sleep while there are some components that must keep working in the background. It looks like our hearts and lungs maintain working when sleeping.

In MCU, all of components besides CPU core can be considered as autonomous equipment generally. Besides CPU core, the MCU consists of CPLDs which start working as soon as configured, and keep working until condition becomes invalid no matter whether the CPU core work or sleep.

Here we only care about the mechanism to sustain running when the CPU core sleeping or hibernating.

https://github.com/fluctuations-IOProgram-Linux-Code-In-Embedded-Chips

3.7.1 Sleep Modes

Except shutting down, MCU life state typically can be classified as following,

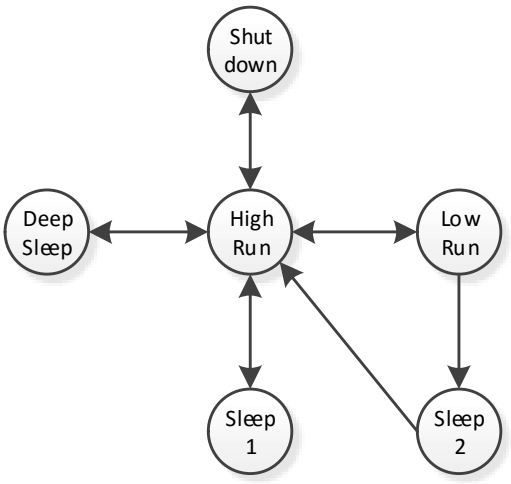

Figure 3-9: Life state transition

		core	Out bus		
			High speed port	Low speed sense port	
High Run	high performance with high consumption	4-32 (MHz)	ON	ON	CPU core run with full speed, we can save power by switching off some useless buses. Must go through this state when entering to or exiting from deep sleep and shut down state. Using internal high speed RC clock or external XTAL clock.
Low Run	Low performance with ultra-low consumption	32-100 (KHz)	ON	ON	CPU core and high speed ports run with low speed Using internal or external RTC
Sleep 1	OFF	OFF	ON	ON	CPU core enter sleep but high speed system clock and high speed ports still be running Using internal high speed RC clock or external XTAL clock.
Sleep 2	OFF	OFF	OFF	ON	Only low speed sense port be running. Using internal or external RTC Need specific (low refresh rate DRAM)
Deep Sleep	OFF	OFF	OFF	OFF	All of components are stop except RTC System can response external events System will return from sleeping entry point

This table only presents one kind of description of system state. Furthermore there are any other types of state definition absolutely.

It is obviously that we can't define the sleeping and autonomic mode clearly. In this case, it is recommended that we can design and develop the sleeping mechanism by referencing

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

autonomic nervous system of a human being.

3.7.2 Autonomic nervous system

The autonomic nervous system (ANS), formerly the vegetative nervous system, is a division of the peripheral nervous system that supplies smooth muscle and glands, and thus influences the function of internal organs.[1] The autonomic nervous system is a control system that acts largely unconsciously and regulates bodily functions, such as the heart rate, digestion, respiratory rate, pupillary response and urination.[2]

The autonomic nervous system is regulated by integrated reflexes through the brainstem to the spinal cord and organs. Autonomic functions include control of respiration, cardiac regulation (the cardiac control center), vasomotor activity (the vasomotor center), and certain reflex actions such as coughing, sneezing, swallowing and vomiting. Those are then subdivided into other areas and are also linked to autonomic subsystems and the peripheral nervous system. The hypothalamus, just above the brain stem, acts as an integrator for autonomic functions, receiving autonomic regulatory input from the limbic system.[3]

https://github ux-Code-In-Embedded-Chips

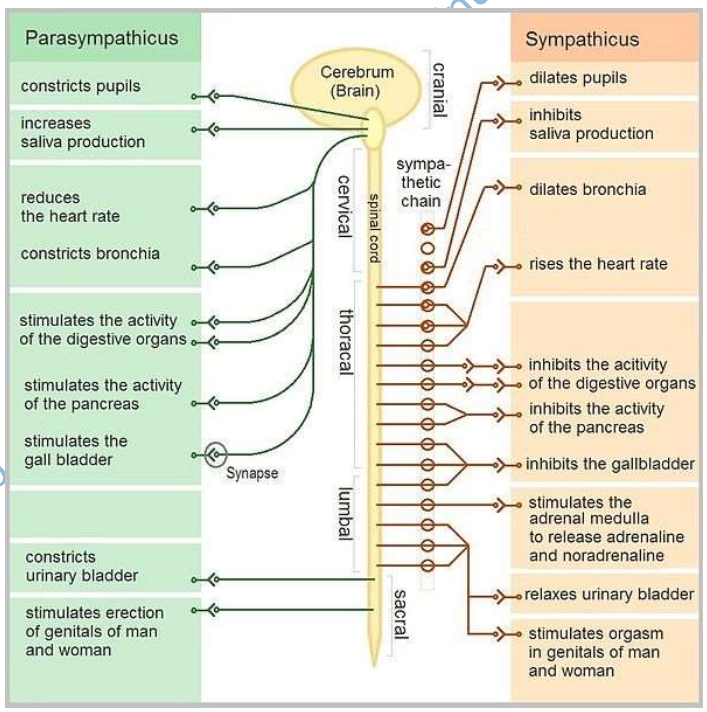

Figure 3-10: Function of the autonomic nervous system

The autonomic nervous system has three branches: the sympathetic nervous system (ANS), the parasympathetic nervous system (PNS) and the enteric nervous system.[4][5][6][7] Some textbooks do not include the enteric nervous system as part of this system.[8] The sympathetic nervous system is often considered the "fight or flight" system, while the parasympathetic nervous system is often considered the "rest and digest" or "feed and

breed" system. In many cases, both of these systems have "opposite" actions where one system activates a physiological response and the other inhibits it. An older simplification of the sympathetic and parasympathetic nervous systems as "excitatory" and "inhibitory" was overturned due to the many exceptions found. A more modern characterization is that the sympathetic nervous system is a "quick response mobilizing system" and the parasympathetic is a "more slowly activated dampening system", but even this has exceptions, such as in sexual arousal and orgasm, wherein both play a role. [3]

Reference: https://en.wikipedia.org/wiki/Autonomic_nervous_system#cite_note-13

Put aside the complex description of autonomic nervous system, we simplify the structure demonstration with heart and stomach. We select heart as representative of life-support circulation and stomach as representative of feed-digest circulation.

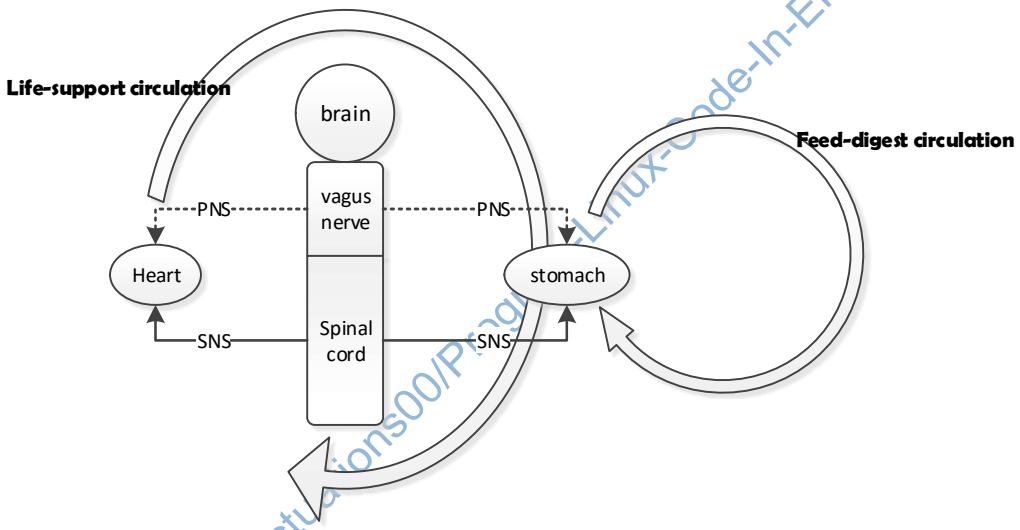

Figure 3-11: Simplified description of ANS with heart and stomach. SNS is an acronym for sympathetic nervous system. PNS is an acronym for parasympathetic nervous system.

https://github.com/fluctuations00/Practical-Linux-Code-In-Embedded-Chips

The sympathetic nervous system's primary responsibility is to constantly keep active above a basic level. The parasympathetic nervous system (PNS) is responsible for activities of peripheral body organs when the body is at rest. The parasympathetic nervous system action is described as being complementary to that of the sympathetic nervous system. From this we consider SNS working in running and PNS in sleeping.

By the way, we discuss the difference of control pathway between SNS and PNS. Because the SNS work on running state in high speed, it will use spinal cord (central nervous) to provide a strict and high speed signal interface. Oppositely, PNS only need low speed nerve interaction. Our body directly connect PNS with vagus nerve which is a part of brain and has not enough to evolve into spinal cord. The vagus nerves which connect each organs, still is a slender shape and carries slow nerve signal.

3.7.3 Mechanism of MCU Autonomic System

We may continue to use the terms of SNS and PNS in MCU autonomic system. Now, SNS represents the part of work in waking status, meanwhile PNS represents the part of work in sleeping status.

This mechanism, which could be called as an algorithm to construct the control and data bus that will work separately (not independently) when awake or sleeping state.

Regardless of from which domain of metaphysics, theology or natural science, our bodies, even if not the best, still be the most rational reference samples in the world. Then, no matter what we can understand, it is always the best way to construct the MCU autonomic system same as a human being.

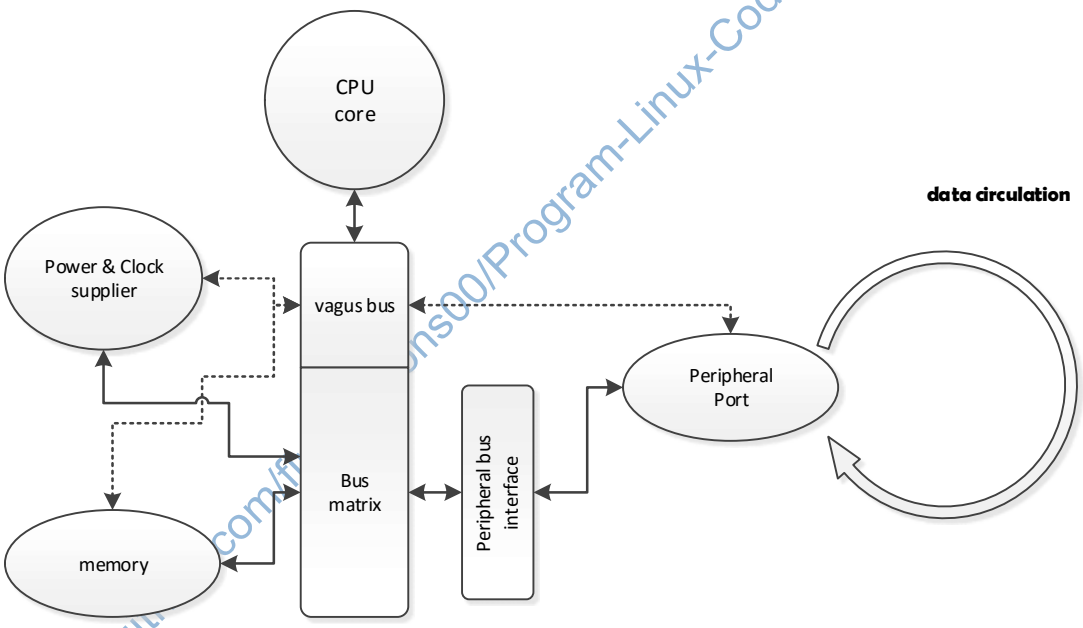

Figure 3-12: Simplified description of MCU autonomic system. The black solid line represent SNS bus and the dotted line represent PNS bus.

Look at figure 3-12, it looks like the skeleton in figure 3-8 but these line are just buses which will carry data stream rather than neural signal. The power and clock supplier act as the heart to provide life-support circulation. The memory acts as stomach and peripheral ports as the larynx and the trachea, then they provide feed-digest circulation for data stream.

The bus matrix acts as spinal cord (central nervous). Because the MCU need a wide cable to carry the high speed data stream, the bus matrix and the bus interface have to been employed to provide a standard for every MCU designer and developer.

https://github.com/fenbisons00/Program-Linux-Code-In-Embedded-Chips

On the other side, the PNS part takes the charge of the work with slow transfer rate in rest or sleep state. So CPU IP generally permit MCU designer apply themselves private definition and can prolong control signal bus from CPU core directly like vagus nerve.

As mentioned before, the SNS and PNS will work separately rather than independently, because there is a relationship of opposite between them. We can separate the two parts completely, but according to the autonomic of human being, it is better to reserve some overlap between them. We can't make a theoretical explanation why to do so, but as mentioned above, our bodies are the most rational.

3.8 Immune Subsystem

I will deduce the immune system with awe. The term of immune comes from historical misunderstanding. The primary purpose of immune system of our bodies is for combination and evolution rather than immunization, which only be a kind of side effect. The same mistake also exist in MCU immune subsystem. This prejudice must be dispelled now for analyzing and understanding the evolution of biological and AI.

No matter whether in biological or AI, it is necessary to obey theory of system kinetics. Each system will change along every input signal. It means combination and evolution actually.

$$\hat{y} = H\hat{x}, \quad \forall \hat{y}, \hat{x} \in \mathbb{R}$$
$$\frac{\partial H}{\partial t} = \hat{x}$$

It means that there isn't linear time invariable system in the world. Any change is a kind of evolution no matter to be better or not.

It will bring us the problem about evolution orient. We only care about whether the system evolve along with our expectation, the kinetic theory only care about the system's stability along with evolving, and MCU immune system only care about itself survival probability.

Whatever we do, we can't let MCU to evolve as our anticipation completely. We only can prepare some servo work to protect that the uncontrolled future damage the system small as possible.

3.8.1 Hardware Part

Hardware part mainly take charge of the exception of hardware failure, such as hardware damage from cosmic-rays, and electronic fault of memory circuit.

https://github.com/fluctuations00/Program-Linux-Coder-In-Embedded-Chips

The primary two methods in hardware part. The one is instruction checker, and the other is opposition between SNS and PNS, as mentioned early. The second method, opposition between SNS and PNS is most mysterious, because this opposition will help system to be rebalance and periodic again after a mistake (perhaps a non-linear signal). Actually, the autonomic system play the same role in human evolution.

3.8.2 Software Part

We will leave the software immune part to operating system.

3.9 Miscellaneous

3.9.1 Multi Purposes of PINs

The desktop computer always extend tentacles through bus arbitrating chip and extended card, such as PCI bus and PCI cards. But the MCU which always be a whole system on one single chip, must realize the extended features by itself to achieve more compacted in size.

Since the PIN number of a MCU is limited, this will conflict with the flexibility required to fit the wide application. For this reason, the MCU's peripheral PINs may be designed to support multiplexing of multiple purpose.

Sometimes the location of some pins perhaps bring the hardware engineer with layout difficulties. Some chip manufactures provide not only multiplexing mentioned before, but also exchangeability between I/O pins. For example, if it seems a little of inharmonious that the circuit in left side of MCU is denser than right, we can change the layout by some pins' exchangeability.

REMARK::
Until now, all of operate systems can't deal with this kind of sophisticated pins' definition well.

https://github.com/fluctuations00/Program-Linux-Code-In-Embedded-Chips

3.9.2 Leaked Current

The pins which will support multiplexing of multiple purpose, should satisfy more than one electronic specification. It will cause the power consumption promoting if we configure these pins incorrect.