Linux Arm Documentation

The kernel development community

Jul 14, 2020

CONTENTS

CHAPTER

ONE

ARM LINUX 2.6 AND UPPER

Please check <ftp://ftp.arm.linux.org.uk/pub/armlinux> for updates.

1.1 Compilation of kernel

In order to compile ARM Linux, you will need a compiler capable of generating ARM ELF code with GNU extensions. GCC 3.3 is known to be a good compiler. Fortunately, you needn't guess. The kernel will report an error if your compiler is a recognized offender.

To build ARM Linux natively, you shouldn't have to alter the ARCH = line in the top level Makefile. However, if you don't have the ARM Linux ELF tools installed as default, then you should change the CROSS_COMPILE line as detailed below.

If you wish to cross-compile, then alter the following lines in the top level make file:

```
ARCH = <whatever>
```

with:

```
ARCH = arm
```

and:

```
CROSS_COMPILE=
```

to:

```
CROSS_COMPILE=<your-path-to-your-compiler-without-gcc>
```

eg.:

```
CROSS_COMPILE=arm-linux-
```

Do a 'make config', followed by 'make Image' to build the kernel (arch/arm/boot/Image). A compressed image can be built by doing a 'make zImage' instead of 'make Image'.

1.2 Bug reports etc

Please send patches to the patch system. For more information, see http://www.arm.linux.org.uk/developer/patches/info.php Always include some explanation as to what the patch does and why it is needed.

Bug reports should be sent to linux-arm-kernel@lists.arm.linux.org.uk, or submitted through the web form at http://www.arm.linux.org.uk/developer/

When sending bug reports, please ensure that they contain all relevant information, eg. the kernel messages that were printed before/during the problem, what you were doing, etc.

1.3 Include files

Several new include directories have been created under include/asm-arm, which are there to reduce the clutter in the top-level directory. These directories, and their purpose is listed below:

arch-*	machine/platform specific header files
hard-ware	driver-internal ARM specific data structures/definitions
mach	descriptions of generic ARM to specific machine interfaces
proc-*	processor dependent header files (currently only two categories)

1.4 Machine/Platform support

The ARM tree contains support for a lot of different machine types. To continue supporting these differences, it has become necessary to split machine-specific parts by directory. For this, the machine category is used to select which directories and files get included (we will use $(MACHINE) to refer to the category)

To this end, we now have arch/arm/mach-$(MACHINE) directories which are designed to house the non-driver files for a particular machine (eg, PCI, memory management, architecture definitions etc). For all future machines, there should be a corresponding arch/arm/mach-$(MACHINE)/include/mach directory.

1.5 Modules

Although modularisation is supported (and required for the FP emulator), each module on an ARM2/ARM250/ARM3 machine when is loaded will take memory up to the next 32k boundary due to the size of the pages. Therefore, is modularisation on these machines really worth it?

However, ARM6 and up machines allow modules to take multiples of 4k, and as such Acorn RiscPCs and other architectures using these processors can make good use of modularisation.

1.6 ADFS Image files

You can access image files on your ADFS partitions by mounting the ADFS partition, and then using the loopback device driver. You must have losetup installed.

Please note that the PCEmulator DOS partitions have a partition table at the start, and as such, you will have to give ‘-o offset’ to losetup.

1.7 Request to developers

When writing device drivers which include a separate assembler file, please include it in with the C file, and not the arch/arm/lib directory. This allows the driver to be compiled as a loadable module without requiring half the code to be compiled into the kernel image.

In general, try to avoid using assembler unless it is really necessary. It makes drivers far less easy to port to other hardware.

1.8 ST506 hard drives

The ST506 hard drive controllers seem to be working fine (if a little slowly). At the moment they will only work off the controllers on an A4x0’s motherboard, but for it to work off a Podule just requires someone with a podule to add the addresses for the IRQ mask and the HDC base to the source.

As of 31/3/96 it works with two drives (you should get the ADFS *configure harddrive set to 2). I’ve got an internal 20MB and a great big external 5.25” FH 64MB drive (who could ever want more :-)).

I’ve just got 240K/s off it (a dd with bs=128k); thats about half of what RiscOS gets; but it’s a heck of a lot better than the 50K/s I was getting last week :-)

Known bug: Drive data errors can cause a hang; including cases where the controller has fixed the error using ECC. (Possibly ONLY in that case ⋯hmm).

1.9 1772 Floppy

This also seems to work OK, but hasn't been stressed much lately. It hasn't got any code for disc change detection in there at the moment which could be a bit of a problem! Suggestions on the correct way to do this are welcome.

1.10 CONFIG_MACH_ and CONFIG_ARCH_

A change was made in 2003 to the macro names for new machines. Historically, CONFIG_ARCH_ was used for the bonafide architecture, e.g. SA1100, as well as implementations of the architecture, e.g. Assabet. It was decided to change the implementation macros to read CONFIG_MACH_ for clarity. Moreover, a retroactive fixup has not been made because it would complicate patching.

Previous registrations may be found online.

<http://www.arm.linux.org.uk/developer/machines/>

1.11 Kernel entry (head.S)

The initial entry into the kernel is via head.S, which uses machine independent code. The machine is selected by the value of 'r1' on entry, which must be kept unique.

Due to the large number of machines which the ARM port of Linux provides for, we have a method to manage this which ensures that we don't end up duplicating large amounts of code.

We group machine (or platform) support code into machine classes. A class typically based around one or more system on a chip devices, and acts as a natural container around the actual implementations. These classes are given directories - arch/arm/mach-<class> and arch/arm/mach-<class> - which contain the source files to/include/mach support the machine class. This directories also contain any machine specific supporting code.

For example, the SA1100 class is based upon the SA1100 and SA1110 SoC devices, and contains the code to support the way the on-board and off- board devices are used, or the device is setup, and provides that machine specific "personality."

For platforms that support device tree (DT), the machine selection is controlled at runtime by passing the device tree blob to the kernel. At compile-time, support for the machine type must be selected. This allows for a single multiplatform kernel build to be used for several machine types.

For platforms that do not use device tree, this machine selection is controlled by the machine type ID, which acts both as a run-time and a

compile-time code selection method. You can register a new machine via the web site at:

<http://www.arm.linux.org.uk/developer/machines/>

Note: Please do not register a machine type for DT-only platforms. If your platform is DT-only, you do not need a registered machine type.

—

Russell King (15/03/2004)

CHAPTER

TWO

BOOTING ARM LINUX

Author: Russell King

Date : 18 May 2002

The following documentation is relevant to 2.4.18-rmk6 and beyond.

In order to boot ARM Linux, you require a boot loader, which is a small program that runs before the main kernel. The boot loader is expected to initialise various devices, and eventually call the Linux kernel, passing information to the kernel.

Essentially, the boot loader should provide (as a minimum) the following:

1. Setup and initialise the RAM.
2. Initialise one serial port.
3. Detect the machine type.
4. Setup the kernel tagged list.
5. Load initramfs.
6. Call the kernel image.

2.1 1. Setup and initialise RAM

Existing boot loaders: MANDATORY

New boot loaders: MANDATORY

The boot loader is expected to find and initialise all RAM that the kernel will use for volatile data storage in the system. It performs this in a machine dependent manner. (It may use internal algorithms to automatically locate and size all RAM, or it may use knowledge of the RAM in the machine, or any other method the boot loader designer sees fit.)

2.2 2. Initialise one serial port

Existing boot loaders: OPTIONAL, RECOMMENDED

New boot loaders: OPTIONAL, RECOMMENDED

The boot loader should initialise and enable one serial port on the target. This allows the kernel serial driver to automatically detect which serial port it should use for the kernel console (generally used for debugging purposes, or communication with the target.)

As an alternative, the boot loader can pass the relevant 'console=' option to the kernel via the tagged lists specifying the port, and serial format options as described in

Documentation/admin-guide/kernel-parameters.rst.

2.3 3. Detect the machine type

Existing boot loaders: OPTIONAL

New boot loaders: MANDATORY except for DT-only platforms

The boot loader should detect the machine type its running on by some method. Whether this is a hard coded value or some algorithm that looks at the connected hardware is beyond the scope of this document. The boot loader must ultimately be able to provide a MACH_TYPE_xxx value to the kernel. (see linux/arch/arm/tools/mach-types). This should be passed to the kernel in register r1.

For DT-only platforms, the machine type will be determined by device tree. set the machine type to all ones (~0). This is not strictly necessary, but assures that it will not match any existing types.

2.4 4. Setup boot data

Existing boot loaders: OPTIONAL, HIGHLY RECOMMENDED

New boot loaders: MANDATORY

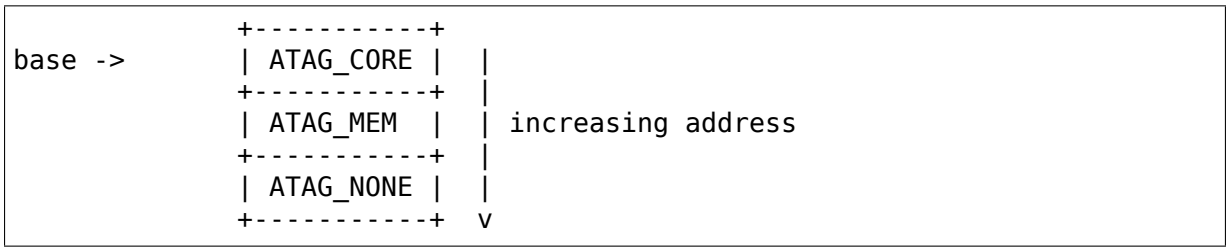
The boot loader must provide either a tagged list or a dtb image for passing configuration data to the kernel. The physical address of the boot data is passed to the kernel in register r2.

2.5 4a. Setup the kernel tagged list

The boot loader must create and initialise the kernel tagged list. A valid tagged list starts with ATAG_CORE and ends with ATAG_NONE. The ATAG_CORE tag may or may not be empty. An empty ATAG_CORE tag has the size field set to '2' (0x00000002). The ATAG_NONE must set the size field to zero.

Any number of tags can be placed in the list. It is undefined whether a repeated tag appends to the information carried by the previous tag, or whether it replaces the information in its entirety; some tags behave as the former, others the latter.

The boot loader must pass at a minimum the size and location of the system memory, and root filesystem location. Therefore, the minimum tagged list should look:



The tagged list should be stored in system RAM.

The tagged list must be placed in a region of memory where neither the kernel decompressor nor initrd 'bootp' program will overwrite it. The recommended placement is in the first 16KiB of RAM.

2.6 4b. Setup the device tree

The boot loader must load a device tree image (dtb) into system ram at a 64bit aligned address and initialize it with the boot data. The dtb format is documented in Documentation/devicetree/booting-without-of.txt. The kernel will look for the dtb magic value of 0xd00dfeed at the dtb physical address to determine if a dtb has been passed instead of a tagged list.

The boot loader must pass at a minimum the size and location of the system memory, and the root filesystem location. The dtb must be placed in a region of memory where the kernel decompressor will not overwrite it, while remaining within the region which will be covered by the kernel's low-memory mapping.

A safe location is just above the 128MiB boundary from start of RAM.

2.7 5. Load initramfs.

Existing boot loaders: OPTIONAL

New boot loaders: OPTIONAL

If an initramfs is in use then, as with the dtb, it must be placed in a region of memory where the kernel decompressor will not overwrite it while also with the region which will be covered by the kernel's low-memory mapping.

A safe location is just above the device tree blob which itself will be loaded just above the 128MiB boundary from the start of RAM as recommended above.

2.8 6. Calling the kernel image

Existing boot loaders: MANDATORY

New boot loaders: MANDATORY

There are two options for calling the kernel zImage. If the zImage is stored in flash, and is linked correctly to be run from flash, then it is legal for the boot loader to call the zImage in flash directly.

The zImage may also be placed in system RAM and called there. The kernel should be placed in the first 128MiB of RAM. It is recommended that it is loaded above 32MiB in order to avoid the need to relocate prior to decompression, which will make the boot process slightly faster.

When booting a raw (non-zImage) kernel the constraints are tighter. In this case the kernel must be loaded at an offset into system equal to TEXT_OFFSET - PAGE_OFFSET.

In any case, the following conditions must be met:

- Quiesce all DMA capable devices so that memory does not get corrupted by bogus network packets or disk data. This will save you many hours of debug.
- CPU register settings
  - r0 = 0,
  - r1 = machine type number discovered in (3) above.
  - r2 = physical address of tagged list in system RAM, or physical address of device tree block (dtb) in system RAM
- CPU mode

  All forms of interrupts must be disabled (IRQs and FIQs)

  For CPUs which do not include the ARM virtualization extensions, the CPU must be in SVC mode. (A special exception exists for Angel)

  CPUs which include support for the virtualization extensions can be entered in HYP mode in order to enable the kernel to make full use of these extensions. This is the recommended boot method for such CPUs, unless the virtualisations are already in use by a pre-installed hypervisor.

If the kernel is not entered in HYP mode for any reason, it must be entered in SVC mode.

- Caches, MMUs

  The MMU must be off.

  Instruction cache may be on or off.

  Data cache must be off.

  If the kernel is entered in HYP mode, the above requirements apply to the HYP mode configuration in addition to the ordinary PL1 (privileged kernel modes) configuration. In addition, all traps into the hypervisor must be disabled, and PL1 access must be granted for all peripherals and CPU resources for which this is architecturally possible. Except for entering in HYP mode, the system configuration should be such that a kernel which does not include support for the virtualization extensions can boot correctly without extra help.

- The boot loader is expected to call the kernel image by jumping directly to the first instruction of the kernel image.

  On CPUs supporting the ARM instruction set, the entry must be made in ARM state, even for a Thumb-2 kernel.

  On CPUs supporting only the Thumb instruction set such as Cortex-M class CPUs, the entry must be made in Thumb state.

CHAPTER

THREE

CLUSTER-WIDE POWER-UP/POWER-DOWN RACE AVOIDANCE ALGORITHM

This file documents the algorithm which is used to coordinate CPU and cluster setup and teardown operations and to manage hardware coherency controls safely.

The section "Rationale" explains what the algorithm is for and why it is needed. "Basic model" explains general concepts using a simplified view of the system. The other sections explain the actual details of the algorithm in use.

3.1 Rationale

In a system containing multiple CPUs, it is desirable to have the ability to turn off individual CPUs when the system is idle, reducing power consumption and thermal dissipation.

In a system containing multiple clusters of CPUs, it is also desirable to have the ability to turn off entire clusters.

Turning entire clusters off and on is a risky business, because it involves performing potentially destructive operations affecting a group of independently running CPUs, while the OS continues to run. This means that we need some coordination in order to ensure that critical cluster-level operations are only performed when it is truly safe to do so.

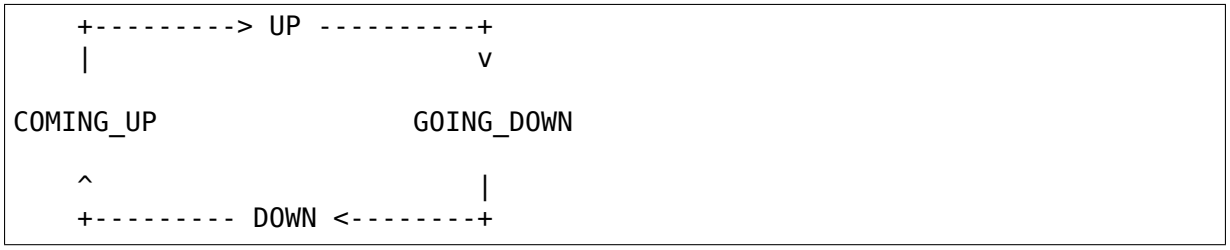
Simple locking may not be sufficient to solve this problem, because mechanisms like Linux spinlocks may rely on coherency mechanisms which are not immediately enabled when a cluster powers up. Since enabling or disabling those mechanisms may itself be a non-atomic operation (such as writing some hardware registers and invalidating large caches), other methods of coordination are required in order to guarantee safe power-down and power-up at the cluster level.

The mechanism presented in this document describes a coherent memory based protocol for performing the needed coordination. It aims to be as lightweight as possible, while providing the required safety properties.

3.2 Basic model

Each cluster and CPU is assigned a state, as follows:

- DOWN
- COMING_UP
- UP
- GOING_DOWN



DOWN: The CPU or cluster is not coherent, and is either powered off or suspended, or is ready to be powered off or suspended.

COMING_UP: The CPU or cluster has committed to moving to the UP state. It may be part way through the process of initialisation and enabling coherency.

UP: The CPU or cluster is active and coherent at the hardware level. A CPU in this state is not necessarily being used actively by the kernel.

GOING_DOWN: The CPU or cluster has committed to moving to the DOWN state. It may be part way through the process of teardown and coherency exit.

Each CPU has one of these states assigned to it at any point in time. The CPU states are described in the "CPU state" section, below.

Each cluster is also assigned a state, but it is necessary to split the state value into two parts (the "cluster" state and "inbound" state) and to introduce additional states in order to avoid races between different CPUs in the cluster simultaneously modifying the state. The cluster- level states are described in the "Cluster state" section.

To help distinguish the CPU states from cluster states in this discussion, the state names are given a CPU_ prefix for the CPU states, and a CLUSTER_ or INBOUND_ prefix for the cluster states.

3.3 CPU state

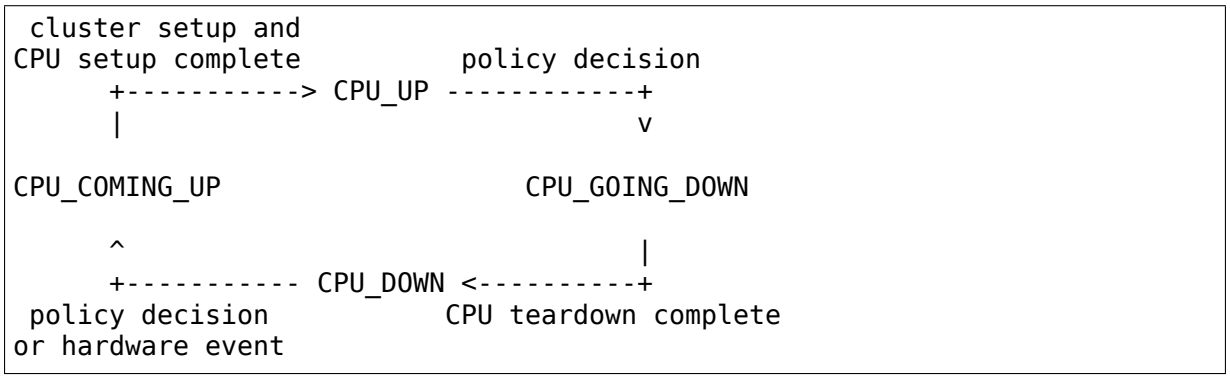
In this algorithm, each individual core in a multi-core processor is referred to as a "CPU". CPUs are assumed to be single-threaded: therefore, a CPU can only be doing one thing at a single point in time.

This means that CPUs fit the basic model closely.

The algorithm defines the following states for each CPU in the system:

- CPU_DOWN

- CPU_COMING_UP
- CPU_UP
- CPU_GOING_DOWN



The definitions of the four states correspond closely to the states of the basic model.

Transitions between states occur as follows.

A trigger event (spontaneous) means that the CPU can transition to the next state as a result of making local progress only, with no requirement for any external event to happen.

CPU_DOWN: A CPU reaches the CPU_DOWN state when it is ready for power-down. On reaching this state, the CPU will typically power itself down or suspend itself, via a WFI instruction or a firmware call.

Next state: CPU_COMING_UP

Conditions: none

Trigger events:

a) an explicit hardware power-up operation, resulting from a policy decision on another CPU;

b) a hardware event, such as an interrupt.

CPU_COMING_UP: A CPU cannot start participating in hardware coherency until the cluster is set up and coherent. If the cluster is not ready, then the CPU will wait in the CPU_COMING_UP state until the cluster has been set up.

Next state: CPU_UP

Conditions: The CPU's parent cluster must be in CLUSTER_UP.

Trigger events: Transition of the parent cluster to CLUSTER_UP.

Refer to the "Cluster state" section for a description of the CLUSTER_UP state.

CPU_UP: When a CPU reaches the CPU_UP state, it is safe for the CPU to start participating in local coherency.

This is done by jumping to the kernel's CPU resume code.

Note that the definition of this state is slightly different from the basic model definition: CPU_UP does not mean that the CPU is coherent yet, but it does

mean that it is safe to resume the kernel. The kernel handles the rest of the resume procedure, so the remaining steps are not visible as part of the race avoidance algorithm.

The CPU remains in this state until an explicit policy decision is made to shut down or suspend the CPU.

Next state: CPU_GOING_DOWN

Conditions: none

Trigger events: explicit policy decision

CPU_GOING_DOWN: While in this state, the CPU exits coherency, including any operations required to achieve this (such as cleaning data caches).

Next state: CPU_DOWN

Conditions: local CPU teardown complete

Trigger events: (spontaneous)

3.4 Cluster state

A cluster is a group of connected CPUs with some common resources. Because a cluster contains multiple CPUs, it can be doing multiple things at the same time. This has some implications. In particular, a CPU can start up while another CPU is tearing the cluster down.

In this discussion, the "outbound side" is the view of the cluster state as seen by a CPU tearing the cluster down. The "inbound side" is the view of the cluster state as seen by a CPU setting the CPU up.

In order to enable safe coordination in such situations, it is important that a CPU which is setting up the cluster can advertise its state independently of the CPU which is tearing down the cluster. For this reason, the cluster state is split into two parts:

"cluster" state: The global state of the cluster; or the state on the outbound side:

- CLUSTER_DOWN
- CLUSTER_UP
- CLUSTER_GOING_DOWN

"inbound" state: The state of the cluster on the inbound side.

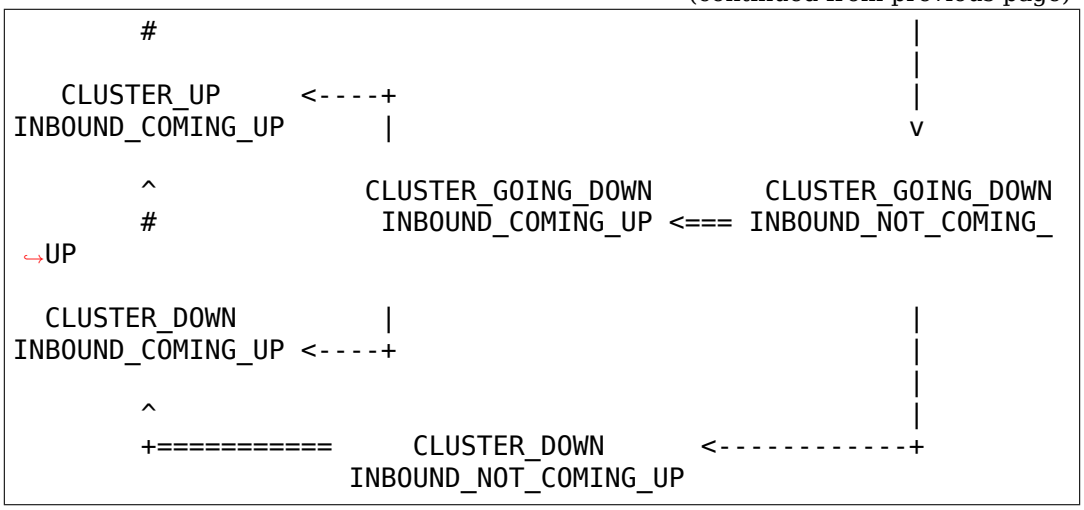
- INBOUND_NOT_COMING_UP
- INBOUND_COMING_UP

The different pairings of these states results in six possible states for the cluster as a whole:

```
                              CLUSTER_UP
        +==========> INBOUND_NOT_COMING_UP -------------+
```

(continues on next page)

(continued from previous page)



Transitions —–> can only be made by the outbound CPU, and only involve changes to the "cluster" state.

Transitions ===##> can only be made by the inbound CPU, and only involve changes to the "inbound" state, except where there is no further transition possible on the outbound side (i.e., the outbound CPU has put the cluster into the CLUSTER_DOWN state).

The race avoidance algorithm does not provide a way to determine which exact CPUs within the cluster play these roles. This must be decided in advance by some other means. Refer to the section "Last man and first man selection" for more explanation.

CLUSTER_DOWN/INBOUND_NOT_COMING_UP is the only state where the cluster can actually be powered down.

The parallelism of the inbound and outbound CPUs is observed by the existence of two different paths from CLUSTER_GOING_DOWN/ INBOUND_NOT_COMING_UP (corresponding to GOING_DOWN in the basic model) to CLUSTER_DOWN/INBOUND_COMING_UP (corresponding to COMING_UP in the basic model). The second path avoids cluster teardown completely.

CLUSTER_UP/INBOUND_COMING_UP is equivalent to UP in the basic model. The final transition to CLUSTER_UP/INBOUND_NOT_COMING_UP is trivial and merely resets the state machine ready for the next cycle.

Details of the allowable transitions follow.

The next state in each case is notated

<cluster state>/<inbound state> (<transitioner>)

where the <transitioner> is the side on which the transition can occur; either the inbound or the outbound side.

CLUSTER_DOWN/INBOUND_NOT_COMING_UP:

Next state: CLUSTER_DOWN/INBOUND_COMING_UP (inbound)

Conditions: none

Trigger events:

a) an explicit hardware power-up operation, resulting from a policy decision on another CPU;

b) a hardware event, such as an interrupt.

CLUSTER_DOWN/INBOUND_COMING_UP:

In this state, an inbound CPU sets up the cluster, including enabling of hardware coherency at the cluster level and any other operations (such as cache invalidation) which are required in order to achieve this.

The purpose of this state is to do sufficient cluster-level setup to enable other CPUs in the cluster to enter coherency safely.

Next state: CLUSTER_UP/INBOUND_COMING_UP (inbound)

Conditions: cluster-level setup and hardware coherency complete

Trigger events: (spontaneous)

CLUSTER_UP/INBOUND_COMING_UP:

Cluster-level setup is complete and hardware coherency is enabled for the cluster. Other CPUs in the cluster can safely enter coherency.

This is a transient state, leading immediately to CLUSTER_UP/INBOUND_NOT_COMING_UP. All other CPUs on the cluster should consider treat these two states as equivalent.

Next state: CLUSTER_UP/INBOUND_NOT_COMING_UP (inbound)

Conditions: none

Trigger events: (spontaneous)

CLUSTER_UP/INBOUND_NOT_COMING_UP:

Cluster-level setup is complete and hardware coherency is enabled for the cluster. Other CPUs in the cluster can safely enter coherency.

The cluster will remain in this state until a policy decision is made to power the cluster down.

Next state: CLUSTER_GOING_DOWN/INBOUND_NOT_COMING_UP (outbound)

Conditions: none

Trigger events: policy decision to power down the cluster

CLUSTER_GOING_DOWN/INBOUND_NOT_COMING_UP:

An outbound CPU is tearing the cluster down. The selected CPU must wait in this state until all CPUs in the cluster are in the CPU_DOWN state.

When all CPUs are in the CPU_DOWN state, the cluster can be torn down, for example by cleaning data caches and exiting cluster-level coherency.

To avoid wasteful unnecessary teardown operations, the outbound should check the inbound cluster state for asynchronous transitions to INBOUND_COMING_UP. Alternatively, individual CPUs can be checked for entry into CPU_COMING_UP or CPU_UP.

Next states:

CLUSTER_DOWN/INBOUND_NOT_COMING_UP (outbound)

Conditions: cluster torn down and ready to power off

Trigger events: (spontaneous)

CLUSTER_GOING_DOWN/INBOUND_COMING_UP (inbound)

Conditions: none

Trigger events:

a) an explicit hardware power-up operation, resulting from a policy decision on another CPU;

b) a hardware event, such as an interrupt.

CLUSTER_GOING_DOWN/INBOUND_COMING_UP:

The cluster is (or was) being torn down, but another CPU has come online in the meantime and is trying to set up the cluster again.

If the outbound CPU observes this state, it has two choices:

a) back out of teardown, restoring the cluster to the CLUSTER_UP state;

b) finish tearing the cluster down and put the cluster in the CLUSTER_DOWN state; the inbound CPU will set up the cluster again from there.

Choice (a) permits the removal of some latency by avoiding unnecessary teardown and setup operations in situations where the cluster is not really going to be powered down.

Next states:

CLUSTER_UP/INBOUND_COMING_UP (outbound)

Conditions: cluster-level setup and hardware coherency complete

Trigger events: (spontaneous)

CLUSTER_DOWN/INBOUND_COMING_UP (outbound)

Conditions: cluster torn down and ready to power off

Trigger events: (spontaneous)

3.5 Last man and First man selection

The CPU which performs cluster tear-down operations on the outbound side is commonly referred to as the “last man”.

The CPU which performs cluster setup on the inbound side is commonly referred to as the “first man”.

The race avoidance algorithm documented above does not provide a mechanism to choose which CPUs should play these roles.

Last man:

When shutting down the cluster, all the CPUs involved are initially executing Linux and hence coherent. Therefore, ordinary spinlocks can be used to select a last man safely, before the CPUs become non-coherent.

First man:

Because CPUs may power up asynchronously in response to external wake-up events, a dynamic mechanism is needed to make sure that only one CPU attempts to play the first man role and do the cluster-level initialisation: any other CPUs must wait for this to complete before proceeding.

Cluster-level initialisation may involve actions such as configuring coherency controls in the bus fabric.

The current implementation in mcpm_head.S uses a separate mutual exclusion mechanism to do this arbitration. This mechanism is documented in detail in vlocks.txt.

3.6 Features and Limitations

Implementation:

The current ARM-based implementation is split between arch/arm/common/mcpm_head.S (low-level inbound CPU operations) and arch/arm/common/mcpm_entry.c (everything else):

__mcpm_cpu_going_down() signals the transition of a CPU to the CPU_GOING_DOWN state.

__mcpm_cpu_down() signals the transition of a CPU to the CPU_DOWN state.

A CPU transitions to CPU_COMING_UP and then to CPU_UP via the low-level power-up code in mcpm_head.S. This could involve CPU-specific setup code, but in the current implementation it does not.

__mcpm_outbound_enter_critical() and __mcpm_outbound_leave_critical() handle transitions from CLUSTER_UP to CLUSTER_GOING_DOWN and from there to CLUSTER_DOWN or back to CLUSTER_UP (in the case of an aborted cluster power-down).

These functions are more complex than the __mcpm_cpu_*() functions due to the extra inter-CPU coordination which is needed for safe transitions at the cluster level.

A cluster transitions from CLUSTER_DOWN back to CLUSTER_UP via the low-level power-up code in mcpm_head.S. This typically involves platform-specific setup code, provided by the platform-specific power_up_setup function registered via mcpm_sync_init.

Deep topologies:

As currently described and implemented, the algorithm does not support CPU topologies involving more than two levels (i.e., clusters of clusters are not supported). The algorithm could be extended by replicating the cluster-level states for the additional topological levels, and modifying the transition rules for the intermediate (non-outermost) cluster levels.

3.7 Colophon

Originally created and documented by Dave Martin for Linaro Limited, in collaboration with Nicolas Pitre and Achin Gupta.

Copyright (C) 2012-2013 Linaro Limited Distributed under the terms of Version 2 of the GNU General Public License, as defined in linux/COPYING.

CHAPTER

FOUR

INTERFACE FOR REGISTERING AND CALLING FIRMWARE-SPECIFIC OPERATIONS FOR ARM

Written by Tomasz Figa <t.figa@samsung.com>

Some boards are running with secure firmware running in TrustZone secure world, which changes the way some things have to be initialized. This makes a need to provide an interface for such platforms to specify available firmware operations and call them when needed.

Firmware operations can be specified by filling in a struct firmware_ops with appropriate callbacks and then registering it with register_firmware_ops() function:

```
void register_firmware_ops(const struct firmware_ops *ops)
```

The ops pointer must be non-NULL. More information about struct firmware_ops and its members can be found in arch/arm/include/asm/firmware.h header.

There is a default, empty set of operations provided, so there is no need to set anything if platform does not require firmware operations.

To call a firmware operation, a helper macro is provided:

```
#define call_firmware_op(op, ...)                                       \
        ((firmware_ops->op) ? firmware_ops->op(__VA_ARGS__) : (-ENOSYS))
```

the macro checks if the operation is provided and calls it or otherwise returns -ENOSYS to signal that given operation is not available (for example, to allow fallback to legacy operation).

Example of registering firmware operations:

```
/* board file */

static int platformX_do_idle(void)
{
        /* tell platformX firmware to enter idle */
        return 0;
}

static int platformX_cpu_boot(int i)
{
        /* tell platformX firmware to boot CPU i */
        return 0;
}
```

(continues on next page)

(continued from previous page)

```
static const struct firmware_ops platformX_firmware_ops = {
        .do_idle        = exynos_do_idle,
        .cpu_boot       = exynos_cpu_boot,
        /* other operations not available on platformX */
};

/* init_early callback of machine descriptor */
static void __init board_init_early(void)
{
        register_firmware_ops(&platformX_firmware_ops);
}
```

Example of using a firmware operation:

```
/* some platform code, e.g. SMP initialization */

__raw_writel(__pa_symbol(exynos4_secondary_startup),
        CPU1_BOOT_REG);

/* Call Exynos specific smc call */
if (call_firmware_op(cpu_boot, cpu) == -ENOSYS)
        cpu_boot_legacy(...); /* Try legacy way */

gic_raise_softirq(cpumask_of(cpu), 1);
```

CHAPTER

FIVE

INTERRUPTS

2.5.2-rmk5: This is the first kernel that contains a major shake up of some of the major architecture-specific subsystems.

Firstly, it contains some pretty major changes to the way we handle the MMU TLB. Each MMU TLB variant is now handled completely separately - we have TLB v3, TLB v4 (without write buffer), TLB v4 (with write buffer), and finally TLB v4 (with write buffer, with I TLB invalidate entry). There is more assembly code inside each of these functions, mainly to allow more flexible TLB handling for the future.

Secondly, the IRQ subsystem.

The 2.5 kernels will be having major changes to the way IRQs are handled. Unfortunately, this means that machine types that touch the irq_desc[] array (basically all machine types) will break, and this means every machine type that we currently have.

Lets take an example. On the Assabet with Neponset, we have:

```
          GPIO25                  IRR:2
SA1100 ------------> Neponset -----------> SA1111
                                  IIR:1
                               -----------> USAR
                                  IIR:0
                               -----------> SMC9196
```

The way stuff currently works, all SA1111 interrupts are mutually exclusive of each other - if you’re processing one interrupt from the SA1111 and another comes in, you have to wait for that interrupt to finish processing before you can service the new interrupt. Eg, an IDE PIO-based interrupt on the SA1111 excludes all other SA1111 and SMC9196 interrupts until it has finished transferring its multi-sector data, which can be a long time. Note also that since we loop in the SA1111 IRQ handler, SA1111 IRQs can hold off SMC9196 IRQs indefinitely.

The new approach brings several new ideas···

We introduce the concept of a “parent”and a “child”. For example, to the Neponset handler, the “parent” is GPIO25, and the “children” d are SA1111, SMC9196 and USAR.

We also bring the idea of an IRQ “chip” (mainly to reduce the size of the irqdesc array). This doesn’t have to be a real “IC”; indeed the SA11x0 IRQs are handled by two separate “chip” structures, one for GPIO0-10, and another for all the rest. It is just a container for the various operations (maybe this’ll change to a better name). This structure has the following operations:

```
struct irqchip {
        /*
         * Acknowledge the IRQ.
         * If this is a level-based IRQ, then it is expected to mask the␣
↪IRQ
         * as well.
         */
        void (*ack)(unsigned int irq);
        /*
         * Mask the IRQ in hardware.
         */
        void (*mask)(unsigned int irq);
        /*
         * Unmask the IRQ in hardware.
         */
        void (*unmask)(unsigned int irq);
        /*
         * Re-run the IRQ
         */
        void (*rerun)(unsigned int irq);
        /*
         * Set the type of the IRQ.
         */
        int (*type)(unsigned int irq, unsigned int, type);
};
```

ack

- required. May be the same function as mask for IRQs handled by do_level_IRQ.

mask

- required.

unmask

- required.

rerun

- optional. Not required if you’re using do_level_IRQ for all IRQs that use this ‘irqchip’. Generally expected to re-trigger the hardware IRQ if possible. If not, may call the handler directly.

type

- optional. If you don’t support changing the type of an IRQ, it should be null so people can detect if they are unable to set the IRQ type.

For each IRQ, we keep the following information:

- “disable” depth (number of disable_irq()s without enable_irq()s)
- flags indicating what we can do with this IRQ (valid, probe, noautounmask) as before
- status of the IRQ (probing, enable, etc)
- chip

- per-IRQ handler
- irqaction structure list

The handler can be one of the 3 standard handlers - “level” , “edge” and “simple” , or your own specific handler if you need to do something special.

The “level”handler is what we currently have - its pretty simple. “edge”knows about the brokenness of such IRQ implementations - that you need to leave the hardware IRQ enabled while processing it, and queueing further IRQ events should the IRQ happen again while processing. The “simple” handler is very basic, and does not perform any hardware manipulation, nor state tracking. This is useful for things like the SMC9196 and USAR above.

5.1 So, what’s changed?

1. Machine implementations must not write to the irqdesc array.
2. New functions to manipulate the irqdesc array. The first 4 are expected to be useful only to machine specific code. The last is recommended to only be used by machine specific code, but may be used in drivers if absolutely necessary.

   set_irq_chip(irq,chip) Set the mask/unmask methods for handling this IRQ

   set_irq_handler(irq,handler) Set the handler for this IRQ (level, edge, simple)

   set_irq_chained_handler(irq,handler) Set a “chained” handler for this IRQ - automatically enables this IRQ (eg, Neponset and SA1111 handlers).

   set_irq_flags(irq,flags) Set the valid/probe/noautoenable flags.

   set_irq_type(irq,type) Set active the IRQ edge(s)/level. This replaces the SA1111 INTPOL manipulation, and the set_GPIO_IRQ_edge() function. Type should be one of IRQ_TYPE_xxx defined in <linux/irq.h>
3. set_GPIO_IRQ_edge() is obsolete, and should be replaced by set_irq_type.
4. Direct access to SA1111 INTPOL is deprecated. Use set_irq_type instead.
5. A handler is expected to perform any necessary acknowledgement of the parent IRQ via the correct chip specific function. For instance, if the SA1111 is directly connected to a SA1110 GPIO, then you should acknowledge the SA1110 IRQ each time you re-read the SA1111 IRQ status.
6. For any child which doesn’t have its own IRQ enable/disable controls (eg, SMC9196), the handler must mask or acknowledge the parent IRQ while the child handler is called, and the child handler should be the “simple” handler (not “edge” nor “level”). After the handler completes, the parent IRQ should be unmasked, and the status of all children must be re-checked for pending events. (see the Neponset IRQ handler for details).
7. fixup_irq() is gone, as is arch/arm/mach-*/include/mach/irq.h

Please note that this will not solve all problems - some of them are hardware based. Mixing level-based and edge-based IRQs on the same parent signal (eg neponset) is one such area where a software based solution can't provide the full answer to low IRQ latency.

CHAPTER

SIX

KERNEL MODE NEON

6.1 TL;DR summary

- Use only NEON instructions, or VFP instructions that don't rely on support code
- Isolate your NEON code in a separate compilation unit, and compile it with '-march=armv7-a -mfpu=neon -mfloat-abi=softfp'
- Put kernel_neon_begin() and kernel_neon_end() calls around the calls into your NEON code
- Don't sleep in your NEON code, and be aware that it will be executed with preemption disabled

6.2 Introduction

It is possible to use NEON instructions (and in some cases, VFP instructions) in code that runs in kernel mode. However, for performance reasons, the NEON/VFP register file is not preserved and restored at every context switch or taken exception like the normal register file is, so some manual intervention is required. Furthermore, special care is required for code that may sleep [i.e., may call schedule()], as NEON or VFP instructions will be executed in a non-preemptible section for reasons outlined below.

6.3 Lazy preserve and restore

The NEON/VFP register file is managed using lazy preserve (on UP systems) and lazy restore (on both SMP and UP systems). This means that the register file is kept 'live', and is only preserved and restored when multiple tasks are contending for the NEON/VFP unit (or, in the SMP case, when a task migrates to another core). Lazy restore is implemented by disabling the NEON/VFP unit after every context switch, resulting in a trap when subsequently a NEON/VFP instruction is issued, allowing the kernel to step in and perform the restore if necessary.

Any use of the NEON/VFP unit in kernel mode should not interfere with this, so it is required to do an 'eager' preserve of the NEON/VFP register file, and enable the NEON/VFP unit explicitly so no exceptions are generated on first subsequent use. This is handled by the function kernel_neon_begin(), which should be called before

any kernel mode NEON or VFP instructions are issued. Likewise, the NEON/VFP unit should be disabled again after use to make sure user mode will hit the lazy restore trap upon next use. This is handled by the function kernel_neon_end().

6.4 Interruptions in kernel mode

For reasons of performance and simplicity, it was decided that there shall be no preserve/restore mechanism for the kernel mode NEON/VFP register contents. This implies that interruptions of a kernel mode NEON section can only be allowed if they are guaranteed not to touch the NEON/VFP registers. For this reason, the following rules and restrictions apply in the kernel: * NEON/VFP code is not allowed in interrupt context; * NEON/VFP code is not allowed to sleep; * NEON/VFP code is executed with preemption disabled.

If latency is a concern, it is possible to put back to back calls to kernel_neon_end() and kernel_neon_begin() in places in your code where none of the NEON registers are live. (Additional calls to kernel_neon_begin() should be reasonably cheap if no context switch occurred in the meantime)

6.5 VFP and support code

Earlier versions of VFP (prior to version 3) rely on software support for things like IEEE-754 compliant underflow handling etc. When the VFP unit needs such software assistance, it signals the kernel by raising an undefined instruction exception. The kernel responds by inspecting the VFP control registers and the current instruction and arguments, and emulates the instruction in software.

Such software assistance is currently not implemented for VFP instructions executed in kernel mode. If such a condition is encountered, the kernel will fail and generate an OOPS.

6.6 Separating NEON code from ordinary code

The compiler is not aware of the special significance of kernel_neon_begin() and kernel_neon_end(), i.e., that it is only allowed to issue NEON/VFP instructions between calls to these respective functions. Furthermore, GCC may generate NEON instructions of its own at -O3 level if -mfpu=neon is selected, and even if the kernel is currently compiled at -O2, future changes may result in NEON/VFP instructions appearing in unexpected places if no special care is taken.

Therefore, the recommended and only supported way of using NEON/VFP in the kernel is by adhering to the following rules:

- isolate the NEON code in a separate compilation unit and compile it with ‘-march=armv7-a -mfpu=neon -mfloat-abi=softfp’;
- issue the calls to kernel_neon_begin(), kernel_neon_end() as well as the calls into the unit containing the NEON code from a compilation unit which is not built with the GCC flag ‘-mfpu=neon’ set.

As the kernel is compiled with ‘-msoft-float’, the above will guarantee that both NEON and VFP instructions will only ever appear in designated compilation units at any optimization level.

6.7 NEON assembler

NEON assembler is supported with no additional caveats as long as the rules above are followed.

6.8 NEON code generated by GCC

The GCC option -ftree-vectorize (implied by -O3) tries to exploit implicit parallelism, and generates NEON code from ordinary C source code. This is fully supported as long as the rules above are followed.

6.9 NEON intrinsics

NEON intrinsics are also supported. However, as code using NEON intrinsics relies on the GCC header <arm_neon.h>, (which #includes <stdint.h>), you should observe the following in addition to the rules above:

- Compile the unit containing the NEON intrinsics with ‘-ffreestanding’ so GCC uses its builtin version of <stdint.h> (this is a C99 header which the kernel does not supply);
- Include <arm_neon.h> last, or at least after <linux/types.h>

CHAPTER

SEVEN

KERNEL-PROVIDED USER HELPERS

These are segment of kernel provided user code reachable from user space at a fixed address in kernel memory. This is used to provide user space with some operations which require kernel help because of unimplemented native feature and/or instructions in many ARM CPUs. The idea is for this code to be executed directly in user mode for best efficiency but which is too intimate with the kernel counter part to be left to user libraries. In fact this code might even differ from one CPU to another depending on the available instruction set, or whether it is a SMP systems. In other words, the kernel reserves the right to change this code as needed without warning. Only the entry points and their results as documented here are guaranteed to be stable.

This is different from (but doesn't preclude) a full blown VDSO implementation, however a VDSO would prevent some assembly tricks with constants that allows for efficient branching to those code segments. And since those code segments only use a few cycles before returning to user code, the overhead of a VDSO indirect far call would add a measurable overhead to such minimalistic operations.

User space is expected to bypass those helpers and implement those things inline (either in the code emitted directly by the compiler, or part of the implementation of a library call) when optimizing for a recent enough processor that has the necessary native support, but only if resulting binaries are already to be incompatible with earlier ARM processors due to usage of similar native instructions for other things. In other words don't make binaries unable to run on earlier processors just for the sake of not using these kernel helpers if your compiled code is not going to use new instructions for other purpose.

New helpers may be added over time, so an older kernel may be missing some helpers present in a newer kernel. For this reason, programs must check the value of __kuser_helper_version (see below) before assuming that it is safe to call any particular helper. This check should ideally be performed only once at process startup time, and execution aborted early if the required helpers are not provided by the kernel version that process is running on.

7.1 kuser_helper_version

Location: 0xffff0ffc

Reference declaration:

```
extern int32_t __kuser_helper_version;
```

Definition:

This field contains the number of helpers being implemented by the running kernel. User space may read this to determine the availability of a particular helper.

Usage example:

```
#define __kuser_helper_version (*(int32_t *)0xffff0ffc)

void check_kuser_version(void)
{
      if (__kuser_helper_version < 2) {
            fprintf(stderr, "can't do atomic operations, kernel too old\n
↪");
            abort();
      }
}
```

Notes:

User space may assume that the value of this field never changes during the lifetime of any single process. This means that this field can be read once during the initialisation of a library or startup phase of a program.

7.2 kuser_get_tls

Location: 0xffff0fe0

Reference prototype:

```
void * __kuser_get_tls(void);
```

Input:

lr = return address

Output:

r0 = TLS value

Clobbered registers:

none

Definition:

Get the TLS value as previously set via the __ARM_NR_set_tls syscall.

Usage example:

```
typedef void * (__kuser_get_tls_t)(void);
#define __kuser_get_tls (*(__kuser_get_tls_t *)0xffff0fe0)

void foo()
{
      void *tls = __kuser_get_tls();
      printf("TLS = %p\n", tls);
}
```

Notes:

- Valid only if __kuser_helper_version >= 1 (from kernel version 2.6.12).

7.3 kuser_cmpxchg

Location: 0xffff0fc0

Reference prototype:

```
int __kuser_cmpxchg(int32_t oldval, int32_t newval, volatile int32_t *ptr);
```

Input:

r0 = oldval r1 = newval r2 = ptr lr = return address

Output:

r0 = success code (zero or non-zero) C flag = set if r0 == 0, clear if r0 != 0

Clobbered registers:

r3, ip, flags

Definition:

Atomically store newval in *ptr only if *ptr is equal to oldval. Return zero if *ptr was changed or non-zero if no exchange happened. The C flag is also set if *ptr was changed to allow for assembly optimization in the calling code.

Usage example:

```
typedef int (__kuser_cmpxchg_t)(int oldval, int newval, volatile int *ptr);
#define __kuser_cmpxchg (*(__kuser_cmpxchg_t *)0xffff0fc0)

int atomic_add(volatile int *ptr, int val)
{
      int old, new;

      do {
              old = *ptr;
              new = old + val;
      } while(__kuser_cmpxchg(old, new, ptr));

      return new;
}
```

Notes:

- This routine already includes memory barriers as needed.
- Valid only if __kuser_helper_version >= 2 (from kernel version 2.6.12).

7.4 kuser_memory_barrier

Location: 0xffff0fa0

Reference prototype:

```
void __kuser_memory_barrier(void);
```

Input:

lr = return address

Output:

none

Clobbered registers:

none

Definition:

Apply any needed memory barrier to preserve consistency with data modified manually and __kuser_cmpxchg usage.

Usage example:

```
typedef void (__kuser_dmb_t)(void);
#define __kuser_dmb (*(__kuser_dmb_t *)0xffff0fa0)
```

Notes:

- Valid only if __kuser_helper_version >= 3 (from kernel version 2.6.15).

7.5 kuser_cmpxchg64

Location: 0xffff0f60

Reference prototype:

```
int __kuser_cmpxchg64(const int64_t *oldval,
                      const int64_t *newval,
                      volatile int64_t *ptr);
```

Input:

r0 = pointer to oldval r1 = pointer to newval r2 = pointer to target value
lr = return address

Output:

r0 = success code (zero or non-zero) C flag = set if r0 == 0, clear if r0 != 0

Clobbered registers:

r3, lr, flags

Definition:

Atomically store the 64-bit value pointed by *newval in *ptr only if *ptr is equal to the 64-bit value pointed by *oldval. Return zero if *ptr was changed or non-zero if no exchange happened.

The C flag is also set if *ptr was changed to allow for assembly optimization in the calling code.

Usage example:

```
typedef int (__kuser_cmpxchg64_t)(const int64_t *oldval,
                                  const int64_t *newval,
                                  volatile int64_t *ptr);
#define __kuser_cmpxchg64 (*(__kuser_cmpxchg64_t *)0xffff0f60)

int64_t atomic_add64(volatile int64_t *ptr, int64_t val)
{
      int64_t old, new;

      do {
              old = *ptr;
              new = old + val;
      } while(__kuser_cmpxchg64(&old, &new, ptr));

      return new;
}
```

Notes:

- This routine already includes memory barriers as needed.
- Due to the length of this sequence, this spans 2 conventional kuser “slots”, therefore 0xffff0f80 is not used as a valid entry point.
- Valid only if __kuser_helper_version >= 5 (from kernel version 3.1).

CHAPTER

EIGHT

KERNEL MEMORY LAYOUT ON ARM LINUX

Russell King <rmk@arm.linux.org.uk>

November 17, 2005 (2.6.15)

This document describes the virtual memory layout which the Linux kernel uses for ARM processors. It indicates which regions are free for platforms to use, and which are used by generic code.

The ARM CPU is capable of addressing a maximum of 4GB virtual memory space, and this must be shared between user space processes, the kernel, and hardware devices.

As the ARM architecture matures, it becomes necessary to reserve certain regions of VM space for use for new facilities; therefore this document may reserve more VM space over time.

Start	End	Use
ffff8000	ffffffff	copy_user_page / clear_user_page use. For SA11xx and Xscale, this is used to setup a minicache mapping.
ffff4000	ffffffff	cache aliasing on ARMv6 and later CPUs.
ffff1000	ffff7fff	Reserved. Platforms must not use this address range.
ffff0000	ffff0fff	CPU vector page. The CPU vectors are mapped here if the CPU supports vector relocation (control register V bit.)
fffe0000	fffeffff	XScale cache flush area. This is used in proc-xscale.S to flush the whole data cache. (XScale does not have TCM.)
fffe8000	fffeffff	DTCM mapping area for platforms with DTCM mounted inside the CPU.
fffe0000	fffe7fff	ITCM mapping area for platforms with ITCM mounted inside the CPU.
ffc00000	feffffff	Fixmap mapping region. Addresses provided by fix_to_virt() will be located here.
fee00000	feffffff	Mapping of PCI I/O space. This is a static mapping within the vmalloc space.
VMALLOC_START	VMALLOC_END-1	vmalloc() / ioremap() space. Memory returned by vmalloc/ioremap will be dynamically placed in this region. Machine specific static mappings are also located here through iotable_init(). VMALLOC_START is based upon the value of the high_memory variable, and VMALLOC_END is equal to 0xff800000.
PAGE_OFFSET	high_memory-1	Kernel direct-mapped RAM region. This maps the platforms RAM, and typically maps all platform RAM in a 1:1 relationship.
PKMAP_BASE	PAGE_OFFSET-1	Permanent kernel mappings One way of mapping HIGHMEM pages into kernel space.
MODULES_VADDR	MODULES_END-1	Kernel module space Kernel modules inserted via insmod are placed here using dynamic mappings.
00001000	TASK_SIZE-1	User space mappings Per-thread mappings are placed here via the mmap() system call.
00000000	0000ffff	CPU vector page / null pointer trap CPUs which do not support vector remapping place their vector page here. NULL pointer dereferences by both the kernel and user space are also caught via this mapping.

Please note that mappings which collide with the above areas may result in a non-bootable kernel, or may cause the kernel to (eventually) panic at run time.

Since future CPUs may impact the kernel mapping layout, user programs must not access any memory which is not mapped inside their 0x0001000 to TASK_SIZE address range. If they wish to access these areas, they must set up their own mappings using open() and mmap().

CHAPTER

NINE

MEMORY ALIGNMENT

Too many problems popped up because of unnoticed misaligned memory access in kernel code lately. Therefore the alignment fixup is now unconditionally configured in for SA11x0 based targets. According to Alan Cox, this is a bad idea to configure it out, but Russell King has some good reasons for doing so on some f***ed up ARM architectures like the EBSA110. However this is not the case on many design I'm aware of, like all SA11x0 based ones.

Of course this is a bad idea to rely on the alignment trap to perform unaligned memory access in general. If those access are predictable, you are better to use the macros provided by include/asm/unaligned.h. The alignment trap can fixup misaligned access for the exception cases, but at a high performance cost. It better be rare.

Now for user space applications, it is possible to configure the alignment trap to SIGBUS any code performing unaligned access (good for debugging bad code), or even fixup the access by software like for kernel code. The later mode isn't recommended for performance reasons (just think about the floating point emulation that works about the same way). Fix your code instead!

Please note that randomly changing the behaviour without good thought is real bad - it changes the behaviour of all unaligned instructions in user space, and might cause programs to fail unexpectedly.

To change the alignment trap behavior, simply echo a number into /proc/cpu/alignment. The number is made up from various bits:

bit	behavior when set
0	A user process performing an unaligned memory access will cause the kernel to print a message indicating process name, pid, pc, instruction, address, and the fault code.
1	The kernel will attempt to fix up the user process performing the unaligned access. This is of course slow (think about the floating point emulator) and not recommended for production use.
2	The kernel will send a SIGBUS signal to the user process performing the unaligned access.

Note that not all combinations are supported - only values 0 through 5. (6 and 7 don't make sense).

For example, the following will turn on the warnings, but without fixing up or sending SIGBUS signals:

```
echo 1 > /proc/cpu/alignment
```

You can also read the content of the same file to get statistical information on unaligned access occurrences plus the current mode of operation for user space code.

Nicolas Pitre, Mar 13, 2001. Modified Russell King, Nov 30, 2001.

CHAPTER

TEN

ARM TCM (TIGHTLY-COUPLED MEMORY) HANDLING IN LINUX

Written by Linus Walleij <linus.walleij@stericsson.com>

Some ARM SoCs have a so-called TCM (Tightly-Coupled Memory). This is usually just a few (4-64) KiB of RAM inside the ARM processor.

Due to being embedded inside the CPU, the TCM has a Harvard-architecture, so there is an ITCM (instruction TCM) and a DTCM (data TCM). The DTCM can not contain any instructions, but the ITCM can actually contain data. The size of DTCM or ITCM is minimum 4KiB so the typical minimum configuration is 4KiB ITCM and 4KiB DTCM.

ARM CPUs have special registers to read out status, physical location and size of TCM memories. arch/arm/include/asm/cputype.h defines a CPUID_TCM register that you can read out from the system control coprocessor. Documentation from ARM can be found at http://infocenter.arm.com, search for "TCM Status Register" to see documents for all CPUs. Reading this register you can determine if ITCM (bits 1-0) and/or DTCM (bit 17-16) is present in the machine.

There is further a TCM region register (search for "TCM Region Registers" at the ARM site) that can report and modify the location size of TCM memories at run-time. This is used to read out and modify TCM location and size. Notice that this is not a MMU table: you actually move the physical location of the TCM around. At the place you put it, it will mask any underlying RAM from the CPU so it is usually wise not to overlap any physical RAM with the TCM.

The TCM memory can then be remapped to another address again using the MMU, but notice that the TCM if often used in situations where the MMU is turned off. To avoid confusion the current Linux implementation will map the TCM 1 to 1 from physical to virtual memory in the location specified by the kernel. Currently Linux will map ITCM to 0xfffe0000 and on, and DTCM to 0xfffe8000 and on, supporting a maximum of 32KiB of ITCM and 32KiB of DTCM.

Newer versions of the region registers also support dividing these TCMs in two separate banks, so for example an 8KiB ITCM is divided into two 4KiB banks with its own control registers. The idea is to be able to lock and hide one of the banks for use by the secure world (TrustZone).

TCM is used for a few things:

- FIQ and other interrupt handlers that need deterministic timing and cannot wait for cache misses.

- Idle loops where all external RAM is set to self-refresh retention mode, so only on-chip RAM is accessible by the CPU and then we hang inside ITCM waiting for an interrupt.
- Other operations which implies shutting off or reconfiguring the external RAM controller.

There is an interface for using TCM on the ARM architecture in <asm/tcm.h>. Using this interface it is possible to:

- Define the physical address and size of ITCM and DTCM.
- Tag functions to be compiled into ITCM.
- Tag data and constants to be allocated to DTCM and ITCM.
- Have the remaining TCM RAM added to a special allocation pool with gen_pool_create() and gen_pool_add() and provice tcm_alloc() and tcm_free() for this memory. Such a heap is great for things like saving device state when shutting off device power domains.

A machine that has TCM memory shall select HAVE_TCM from arch/arm/Kconfig for itself. Code that needs to use TCM shall #include <asm/tcm.h>

Functions to go into itcm can be tagged like this: int __tcmfunc foo(int bar);

Since these are marked to become long_calls and you may want to have functions called locally inside the TCM without wasting space, there is also the __tcmlocalfunc prefix that will make the call relative.

Variables to go into dtcm can be tagged like this:

```
int __tcmdata foo;
```

Constants can be tagged like this:

```
int __tcmconst foo;
```

To put assembler into TCM just use:

```
.section ".tcm.text" or .section ".tcm.data"
```

respectively.

Example code:

```
#include <asm/tcm.h>

/* Uninitialized data */
static u32 __tcmdata tcmvar;
/* Initialized data */
static u32 __tcmdata tcmassigned = 0x2BADBABEU;
/* Constant */
static const u32 __tcmconst tcmconst = 0xCAFEBABEU;

static void __tcmlocalfunc tcm_to_tcm(void)
{
      int i;
      for (i = 0; i < 100; i++)
```

(continues on next page)

(continued from previous page)

```
                tcmvar ++;
}

static void __tcmfunc hello_tcm(void)
{
        /* Some abstract code that runs in ITCM */
        int i;
        for (i = 0; i < 100; i++) {
                tcmvar ++;
        }
        tcm_to_tcm();
}

static void __init test_tcm(void)
{
        u32 *tcmem;
        int i;

        hello_tcm();
        printk("Hello TCM executed from ITCM RAM\n");

        printk("TCM variable from testrun: %u @ %p\n", tcmvar, &tcmvar);
        tcmvar = 0xDEADBEEFU;
        printk("TCM variable: 0x%x @ %p\n", tcmvar, &tcmvar);

        printk("TCM assigned variable: 0x%x @ %p\n", tcmassigned, &
↪tcmassigned);

        printk("TCM constant: 0x%x @ %p\n", tcmconst, &tcmconst);

        /* Allocate some TCM memory from the pool */
        tcmem = tcm_alloc(20);
        if (tcmem) {
                printk("TCM Allocated 20 bytes of TCM @ %p\n", tcmem);
                tcmem[0] = 0xDEADBEEFU;
                tcmem[1] = 0x2BADBABEU;
                tcmem[2] = 0xCAFEBABEU;
                tcmem[3] = 0xDEADBEEFU;
                tcmem[4] = 0x2BADBABEU;
                for (i = 0; i < 5; i++)
                        printk("TCM tcmem[%d] = %08x\n", i, tcmem[i]);
                tcm_free(tcmem, 20);
        }
}
```

CHAPTER

ELEVEN

KERNEL INITIALISATION PARAMETERS ON ARM LINUX

The following document describes the kernel initialisation parameter structure, otherwise known as 'struct param_struct' which is used for most ARM Linux architectures.

This structure is used to pass initialisation parameters from the kernel loader to the Linux kernel proper, and may be short lived through the kernel initialisation process. As a general rule, it should not be referenced outside of arch/arm/kernel/setup.c:setup_arch().

There are a lot of parameters listed in there, and they are described below:

page_size This parameter must be set to the page size of the machine, and will be checked by the kernel.

nr_pages This is the total number of pages of memory in the system. If the memory is banked, then this should contain the total number of pages in the system.

If the system contains separate VRAM, this value should not include this information.

ramdisk_size This is now obsolete, and should not be used.

flags Various kernel flags, including:

bit 0	1 = mount root read only
bit 1	unused
bit 2	0 = load ramdisk
bit 3	0 = prompt for ramdisk

rootdev major/minor number pair of device to mount as the root filesystem.

video_num_cols / video_num_rows These two together describe the character size of the dummy console, or VGA console character size. They should not be used for any other purpose.

It's generally a good idea to set these to be either standard VGA, or the equivalent character size of your fbcon display. This then allows all the bootup messages to be displayed correctly.

video_x / video_y This describes the character position of cursor on VGA console, and is otherwise unused. (should not be used for other console types, and should not be used for other purposes).

memc_control_reg MEMC chip control register for Acorn Archimedes and Acorn A5000 based machines. May be used differently by different architectures.

sounddefault Default sound setting on Acorn machines. May be used differently by different architectures.

adfsdrives Number of ADFS/MFM disks. May be used differently by different architectures.

bytes_per_char_h / bytes_per_char_v These are now obsolete, and should not be used.

pages_in_bank[4] Number of pages in each bank of the systems memory (used for RiscPC). This is intended to be used on systems where the physical memory is non-contiguous from the processors point of view.

pages_in_vram Number of pages in VRAM (used on Acorn RiscPC). This value may also be used by loaders if the size of the video RAM can't be obtained from the hardware.

initrd_start / initrd_size This describes the kernel virtual start address and size of the initial ramdisk.

rd_start Start address in sectors of the ramdisk image on a floppy disk.

system_rev system revision number.

system_serial_low / system_serial_high system 64-bit serial number

mem_fclk_21285 The speed of the external oscillator to the 21285 (footbridge), which control's the speed of the memory bus, timer & serial port. Depending upon the speed of the cpu its value can be between 0-66 MHz. If no params are passed or a value of zero is passed, then a value of 50 Mhz is the default on 21285 architectures.

paths[8][128] These are now obsolete, and should not be used.

commandline Kernel command line parameters. Details can be found elsewhere.

CHAPTER

TWELVE

SOFTWARE EMULATION OF DEPRECATED SWP INSTRUCTION (CONFIG_SWP_EMULATE)

ARMv6 architecture deprecates use of the SWP/SWPB instructions, and recommeds moving to the load-locked/store-conditional instructions LDREX and STREX.

ARMv7 multiprocessing extensions introduce the ability to disable these instructions, triggering an undefined instruction exception when executed. Trapped instructions are emulated using an LDREX/STREX or LDREXB/STREXB sequence. If a memory access fault (an abort) occurs, a segmentation fault is signalled to the triggering process.

/proc/cpu/swp_emulation holds some statistics/information, including the PID of the last process to trigger the emulation to be invocated. For example:

```
Emulated SWP:          12
Emulated SWPB:                 0
Aborted SWP{B}:                1
Last process:          314
```

NOTE: when accessing uncached shared regions, LDREX/STREX rely on an external transaction monitoring block called a global monitor to maintain update atomicity. If your system does not implement a global monitor, this option can cause programs that perform SWP operations to uncached memory to deadlock, as the STREX operation will always fail.

CHAPTER

THIRTEEN

THE UNIFIED EXTENSIBLE FIRMWARE INTERFACE (UEFI)

UEFI, the Unified Extensible Firmware Interface, is a specification governing the behaviours of compatible firmware interfaces. It is maintained by the UEFI Forum - http://www.uefi.org/.

UEFI is an evolution of its predecessor 'EFI', so the terms EFI and UEFI are used somewhat interchangeably in this document and associated source code. As a rule, anything new uses 'UEFI', whereas 'EFI' refers to legacy code or specifications.

13.1 UEFI support in Linux

Booting on a platform with firmware compliant with the UEFI specification makes it possible for the kernel to support additional features:

- UEFI Runtime Services
- Retrieving various configuration information through the standardised interface of UEFI configuration tables. (ACPI, SMBIOS, ⋯)

For actually enabling [U]EFI support, enable:

- CONFIG_EFI=y
- CONFIG_EFI_VARS=y or m

The implementation depends on receiving information about the UEFI environment in a Flattened Device Tree (FDT) - so is only available with CONFIG_OF.

13.2 UEFI stub

The "stub" is a feature that extends the Image/zImage into a valid UEFI PE/COFF executable, including a loader application that makes it possible to load the kernel directly from the UEFI shell, boot menu, or one of the lightweight bootloaders like Gummiboot or rEFInd.

The kernel image built with stub support remains a valid kernel image for booting in non-UEFI environments.

13.3 UEFI kernel support on ARM

UEFI kernel support on the ARM architectures (arm and arm64) is only available when boot is performed through the stub.

When booting in UEFI mode, the stub deletes any memory nodes from a provided DT. Instead, the kernel reads the UEFI memory map.

The stub populates the FDT /chosen node with (and the kernel scans for) the following parameters:

Name	Size	Description
linux,uefi-system-table	64-bit	Physical address of the UEFI System Table.
linux,uefi-mmap-start	64-bit	Physical address of the UEFI memory map, populated by the UEFI GetMemoryMap() call.
linux,uefi-mmap-size	32-bit	Size in bytes of the UEFI memory map pointed to in previous entry.
linux,uefi-mmap-desc-size	32-bit	Size in bytes of each entry in the UEFI memory map.
linux,uefi-mmap-desc-ver	32-bit	Version of the mmap descriptor format.

CHAPTER

FOURTEEN

VLOCKS FOR BARE-METAL MUTUAL EXCLUSION

Voting Locks, or "vlocks" provide a simple low-level mutual exclusion mechanism, with reasonable but minimal requirements on the memory system.

These are intended to be used to coordinate critical activity among CPUs which are otherwise non-coherent, in situations where the hardware provides no other mechanism to support this and ordinary spinlocks cannot be used.

vlocks make use of the atomicity provided by the memory system for writes to a single memory location. To arbitrate, every CPU "votes for itself", by storing a unique number to a common memory location. The final value seen in that memory location when all the votes have been cast identifies the winner.

In order to make sure that the election produces an unambiguous result in finite time, a CPU will only enter the election in the first place if no winner has been chosen and the election does not appear to have started yet.

14.1 Algorithm

The easiest way to explain the vlocks algorithm is with some pseudo-code:

```
int currently_voting[NR_CPUS] = { 0, };
int last_vote = -1; /* no votes yet */

bool vlock_trylock(int this_cpu)
{
        /* signal our desire to vote */
        currently_voting[this_cpu] = 1;
        if (last_vote != -1) {
                /* someone already volunteered himself */
                currently_voting[this_cpu] = 0;
                return false; /* not ourself */
        }

        /* let's suggest ourself */
        last_vote = this_cpu;
        currently_voting[this_cpu] = 0;

        /* then wait until everyone else is done voting */
        for_each_cpu(i) {
                while (currently_voting[i] != 0)
                        /* wait */;
```

(continues on next page)

(continued from previous page)

```
        }

        /* result */
        if (last_vote == this_cpu)
                return true; /* we won */
        return false;
}

bool vlock_unlock(void)
{
        last_vote = -1;
}
```

The currently_voting[] array provides a way for the CPUs to determine whether an election is in progress, and plays a role analogous to the "entering" array in Lamport's bakery algorithm [1].

However, once the election has started, the underlying memory system atomicity is used to pick the winner. This avoids the need for a static priority rule to act as a tie-breaker, or any counters which could overflow.

As long as the last_vote variable is globally visible to all CPUs, it will contain only one value that won't change once every CPU has cleared its currently_voting flag.

14.2 Features and limitations

- vlocks are not intended to be fair. In the contended case, it is the _last_ CPU which attempts to get the lock which will be most likely to win.

  vlocks are therefore best suited to situations where it is necessary to pick a unique winner, but it does not matter which CPU actually wins.

- Like other similar mechanisms, vlocks will not scale well to a large number of CPUs.

  vlocks can be cascaded in a voting hierarchy to permit better scaling if necessary, as in the following hypothetical example for 4096 CPUs:

```
/* first level: local election */
my_town = towns[(this_cpu >> 4) & 0xf];
I_won = vlock_trylock(my_town, this_cpu & 0xf);
if (I_won) {
        /* we won the town election, let's go for the state */
        my_state = states[(this_cpu >> 8) & 0xf];
        I_won = vlock_lock(my_state, this_cpu & 0xf));
        if (I_won) {
                /* and so on */
                I_won = vlock_lock(the_whole_country, this_cpu & 0xf];
                if (I_won) {
                        /* ... */
                }
                vlock_unlock(the_whole_country);
        }
        vlock_unlock(my_state);
```

(continues on next page)

(continued from previous page)

```
}
vlock_unlock(my_town);
```

14.3 ARM implementation

The current ARM implementation [2] contains some optimisations beyond the basic algorithm:

- By packing the members of the currently_voting array close together, we can read the whole array in one transaction (providing the number of CPUs potentially contending the lock is small enough). This reduces the number of round-trips required to external memory.

  In the ARM implementation, this means that we can use a single load and comparison:

  ```
  LDR     Rt, [Rn]
  CMP     Rt, #0
  ```

  …in place of code equivalent to:

  ```
  LDRB    Rt, [Rn]
  CMP     Rt, #0
  LDRBEQ  Rt, [Rn, #1]
  CMPEQ   Rt, #0
  LDRBEQ  Rt, [Rn, #2]
  CMPEQ   Rt, #0
  LDRBEQ  Rt, [Rn, #3]
  CMPEQ   Rt, #0
  ```

  This cuts down on the fast-path latency, as well as potentially reducing bus contention in contended cases.

  The optimisation relies on the fact that the ARM memory system guarantees coherency between overlapping memory accesses of different sizes, similarly to many other architectures. Note that we do not care which element of currently_voting appears in which bits of Rt, so there is no need to worry about endianness in this optimisation.

  If there are too many CPUs to read the currently_voting array in one transaction then multiple transations are still required. The implementation uses a simple loop of word-sized loads for this case. The number of transactions is still fewer than would be required if bytes were loaded individually.

  In principle, we could aggregate further by using LDRD or LDM, but to keep the code simple this was not attempted in the initial implementation.

- vlocks are currently only used to coordinate between CPUs which are unable to enable their caches yet. This means that the implementation removes many of the barriers which would be required when executing the algorithm in cached memory.

  packing of the currently_voting array does not work with cached memory unless all CPUs contending the lock are cache-coherent, due to cache write-

backs from one CPU clobbering values written by other CPUs. (Though if all the CPUs are cache-coherent, you should be probably be using proper spinlocks instead anyway).

- The "no votes yet" value used for the last_vote variable is 0 (not -1 as in the pseudocode). This allows statically-allocated vlocks to be implicitly initialised to an unlocked state simply by putting them in .bss.

  An offset is added to each CPU's ID for the purpose of setting this variable, so that no CPU uses the value 0 for its ID.

14.4 Colophon

Originally created and documented by Dave Martin for Linaro Limited, for use in ARM-based big.LITTLE platforms, with review and input gratefully received from Nicolas Pitre and Achin Gupta. Thanks to Nicolas for grabbing most of this text out of the relevant mail thread and writing up the pseudocode.

Copyright (C) 2012-2013 Linaro Limited Distributed under the terms of Version 2 of the GNU General Public License, as defined in linux/COPYING.

14.5 References

[1] Lamport, L. "A New Solution of Dijkstra's Concurrent Programming Problem", Communications of the ACM 17, 8 (August 1974), 453-455.

https://en.wikipedia.org/wiki/Lamport%27s_bakery_algorithm

[2] linux/arch/arm/common/vlock.S, www.kernel.org.

CHAPTER

FIFTEEN

PORTING

Taken from list archive at http://lists.arm.linux.org.uk/pipermail/linux-arm-kernel/2001-July/004064.html

15.1 Initial definitions

The following symbol definitions rely on you knowing the translation that __virt_to_phys() does for your machine. This macro converts the passed virtual address to a physical address. Normally, it is simply:

phys = virt - PAGE_OFFSET + PHYS_OFFSET

15.2 Decompressor Symbols

ZTEXTADDR Start address of decompressor. There's no point in talking about virtual or physical addresses here, since the MMU will be off at the time when you call the decompressor code. You normally call the kernel at this address to start it booting. This doesn't have to be located in RAM, it can be in flash or other read-only or read-write addressable medium.

ZBSSADDR Start address of zero-initialised work area for the decompressor. This must be pointing at RAM. The decompressor will zero initialise this for you. Again, the MMU will be off.

ZRELADDR This is the address where the decompressed kernel will be written, and eventually executed. The following constraint must be valid:

__virt_to_phys(TEXTADDR) == ZRELADDR

The initial part of the kernel is carefully coded to be position independent.

INITRD_PHYS Physical address to place the initial RAM disk. Only relevant if you are using the bootpImage stuff (which only works on the old struct param_struct).

INITRD_VIRT Virtual address of the initial RAM disk. The following constraint must be valid:

__virt_to_phys(INITRD_VIRT) == INITRD_PHYS

PARAMS_PHYS Physical address of the struct param_struct or tag list, giving the kernel various parameters about its execution environment.

15.3 Kernel Symbols

PHYS_OFFSET Physical start address of the first bank of RAM.

PAGE_OFFSET Virtual start address of the first bank of RAM. During the kernel boot phase, virtual address PAGE_OFFSET will be mapped to physical address PHYS_OFFSET, along with any other mappings you supply. This should be the same value as TASK_SIZE.

TASK_SIZE The maximum size of a user process in bytes. Since user space always starts at zero, this is the maximum address that a user process can access+1. The user space stack grows down from this address.

Any virtual address below TASK_SIZE is deemed to be user process area, and therefore managed dynamically on a process by process basis by the kernel. I' ll call this the user segment.

Anything above TASK_SIZE is common to all processes. I'll call this the kernel segment.

(In other words, you can' t put IO mappings below TASK_SIZE, and hence PAGE_OFFSET).

TEXTADDR Virtual start address of kernel, normally PAGE_OFFSET + 0x8000. This is where the kernel image ends up. With the latest kernels, it must be located at 32768 bytes into a 128MB region. Previous kernels placed a restriction of 256MB here.

DATAADDR Virtual address for the kernel data segment. Must not be defined when using the decompressor.

VMALLOC_START / VMALLOC_END Virtual addresses bounding the vmalloc() area. There must not be any static mappings in this area; vmalloc will overwrite them. The addresses must also be in the kernel segment (see above). Normally, the vmalloc() area starts VMALLOC_OFFSET bytes above the last virtual RAM address (found using variable high_memory).

VMALLOC_OFFSET Offset normally incorporated into VMALLOC_START to provide a hole between virtual RAM and the vmalloc area. We do this to allow out of bounds memory accesses (eg, something writing off the end of the mapped memory map) to be caught. Normally set to 8MB.

15.4 Architecture Specific Macros

BOOT_MEM(pram,pio,vio) pram specifies the physical start address of RAM. Must always be present, and should be the same as PHYS_OFFSET.

pio is the physical address of an 8MB region containing IO for use with the debugging macros in arch/arm/kernel/debug-armv.S.

vio is the virtual address of the 8MB debugging region.

It is expected that the debugging region will be re-initialised by the architecture specific code later in the code (via the MAPIO function).

BOOT_PARAMS Same as, and see PARAMS_PHYS.

FIXUP(func) Machine specific fixups, run before memory subsystems have been initialised.

MAPIO(func) Machine specific function to map IO areas (including the debug region above).

INITIRQ(func) Machine specific function to initialise interrupts.

CHAPTER

SIXTEEN

SOC-SPECIFIC DOCUMENTS

16.1 Release Notes for Linux on Intel's IXP4xx Network Processor

16.1.1 Maintained by Deepak Saxena <dsaxena@plexity.net>

1. Overview

Intel's IXP4xx network processor is a highly integrated SOC that is targeted for network applications, though it has become popular in industrial control and other areas due to low cost and power consumption. The IXP4xx family currently consists of several processors that support different network offload functions such as encryption, routing, firewalling, etc. The IXP46x family is an updated version which supports faster speeds, new memory and flash configurations, and more integration such as an on-chip I2C controller.

For more information on the various versions of the CPU, see:

> http://developer.intel.com/design/network/products/npfamily/ixp4xx.htm

Intel also made the IXCP1100 CPU for sometime which is an IXP4xx stripped of much of the network intelligence.

2. Linux Support

Linux currently supports the following features on the IXP4xx chips:

- Dual serial ports
- PCI interface
- Flash access (MTD/JFFS)
- I2C through GPIO on IXP42x
- GPIO for input/output/interrupts See arch/arm/mach-ixp4xx/include/mach/platform.h for access functions.
- Timers (watchdog, OS)

The following components of the chips are not supported by Linux and require the use of Intel's proprietary CSR software:

- USB device interface
- Network interfaces (HSS, Utopia, NPEs, etc)

- Network offload functionality

If you need to use any of the above, you need to download Intel's software from:

http://developer.intel.com/design/network/products/npfamily/ixp425.htm

DO NOT POST QUESTIONS TO THE LINUX MAILING LISTS REGARDING THE PROPRIETARY SOFTWARE.

There are several websites that provide directions/pointers on using Intel's software:

- http://sourceforge.net/projects/ixp4xx-osdg/ Open Source Developer's Guide for using uClinux and the Intel libraries
- http://gatewaymaker.sourceforge.net/ Simple one page summary of building a gateway using an IXP425 and Linux
- http://ixp425.sourceforge.net/ ATM device driver for IXP425 that relies on Intel's libraries

3. Known Issues/Limitations

3a. Limited inbound PCI window

The IXP4xx family allows for up to 256MB of memory but the PCI interface can only expose 64MB of that memory to the PCI bus. This means that if you are running with > 64MB, all PCI buffers outside of the accessible range will be bounced using the routines in arch/arm/common/dmabounce.c.

3b. Limited outbound PCI window

IXP4xx provides two methods of accessing PCI memory space:

1) A direct mapped window from 0x48000000 to 0x4bffffff (64MB). To access PCI via this space, we simply ioremap() the BAR into the kernel and we can use the standard read[bwl]/write[bwl] macros. This is the preffered method due to speed but it limits the system to just 64MB of PCI memory. This can be problamatic if using video cards and other memory-heavy devices.

2) If > 64MB of memory space is required, the IXP4xx can be configured to use indirect registers to access PCI This allows for up to 128MB (0x48000000 to 0x4fffffff) of memory on the bus. The disadvantage of this is that every PCI access requires three local register accesses plus a spinlock, but in some cases the performance hit is acceptable. In addition, you cannot mmap() PCI devices in this case due to the indirect nature of the PCI window.

By default, the direct method is used for performance reasons. If you need more PCI memory, enable the IXP4XX_INDIRECT_PCI config option.

3c. GPIO as Interrupts

Currently the code only handles level-sensitive GPIO interrupts

4. Supported platforms

ADI Engineering Coyote Gateway Reference Platform http://www.adiengineering.com/productsCoyote.html

The ADI Coyote platform is reference design for those building small residential/office gateways. One NPE is connected to a 10/100 interface, one to 4-port 10/100 switch, and the third to and ADSL interface. In addition, it also supports to POTs interfaces connected via SLICs. Note that those are not supported by Linux ATM. Finally, the platform has two mini-PCI slots used for 802.11[bga] cards. Finally, there is an IDE port hanging off the expansion bus.

Gateworks Avila Network Platform http://www.gateworks.com/support/overview.php

The Avila platform is basically and IXDP425 with the 4 PCI slots replaced with mini-PCI slots and a CF IDE interface hanging off the expansion bus.

Intel IXDP425 Development Platform http://www.intel.com/design/network/products/npfamily/ixdpg425.htm

This is Intel's standard reference platform for the IXDP425 and is also known as the Richfield board. It contains 4 PCI slots, 16MB of flash, two 10/100 ports and one ADSL port.

Intel IXDP465 Development Platform http://www.intel.com/design/network/products/npfamily/ixdp465.htm

This is basically an IXDP425 with an IXP465 and 32M of flash instead of just 16.

Intel IXDPG425 Development Platform

This is basically and ADI Coyote board with a NEC EHCI controller added. One issue with this board is that the mini-PCI slots only have the 3.3v line connected, so you can't use a PCI to mini-PCI adapter with an E100 card. So to NFS root you need to use either the CSR or a WiFi card and a ramdisk that BOOTPs and then does a pivot_root to NFS.

Motorola PrPMC1100 Processor Mezanine Card http://www.fountainsys.com

The PrPMC1100 is based on the IXCP1100 and is meant to plug into and IXP2400/2800 system to act as the system controller. It simply contains a CPU and 16MB of flash on the board and needs to be plugged into a carrier board to function. Currently Linux only supports the Motorola PrPMC carrier board for this platform.

5. TODO LIST

- Add support for Coyote IDE
- Add support for edge-based GPIO interrupts
- Add support for CF IDE on expansion bus

6. Thanks

The IXP4xx work has been funded by Intel Corp. and MontaVista Software, Inc.

The following people have contributed patches/comments/etc:

- Lennerty Buytenhek
- Lutz Jaenicke

- Justin Mayfield
- Robert E. Ranslam

[I know I’ve forgotten others, please email me to be added]

Last Update: 01/04/2005

16.2 ARM Marvell SoCs

This document lists all the ARM Marvell SoCs that are currently supported in mainline by the Linux kernel. As the Marvell families of SoCs are large and complex, it is hard to understand where the support for a particular SoC is available in the Linux kernel. This document tries to help in understanding where those SoCs are supported, and to match them with their corresponding public datasheet, when available.

16.2.1 Orion family

Flavors:

- 88F5082
- 88F5181
- 88F5181L
- 88F5182
  - Datasheet: http://www.embeddedarm.com/documentation/third-party/MV88F5182-datasheet.pdf
  - Programmer’s User Guide: http://www.embeddedarm.com/documentation/third-party/MV88F5182-opensource-manual.pdf
  - User Manual: http://www.embeddedarm.com/documentation/third-party/MV88F5182-usermanual.pdf
- 88F5281
  - Datasheet: http://www.ocmodshop.com/images/reviews/networking/qnap_ts409u/marvel_88f5281_data_sheet.pdf
- 88F6183

Core: Feroceon 88fr331 (88f51xx) or 88fr531-vd (88f52xx) ARMv5 compatible

Linux kernel mach directory: arch/arm/mach-orion5x

Linux kernel plat directory: arch/arm/plat-orion

16.2.2 Kirkwood family

Flavors:

- 88F6282 a.k.a Armada 300
  - Product Brief : http://www.marvell.com/embedded-processors/armada-300/assets/armada_310.pdf
- 88F6283 a.k.a Armada 310
  - Product Brief : http://www.marvell.com/embedded-processors/armada-300/assets/armada_310.pdf
- 88F6190
  - Product Brief : http://www.marvell.com/embedded-processors/kirkwood/assets/88F6190-003_WEB.pdf
  - Hardware Spec : http://www.marvell.com/embedded-processors/kirkwood/assets/HW_88F619x_OpenSource.pdf
  - Functional Spec: http://www.marvell.com/embedded-processors/kirkwood/assets/FS_88F6180_9x_6281_OpenSource.pdf
- 88F6192
  - Product Brief : http://www.marvell.com/embedded-processors/kirkwood/assets/88F6192-003_ver1.pdf
  - Hardware Spec : http://www.marvell.com/embedded-processors/kirkwood/assets/HW_88F619x_OpenSource.pdf
  - Functional Spec: http://www.marvell.com/embedded-processors/kirkwood/assets/FS_88F6180_9x_6281_OpenSource.pdf
- 88F6182
- 88F6180
  - Product Brief : http://www.marvell.com/embedded-processors/kirkwood/assets/88F6180-003_ver1.pdf
  - Hardware Spec : http://www.marvell.com/embedded-processors/kirkwood/assets/HW_88F6180_OpenSource.pdf
  - Functional Spec: http://www.marvell.com/embedded-processors/kirkwood/assets/FS_88F6180_9x_6281_OpenSource.pdf
- 88F6281

- Product Brief : http://www.marvell.com/embedded-processors/kirkwood/assets/88F6281-004_ver1.pdf
- Hardware Spec : http://www.marvell.com/embedded-processors/kirkwood/assets/HW_88F6281_OpenSource.pdf
- Functional Spec: http://www.marvell.com/embedded-processors/kirkwood/assets/FS_88F6180_9x_6281_OpenSource.pdf

Homepage: http://www.marvell.com/embedded-processors/kirkwood/

Core: Feroceon 88fr131 ARMv5 compatible

Linux kernel mach directory: arch/arm/mach-mvebu

Linux kernel plat directory: none

16.2.3 Discovery family

Flavors:

- MV78100
  - Product Brief : http://www.marvell.com/embedded-processors/discovery-innovation/assets/MV78100-003_WEB.pdf
  - Hardware Spec : http://www.marvell.com/embedded-processors/discovery-innovation/assets/HW_MV78100_OpenSource.pdf
  - Functional Spec: http://www.marvell.com/embedded-processors/discovery-innovation/assets/FS_MV76100_78100_78200_OpenSource.pdf
- MV78200
  - Product Brief : http://www.marvell.com/embedded-processors/discovery-innovation/assets/MV78200-002_WEB.pdf
  - Hardware Spec : http://www.marvell.com/embedded-processors/discovery-innovation/assets/HW_MV78200_OpenSource.pdf
  - Functional Spec: http://www.marvell.com/embedded-processors/discovery-innovation/assets/FS_MV76100_78100_78200_OpenSource.pdf
- MV76100

  Not supported by the Linux kernel.

Core: Feroceon 88fr571-vd ARMv5 compatible

Linux kernel mach directory: arch/arm/mach-mv78xx0

Linux kernel plat directory: arch/arm/plat-orion

16.2.4 EBU Armada family

Armada 370 Flavors:

- 88F6710
- 88F6707
- 88F6W11
- Product Brief: http://www.marvell.com/embedded-processors/armada-300/assets/Marvell_ARMADA_370_SoC.pdf
- Hardware Spec: http://www.marvell.com/embedded-processors/armada-300/assets/ARMADA370-datasheet.pdf
- Functional Spec: http://www.marvell.com/embedded-processors/armada-300/assets/ARMADA370-FunctionalSpec-datasheet.pdf

Core: Sheeva ARMv7 compatible PJ4B

Armada 375 Flavors:

- 88F6720
- Product Brief: http://www.marvell.com/embedded-processors/armada-300/assets/ARMADA_375_SoC-01_product_brief.pdf

Core: ARM Cortex-A9

Armada 38x Flavors:

- 88F6810 Armada 380
- 88F6820 Armada 385
- 88F6828 Armada 388
- Product infos: http://www.marvell.com/embedded-processors/armada-38x/
- Functional Spec: https://marvellcorp.wufoo.com/forms/marvell-armada-38x-functional-specifications/

Core: ARM Cortex-A9

Armada 39x Flavors:

- 88F6920 Armada 390
- 88F6928 Armada 398
- Product infos: http://www.marvell.com/embedded-processors/armada-39x/

Core: ARM Cortex-A9

Armada XP Flavors:

- MV78230
- MV78260
- MV78460

NOTE: not to be confused with the non-SMP 78xx0 SoCs

Product Brief: http://www.marvell.com/embedded-processors/armada-xp/assets/Marvell-ArmadaXP-SoC-product%20brief.pdf

Functional Spec: http://www.marvell.com/embedded-processors/armada-xp/assets/ARMADA-XP-Functional-SpecDatasheet.pdf

- Hardware Specs:
  - http://www.marvell.com/embedded-processors/armada-xp/assets/HW_MV78230_OS.PDF
  - http://www.marvell.com/embedded-processors/armada-xp/assets/HW_MV78260_OS.PDF
  - http://www.marvell.com/embedded-processors/armada-xp/assets/HW_MV78460_OS.PDF

Core: Sheeva ARMv7 compatible Dual-core or Quad-core PJ4B-MP

Linux kernel mach directory: arch/arm/mach-mvebu

Linux kernel plat directory: none

16.2.5 EBU Armada family ARMv8

Armada 3710/3720 Flavors:

- 88F3710
- 88F3720

Core: ARM Cortex A53 (ARMv8)

Homepage: http://www.marvell.com/embedded-processors/armada-3700/

Product Brief: http://www.marvell.com/embedded-processors/assets/PB-88F3700-FNL.pdf

Device tree files: arch/arm64/boot/dts/marvell/armada-37*

Armada 7K Flavors:

- 88F7020 (AP806 Dual + one CP110)
- 88F7040 (AP806 Quad + one CP110)

Core: ARM Cortex A72

Homepage: http://www.marvell.com/embedded-processors/armada-70xx/

Product Brief:

- http://www.marvell.com/embedded-processors/assets/Armada7020PB-Jan2016.pdf
- http://www.marvell.com/embedded-processors/assets/Armada7040PB-Jan2016.pdf

Device tree files: arch/arm64/boot/dts/marvell/armada-70*

Armada 8K Flavors:

- 88F8020 (AP806 Dual + two CP110)
- 88F8040 (AP806 Quad + two CP110)

Core: ARM Cortex A72

Homepage: http://www.marvell.com/embedded-processors/armada-80xx/

Product Brief:

- http://www.marvell.com/embedded-processors/assets/Armada8020PB-Jan2016.pdf
- http://www.marvell.com/embedded-processors/assets/Armada8040PB-Jan2016.pdf

Device tree files: arch/arm64/boot/dts/marvell/armada-80*

16.2.6 Avanta family

Flavors:

- 88F6510
- 88F6530P
- 88F6550
- 88F6560

Homepage: http://www.marvell.com/broadband/

Product Brief: http://www.marvell.com/broadband/assets/Marvell_Avanta_88F6510_305_060-001_product_brief.pdf

No public datasheet available.

Core: ARMv5 compatible

Linux kernel mach directory: no code in mainline yet, planned for the future

Linux kernel plat directory: no code in mainline yet, planned for the future

16.2.7 Storage family

Armada SP:

- 88RC1580

Product infos: http://www.marvell.com/storage/armada-sp/

Core: Sheeva ARMv7 comatible Quad-core PJ4C

(not supported in upstream Linux kernel)

16.2.8 Dove family (application processor)

Flavors:

- 88AP510 a.k.a Armada 510

  Product Brief: http://www.marvell.com/application-processors/armada-500/assets/Marvell_Armada510_SoC.pdf

  Hardware Spec: http://www.marvell.com/application-processors/armada-500/assets/Armada-510-Hardware-Spec.pdf

Functional Spec: http://www.marvell.com/application-processors/armada-500/assets/Armada-510-Functional-Spec.pdf

Homepage: http://www.marvell.com/application-processors/armada-500/

Core: ARMv7 compatible

Directory:

- arch/arm/mach-mvebu (DT enabled platforms)
- arch/arm/mach-dove (non-DT enabled platforms)

16.2.9 PXA 2xx/3xx/93x/95x family

Flavors:

- **PXA21x, PXA25x, PXA26x**
  - Application processor only
  - Core: ARMv5 XScale1 core
- **PXA270, PXA271, PXA272**
  - Product Brief : http://www.marvell.com/application-processors/pxa-family/assets/pxa_27x_pb.pdf
  - Design guide : http://www.marvell.com/application-processors/pxa-family/assets/pxa_27x_design_guide.pdf
  - Developers manual : http://www.marvell.com/application-processors/pxa-family/assets/pxa_27x_dev_man.pdf

  - Specification : http://www.marvell.com/application-processors/pxa-family/assets/pxa_27x_emts.pdf
  - Specification update : http://www.marvell.com/application-processors/pxa-family/assets/pxa_27x_spec_update.pdf
  - Application processor only
  - Core: ARMv5 XScale2 core

- **PXA300, PXA310, PXA320**
  - PXA 300 Product Brief : http://www.marvell.com/application-processors/pxa-family/assets/PXA300_PB_R4.pdf
  - PXA 310 Product Brief : http://www.marvell.com/application-processors/pxa-family/assets/PXA310_PB_R4.pdf
  - PXA 320 Product Brief : http://www.marvell.com/application-processors/pxa-family/assets/PXA320_PB_R4.pdf
  - Design guide : http://www.marvell.com/application-processors/pxa-family/assets/PXA3xx_Design_Guide.pdf
  - Developers manual : http://www.marvell.com/application-processors/pxa-family/assets/PXA3xx_Developers_Manual.zip
  - Specifications : http://www.marvell.com/application-processors/pxa-family/assets/PXA3xx_EMTS.pdf
  - Specification Update : http://www.marvell.com/application-processors/pxa-family/assets/PXA3xx_Spec_Update.zip
  - Reference Manual : http://www.marvell.com/application-processors/pxa-family/assets/PXA3xx_TavorP_BootROM_Ref_Manual.pdf
  - Application processor only
  - Core: ARMv5 XScale3 core

- **PXA930, PXA935**
  - Application processor with Communication processor
  - Core: ARMv5 XScale3 core

- **PXA955**
  - Application processor with Communication processor
  - Core: ARMv7 compatible Sheeva PJ4 core

Comments:

- This line of SoCs originates from the XScale family developed by Intel and acquired by Marvell in ~2006. The PXA21x, PXA25x, PXA26x, PXA27x, PXA3xx and PXA93x were developed by Intel, while the later PXA95x were developed by Marvell.
- Due to their XScale origin, these SoCs have virtually nothing in common with the other (Kirkwood, Dove, etc.) families of Marvell SoCs, except with the MMP/MMP2 family of SoCs.

Linux kernel mach directory: arch/arm/mach-pxa

Linux kernel plat directory: arch/arm/plat-pxa

16.2.10 MMP/MMP2/MMP3 family (communication processor)

Flavors:

- **PXA168, a.k.a Armada 168**
  - Homepage : http://www.marvell.com/application-processors/armada-100/armada-168.jsp
  - Product brief : http://www.marvell.com/application-processors/armada-100/assets/pxa_168_pb.pdf
  - Hardware manual : http://www.marvell.com/application-processors/armada-100/assets/armada_16x_datasheet.pdf
  - Software manual : http://www.marvell.com/application-processors/armada-100/assets/armada_16x_software_manual.pdf
  - Specification update : http://www.marvell.com/application-processors/armada-100/assets/ARMADA16x_Spec_update.pdf
  - Boot ROM manual : http://www.marvell.com/application-processors/armada-100/assets/armada_16x_ref_manual.pdf
  - App node package : http://www.marvell.com/application-processors/armada-100/assets/armada_16x_app_note_package.pdf
  - Application processor only
  - Core: ARMv5 compatible Marvell PJ1 88sv331 (Mohawk)
- **PXA910/PXA920**
  - Homepage : http://www.marvell.com/communication-processors/pxa910/

  - Product Brief : http://www.marvell.com/communication-processors/pxa910/assets/Marvell_PXA910_Platform-001_PB_final.pdf
  - Application processor with Communication processor
  - Core: ARMv5 compatible Marvell PJ1 88sv331 (Mohawk)

- **PXA688, a.k.a. MMP2, a.k.a Armada 610**
  - Product Brief : http://www.marvell.com/application-processors/armada-600/assets/armada610_pb.pdf
  - Application processor only
  - Core: ARMv7 compatible Sheeva PJ4 88sv581x core

- **PXA2128, a.k.a. MMP3 (OLPC XO4, Linux support not upstream)**
  - Product Brief : http://www.marvell.com/application-processors/armada/pxa2128/assets/Marvell-ARMADA-PXA2128-SoC-PB.pdf
  - Application processor only
  - Core: Dual-core ARMv7 compatible Sheeva PJ4C core

- **PXA960/PXA968/PXA978 (Linux support not upstream)**
  - Application processor with Communication Processor
  - Core: ARMv7 compatible Sheeva PJ4 core

- **PXA986/PXA988 (Linux support not upstream)**
  - Application processor with Communication Processor
  - Core: Dual-core ARMv7 compatible Sheeva PJ4B-MP core

- **PXA1088/PXA1920 (Linux support not upstream)**
  - Application processor with Communication Processor
  - Core: quad-core ARMv7 Cortex-A7

- **PXA1908/PXA1928/PXA1936**
  - Application processor with Communication Processor
  - Core: multi-core ARMv8 Cortex-A53

Comments:

- This line of SoCs originates from the XScale family developed by Intel and acquired by Marvell in ~2006. All the processors of this MMP/MMP2 family were developed by Marvell.
- Due to their XScale origin, these SoCs have virtually nothing in common with the other (Kirkwood, Dove, etc.) families of Marvell SoCs, except with the PXA family of SoCs listed above.

Linux kernel mach directory: arch/arm/mach-mmp

Linux kernel plat directory: arch/arm/plat-pxa

16.2.11 Berlin family (Multimedia Solutions)

- **Flavors:**
  - **88DE3010, Armada 1000 (no Linux support)**
    - Core: Marvell PJ1 (ARMv5TE), Dual-core
    - Product Brief: http://www.marvell.com.cn/digital-entertainment/assets/armada_1000_pb.pdf
  - **88DE3005, Armada 1500 Mini**
    - Design name: BG2CD
    - Core: ARM Cortex-A9, PL310 L2CC
  - **88DE3006, Armada 1500 Mini Plus**
    - Design name: BG2CDP
    - Core: Dual Core ARM Cortex-A7
  - **88DE3100, Armada 1500**
    - Design name: BG2
    - Core: Marvell PJ4B-MP (ARMv7), Tauros3 L2CC
  - **88DE3114, Armada 1500 Pro**
    - Design name: BG2Q
    - Core: Quad Core ARM Cortex-A9, PL310 L2CC
  - **88DE3214, Armada 1500 Pro 4K**
    - Design name: BG3
    - Core: ARM Cortex-A15, CA15 integrated L2CC
  - **88DE3218, ARMADA 1500 Ultra**
    - Core: ARM Cortex-A53

Homepage: https://www.synaptics.com/products/multimedia-solutions
Directory: arch/arm/mach-berlin

Comments:

- This line of SoCs is based on Marvell Sheeva or ARM Cortex CPUs with Synopsys DesignWare (IRQ, GPIO, Timers, ⋯) and PXA IP (SDHCI, USB, ETH, ⋯).
- The Berlin family was acquired by Synaptics from Marvell in 2017.

16.2.12 CPU Cores

The XScale cores were designed by Intel, and shipped by Marvell in the older PXA processors. Feroceon is a Marvell designed core that developed in-house, and that evolved into Sheeva. The XScale and Feroceon cores were phased out over time and replaced with Sheeva cores in later products, which subsequently got replaced with licensed ARM Cortex-A cores.

XScale 1 CPUID 0x69052xxx ARMv5, iWMMXt

XScale 2 CPUID 0x69054xxx ARMv5, iWMMXt

XScale 3 CPUID 0x69056xxx or 0x69056xxx ARMv5, iWMMXt

Feroceon-1850 88fr331 "Mohawk" CPUID 0x5615331x or 0x41xx926x ARMv5TE, single issue

Feroceon-2850 88fr531-vd "Jolteon" CPUID 0x5605531x or 0x41xx926x ARMv5TE, VFP, dual-issue

Feroceon 88fr571-vd "Jolteon" CPUID 0x5615571x ARMv5TE, VFP, dual-issue

Feroceon 88fr131 "Mohawk-D" CPUID 0x5625131x ARMv5TE, single-issue in-order

Sheeva PJ1 88sv331 "Mohawk" CPUID 0x561584xx ARMv5, single-issue iWMMXt v2

Sheeva PJ4 88sv581x "Flareon" CPUID 0x560f581x ARMv7, idivt, optional iWMMXt v2

Sheeva PJ4B 88sv581x CPUID 0x561f581x ARMv7, idivt, optional iWMMXt v2

Sheeva PJ4B-MP / PJ4C CPUID 0x562f584x ARMv7, idivt/idiva, LPAE, optional iWMMXt v2 and/or NEON

16.2.13 Long-term plans

- Unify the mach-dove/, mach-mv78xx0/, mach-orion5x/ into the mach-mvebu/ to support all SoCs from the Marvell EBU (Engineering Business Unit) in a single mach-<foo> directory. The plat-orion/ would therefore disappear.
- Unify the mach-mmp/ and mach-pxa/ into the same mach-pxa directory. The plat-pxa/ would therefore disappear.

16.2.14 Credits

- Maen Suleiman <maen@marvell.com>
- Lior Amsalem <alior@marvell.com>
- Thomas Petazzoni <thomas.petazzoni@free-electrons.com>
- Andrew Lunn <andrew@lunn.ch>
- Nicolas Pitre <nico@fluxnic.net>
- Eric Miao <eric.y.miao@gmail.com>

16.3 ARM Microchip SoCs (aka AT91)

16.3.1 Introduction

This document gives useful information about the ARM Microchip SoCs that are currently supported in Linux Mainline (you know, the one on kernel.org).

It is important to note that the Microchip (previously Atmel) ARM-based MPU product line is historically named "AT91" or "at91" throughout the Linux kernel development process even if this product prefix has completely disappeared from the official Microchip product name. Anyway, files, directories, git trees, git branches/tags and email subject always contain this "at91" sub-string.

16.3.2 AT91 SoCs

Documentation and detailed datasheet for each product are available on the Microchip website: http://www.microchip.com.

Flavors:

- ARM 920 based SoC - at91rm9200
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-1768-32-bit-ARM920T-Embedded-Microprocessor-AT91RM9200_Datasheet.pdf
- ARM 926 based SoCs - at91sam9260
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-6221-32-bit-ARM926EJ-S-Embedded-Microprocessor-SAM9260_Datasheet.pdf
  - at91sam9xe
    - Datasheet

      http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-6254-32-bit-ARM926EJ-S-Embedded-Microprocessor-SAM9XE_Datasheet.pdf

- at91sam9261
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-6062-ARM926EJ-S-Microprocessor-SAM9261_Datasheet.pdf
- at91sam9263
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-6249-32-bit-ARM926EJ-S-Embedded-Microprocessor-SAM9263_Datasheet.pdf
- at91sam9rl
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/doc6289.pdf
- at91sam9g20
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/DS60001516A.pdf
- at91sam9g45 family - at91sam9g45 - at91sam9g46 - at91sam9m10 - at91sam9m11 (device superset)
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-6437-32-bit-ARM926-Embedded-Microprocessor-SAM9M11_Datasheet.pdf
- at91sam9x5 family (aka “The 5 series”) - at91sam9g15 - at91sam9g25 - at91sam9g35 - at91sam9x25 - at91sam9x35
  - Datasheet (can be considered as covering the whole family)

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11055-32-bit-ARM926EJ-S-Microcontroller-SAM9X35_Datasheet.pdf
- at91sam9n12
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/DS60001517A.pdf
- sam9x60
  - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/SAM9X60-Data-Sheet-DS60001579A.pdf

- ARM Cortex-A5 based SoCs - sama5d3 family
  - sama5d31
  - sama5d33
  - sama5d34
  - sama5d35
  - sama5d36 (device superset)
    - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11121-32-bit-Cortex-A5-Microcontroller-SAMA5D3_Datasheet_B.pdf
- ARM Cortex-A5 + NEON based SoCs - sama5d4 family
  - sama5d41
  - sama5d42
  - sama5d43
  - sama5d44 (device superset)
    - Datasheet

    http://ww1.microchip.com/downloads/en/DeviceDoc/60001525A.pdf
  - sama5d2 family
    - sama5d21
    - sama5d22
    - sama5d23
    - sama5d24
    - sama5d26
    - sama5d27 (device superset)
    - sama5d28 (device superset + environmental monitors)
      - Datasheet

      http://ww1.microchip.com/downloads/en/DeviceDoc/DS60001476B.pdf
- ARM Cortex-M7 MCUs - sams70 family
  - sams70j19
  - sams70j20
  - sams70j21
  - sams70n19
  - sams70n20
  - sams70n21

- sams70q19
- sams70q20
- sams70q21
- samv70 family
  - samv70j19
  - samv70j20
  - samv70n19
  - samv70n20
  - samv70q19
  - samv70q20
- samv71 family
  - samv71j19
  - samv71j20
  - samv71j21
  - samv71n19
  - samv71n20
  - samv71n21
  - samv71q19
  - samv71q20
  - samv71q21
    - Datasheet

      http://ww1.microchip.com/downloads/en/DeviceDoc/SAM-E70-S70-V70-V71-Family-Data-Sheet-DS60001527D.pdf

16.3.3 Linux kernel information

Linux kernel mach directory: arch/arm/mach-at91 MAINTAINERS entry is: "ARM/Microchip (AT91) SoC support"

16.3.4 Device Tree for AT91 SoCs and boards

All AT91 SoCs are converted to Device Tree. Since Linux 3.19, these products must use this method to boot the Linux kernel.

Work In Progress statement: Device Tree files and Device Tree bindings that apply to AT91 SoCs and boards are considered as "Unstable". To be completely clear, any at91 binding can change at any time. So, be sure to use a Device Tree Binary and a Kernel Image generated from the same source tree. Please refer

to the Documentation/devicetree/bindings/ABI.rst file for a definition of a "Stable" binding/ABI. This statement will be removed by AT91 MAINTAINERS when appropriate.

Naming conventions and best practice:

- SoCs Device Tree Source Include files are named after the official name of the product (at91sam9g20.dtsi or sama5d33.dtsi for instance).
- Device Tree Source Include files (.dtsi) are used to collect common nodes that can be shared across SoCs or boards (sama5d3.dtsi or at91sam9x5cm.dtsi for instance). When collecting nodes for a particular peripheral or topic, the identifier have to be placed at the end of the file name, separated with a "_" (at91sam9x5_can.dtsi or sama5d3_gmac.dtsi for example).
- board Device Tree Source files (.dts) are prefixed by the string "at91-" so that they can be identified easily. Note that some files are historical exceptions to this rule (sama5d3[13456]ek.dts, usb_a9g20.dts or animeo_ip.dts for example).

16.4 NetWinder specific documentation

The NetWinder is a small low-power computer, primarily designed to run Linux. It is based around the StrongARM RISC processor, DC21285 PCI bridge, with PC-type hardware glued around it.

16.4.1 Port usage

Min	Max	Description
0x0000	0x000f	DMA1
0x0020	0x0021	PIC1
0x0060	0x006f	Keyboard
0x0070	0x007f	RTC
0x0080	0x0087	DMA1
0x0088	0x008f	DMA2
0x00a0	0x00a3	PIC2
0x00c0	0x00df	DMA2
0x0180	0x0187	IRDA
0x01f0	0x01f6	ide0
0x0201		Game port
0x0203		RWA010 configuration read
0x0220	?	SoundBlaster
0x0250	?	WaveArtist
0x0279		RWA010 configuration index
0x02f8	0x02ff	Serial ttyS1
0x0300	0x031f	Ether10
0x0338		GPIO1
0x033a		GPIO2
0x0370	0x0371	W83977F configuration registers
0x0388	?	AdLib
0x03c0	0x03df	VGA
0x03f6		ide0
0x03f8	0x03ff	Serial ttyS0
0x0400	0x0408	DC21143
0x0480	0x0487	DMA1
0x0488	0x048f	DMA2
0x0a79		RWA010 configuration write
0xe800	0xe80f	ide0/ide1 BM DMA

16.4.2 Interrupt usage

IRQ	type	Description
0	ISA	100Hz timer
1	ISA	Keyboard
2	ISA	cascade
3	ISA	Serial ttyS1
4	ISA	Serial ttyS0
5	ISA	PS/2 mouse
6	ISA	IRDA
7	ISA	Printer
8	ISA	RTC alarm
9	ISA	
10	ISA	GP10 (Orange reset button)
11	ISA	
12	ISA	WaveArtist
13	ISA	
14	ISA	hda1
15	ISA	

16.4.3 DMA usage

DMA	type	Description
0	ISA	IRDA
1	ISA	
2	ISA	cascade
3	ISA	WaveArtist
4	ISA	
5	ISA	
6	ISA	
7	ISA	WaveArtist

16.5 NetWinder’ s floating point emulator

16.5.1 Introduction

This directory contains the version 0.92 test release of the NetWinder Floating Point Emulator.

The majority of the code was written by me, Scott Bambrough It is written in C, with a small number of routines in inline assembler where required. It was written quickly, with a goal of implementing a working version of all the floating point instructions the compiler emits as the first target. I have attempted to be as optimal as possible, but there remains much room for improvement.

I have attempted to make the emulator as portable as possible. One of the problems is with leading underscores on kernel symbols. Elf kernels have no

leading underscores, a.out compiled kernels do. I have attempted to use the C_SYMBOL_NAME macro wherever this may be important.

Another choice I made was in the file structure. I have attempted to contain all operating system specific code in one module (fpmodule.*). All the other files contain emulator specific code. This should allow others to port the emulator to NetBSD for instance relatively easily.

The floating point operations are based on SoftFloat Release 2, by John Hauser. SoftFloat is a software implementation of floating-point that conforms to the IEC/IEEE Standard for Binary Floating-point Arithmetic. As many as four formats are supported: single precision, double precision, extended double precision, and quadruple precision. All operations required by the standard are implemented, except for conversions to and from decimal. We use only the single precision, double precision and extended double precision formats. The port of SoftFloat to the ARM was done by Phil Blundell, based on an earlier port of SoftFloat version 1 by Neil Carson for NetBSD/arm32.

The file README.FPE contains a description of what has been implemented so far in the emulator. The file TODO contains a information on what remains to be done, and other ideas for the emulator.

Bug reports, comments, suggestions should be directed to me at <scottb@netwinder.org>. General reports of "this program doesn't work correctly when your emulator is installed" are useful for determining that bugs still exist; but are virtually useless when attempting to isolate the problem. Please report them, but don't expect quick action. Bugs still exist. The problem remains in isolating which instruction contains the bug. Small programs illustrating a specific problem are a godsend.

Legal Notices

The NetWinder Floating Point Emulator is free software. Everything Rebel.com has written is provided under the GNU GPL. See the file COPYING for copying conditions. Excluded from the above is the SoftFloat code. John Hauser's legal notice for SoftFloat is included below.

SoftFloat Legal Notice

SoftFloat was written by John R. Hauser. This work was made possible in part by the International Computer Science Institute, located at Suite 600, 1947 Center Street, Berkeley, California 94704. Funding was partially provided by the National Science Foundation under grant MIP-9311980. The original version of this code was written as part of a project to build a fixed-point vector processor in collaboration with the University of California at Berkeley, overseen by Profs. Nelson Morgan and John Wawrzynek.

THIS SOFTWARE IS DISTRIBUTED AS IS, FOR FREE. Although reasonable effort has been made to avoid it, THIS SOFTWARE MAY CONTAIN FAULTS THAT WILL AT TIMES RESULT IN INCORRECT BEHAVIOR. USE OF THIS SOFTWARE IS RESTRICTED TO PERSONS AND ORGANIZATIONS WHO CAN AND WILL TAKE FULL RESPONSIBILITY FOR ANY AND ALL LOSSES, COSTS, OR OTHER PROBLEMS ARISING FROM ITS USE. ————————————————————————————-

16.5.2 Current State

The following describes the current state of the NetWinder's floating point emulator.

In the following nomenclature is used to describe the floating point instructions. It follows the conventions in the ARM manual.

```
<S|D|E> = <single|double|extended>, no default
{P|M|Z} = {round to +infinity,round to -infinity,round to zero},
          default = round to nearest
```

Note: items enclosed in {} are optional.

Floating Point Coprocessor Data Transfer Instructions (CPDT)

LDF/STF - load and store floating

<LDF|STF>{cond}<S|D|E> Fd, Rn <LDF|STF>{cond}<S|D|E> Fd, [Rn, #<expression>]{!} <LDF|STF>{cond}<S|D|E> Fd, [Rn], #<expression>

These instructions are fully implemented.

LFM/SFM - load and store multiple floating

Form 1 syntax: <LFM|SFM>{cond}<S|D|E> Fd, <count>, [Rn] <LFM|SFM>{cond}<S|D|E> Fd, <count>, [Rn, #<expression>]{!} <LFM|SFM>{cond}<S|D|E> Fd, <count>, [Rn], #<expression>

Form 2 syntax: <LFM|SFM>{cond}<FD,EA> Fd, <count>, [Rn]{!}

These instructions are fully implemented. They store/load three words for each floating point register into the memory location given in the instruction. The format in memory is unlikely to be compatible with other implementations, in particular the actual hardware. Specific mention of this is made in the ARM manuals.

Floating Point Coprocessor Register Transfer Instructions (CPRT)

Conversions, read/write status/control register instructions

FLT{cond}<S,D,E>{P,M,Z} Fn, Rd Convert integer to floating point FIX{cond}{P,M,Z} Rd, Fn Convert floating point to integer WFS{cond} Rd Write floating point status register RFS{cond} Rd Read floating point status register WFC{cond} Rd Write floating point control register RFC{cond} Rd Read floating point control register

FLT/FIX are fully implemented.

RFS/WFS are fully implemented.

RFC/WFC are fully implemented. RFC/WFC are supervisor only instructions, and presently check the CPU mode, and do an invalid instruction trap if not called from supervisor mode.

Compare instructions

CMF{cond} Fn, Fm Compare floating CMFE{cond} Fn, Fm Compare floating with exception CNF{cond} Fn, Fm Compare negated floating CNFE{cond} Fn, Fm Compare negated floating with exception

These are fully implemented.

Floating Point Coprocessor Data Instructions (CPDT)

Dyadic operations:

ADF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - add SUF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - subtract RSF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse subtract MUF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - multiply DVF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - divide RDV{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse divide

These are fully implemented.

FML{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - fast multiply FDV{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - fast divide FRD{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - fast reverse divide

These are fully implemented as well. They use the same algorithm as the non-fast versions. Hence, in this implementation their performance is equivalent to the MUF/DVF/RDV instructions. This is acceptable according to the ARM manual. The manual notes these are defined only for single operands, on the actual FPA11 hardware they do not work for double or extended precision operands. The emulator currently does not check the requested permissions conditions, and performs the requested operation.

RMF{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - IEEE remainder

This is fully implemented.

Monadic operations:

MVF{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - move MNF{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - move negated

These are fully implemented.

ABS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - absolute value SQT{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - square root RND{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - round

These are fully implemented.

URD{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - unnormalized round NRM{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - normalize

These are implemented. URD is implemented using the same code as the RND instruction. Since URD cannot return a unnormalized number, NRM becomes a NOP.

Library calls:

POW{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - power RPW{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse power POL{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - polar angle (arctan2)

LOG{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - logarithm to base 10 LGN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - logarithm to base e EXP{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - exponent SIN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - sine COS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - cosine TAN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - tangent ASN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arc-sine ACS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arccosine ATN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arctangent

These are not implemented. They are not currently issued by the compiler, and are handled by routines in libc. These are not implemented by the FPA11 hardware, but are handled by the floating point support code. They should be implemented in future versions.

Signalling:

Signals are implemented. However current ELF kernels produced by Rebel.com have a bug in them that prevents the module from generating a SIGFPE. This is caused by a failure to alias fp_current to the kernel variable current_set[0] correctly.

The kernel provided with this distribution (vmlinux-nwfpe-0.93) contains a fix for this problem and also incorporates the current version of the emulator directly. It is possible to run with no floating point module loaded with this kernel. It is provided as a demonstration of the technology and for those who want to do floating point work that depends on signals. It is not strictly necessary to use the module.

A module (either the one provided by Russell King, or the one in this distribution) can be loaded to replace the functionality of the emulator built into the kernel.

16.5.3 Notes

There seems to be a problem with exp(double) and our emulator. I haven't been able to track it down yet. This does not occur with the emulator supplied by Russell King.

I also found one oddity in the emulator. I don't think it is serious but will point it out. The ARM calling conventions require floating point registers f4-f7 to be preserved over a function call. The compiler quite often uses an stfe instruction to save f4 on the stack upon entry to a function, and an ldfe instruction to restore it before returning.

I was looking at some code, that calculated a double result, stored it in f4 then made a function call. Upon return from the function call the number in f4 had been converted to an extended value in the emulator.

This is a side effect of the stfe instruction. The double in f4 had to be converted to extended, then stored. If an lfm/sfm combination had been used, then no conversion would occur. This has performance considerations. The result from the function call and f4 were used in a multiplication. If the emulator sees a multiply of

a double and extended, it promotes the double to extended, then does the multiply in extended precision.

This code will cause this problem:

double x, y, z; z = log(x)/log(y);

The result of log(x) (a double) will be calculated, returned in f0, then moved to f4 to preserve it over the log(y) call. The division will be done in extended precision, due to the stfe instruction used to save f4 in log(y).

16.5.4 TODO LIST

```
POW{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - power
RPW{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - reverse power
POL{cond}<S|D|E>{P,M,Z} Fd, Fn, <Fm,#value> - polar angle (arctan2)

LOG{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - logarithm to base 10
LGN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - logarithm to base e
EXP{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - exponent
SIN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - sine
COS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - cosine
TAN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - tangent
ASN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arcsine
ACS{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arccosine
ATN{cond}<S|D|E>{P,M,Z} Fd, <Fm,#value> - arctangent
```

These are not implemented. They are not currently issued by the compiler, and are handled by routines in libc. These are not implemented by the FPA11 hardware, but are handled by the floating point support code. They should be implemented in future versions.

There are a couple of ways to approach the implementation of these. One method would be to use accurate table methods for these routines. I have a couple of papers by S. Gal from IBM's research labs in Haifa, Israel that seem to promise extreme accuracy (in the order of 99.8%) and reasonable speed. These methods are used in GLIBC for some of the transcendental functions.

Another approach, which I know little about is CORDIC. This stands for Coordinate Rotation Digital Computer, and is a method of computing transcendental functions using mostly shifts and adds and a few multiplications and divisions. The ARM excels at shifts and adds, so such a method could be promising, but requires more research to determine if it is feasible.

Rounding Methods

The IEEE standard defines 4 rounding modes. Round to nearest is the default, but rounding to + or - infinity or round to zero are also allowed. Many architectures allow the rounding mode to be specified by modifying bits in a control register. Not so with the ARM FPA11 architecture. To change the rounding mode one must specify it with each instruction.

This has made porting some benchmarks difficult. It is possible to introduce such a capability into the emulator. The FPCR contains bits describing the rounding

mode. The emulator could be altered to examine a flag, which if set forced it to ignore the rounding mode in the instruction, and use the mode specified in the bits in the FPCR.

This would require a method of getting/setting the flag, and the bits in the FPCR. This requires a kernel call in ArmLinux, as WFC/RFC are supervisor only instructions. If anyone has any ideas or comments I would like to hear them.

NOTE: pulled out from some docs on ARM floating point, specifically for the Acorn FPE, but not limited to it:

The floating point control register (FPCR) may only be present in some implementations: it is there to control the hardware in an implementation- specific manner, for example to disable the floating point system. The user mode of the ARM is not permitted to use this register (since the right is reserved to alter it between implementations) and the WFC and RFC instructions will trap if tried in user mode.

Hence, the answer is yes, you could do this, but then you will run a high risk of becoming isolated if and when hardware FP emulation comes out

– Russell.

16.6 TI Keystone Linux Overview

16.6.1 Introduction

Keystone range of SoCs are based on ARM Cortex-A15 MPCore Processors and c66x DSP cores. This document describes essential information required for users to run Linux on Keystone based EVMs from Texas Instruments.

Following SoCs & EVMs are currently supported:-

K2HK SoC and EVM

a.k.a Keystone 2 Hawking/Kepler SoC TCI6636K2H & TCI6636K2K: See documentation at

http://www.ti.com/product/tci6638k2k http://www.ti.com/product/tci6638k2h

EVM: http://www.advantech.com/Support/TI-EVM/EVMK2HX_sd.aspx

K2E SoC and EVM

a.k.a Keystone 2 Edison SoC

K2E - 66AK2E05:

See documentation at

http://www.ti.com/product/66AK2E05/technicaldocuments

EVM: https://www.einfochips.com/index.php/partnerships/texas-instruments/k2e-evm.html

K2L SoC and EVM

a.k.a Keystone 2 Lamarr SoC

K2L - TCI6630K2L:

See documentation at http://www.ti.com/product/TCI6630K2L/technicaldocuments

EVM: https://www.einfochips.com/index.php/partnerships/texas-instruments/k2l-evm.html

16.6.2 Configuration

All of the K2 SoCs/EVMs share a common defconfig, keystone_defconfig and same image is used to boot on individual EVMs. The platform configuration is specified through DTS. Following are the DTS used:

K2HK EVM: k2hk-evm.dts

K2E EVM: k2e-evm.dts

K2L EVM: k2l-evm.dts

The device tree documentation for the keystone machines are located at

Documentation/devicetree/bindings/arm/keystone/keystone.txt

16.6.3 Document Author

Murali Karicheri <m-karicheri2@ti.com>

Copyright 2015 Texas Instruments

16.7 Texas Instruments Keystone Navigator Queue Management SubSystem driver

Driver source code path drivers/soc/ti/knav_qmss.c drivers/soc/ti/knav_qmss_acc.c

The QMSS (Queue Manager Sub System) found on Keystone SOCs is one of the main hardware sub system which forms the backbone of the Keystone multi-core Navigator. QMSS consist of queue managers, packed-data structure processors(PDSP), linking RAM, descriptor pools and infrastructure Packet DMA. The Queue Manager is a hardware module that is responsible for accelerating management of the packet queues. Packets are queued/de-queued by writing or reading descriptor address to a particular memory mapped location. The PDSPs perform QMSS related functions like accumulation, QoS, or event management. Linking RAM registers are used to link the descriptors which are stored in descriptor RAM. Descriptor RAM is configurable as internal or external memory. The QMSS driver

manages the PDSP setups, linking RAM regions, queue pool management (allocation, push, pop and notify) and descriptor pool management.

knav qmss driver provides a set of APIs to drivers to open/close qmss queues, allocate descriptor pools, map the descriptors, push/pop to queues etc. For details of the available APIs, please refers to include/linux/soc/ti/knav_qmss.h

DT documentation is available at Documentation/devicetree/bindings/soc/ti/keystone-navigator-qmss.txt

16.7.1 Accumulator QMSS queues using PDSP firmware

The QMSS PDSP firmware support accumulator channel that can monitor a single queue or multiple contiguous queues. drivers/soc/ti/knav_qmss_acc.c is the driver that interface with the accumulator PDSP. This configures accumulator channels defined in DTS (example in DT documentation) to monitor 1 or 32 queues per channel. More description on the firmware is available in CPPI/QMSS Low Level Driver document (docs/CPPI_QMSS_LLD_SDS.pdf) at

> git://git.ti.com/keystone-rtos/qmss-lld.git

k2_qmss_pdsp_acc48_k2_le_1_0_0_9.bin firmware supports upto 48 accumulator channels. This firmware is available under ti-keystone folder of firmware.git at

> git://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git

To use copy the firmware image to lib/firmware folder of the initramfs or ubifs file system and provide a sym link to k2_qmss_pdsp_acc48_k2_le_1_0_0_9.bin in the file system and boot up the kernel. User would see

> “firmware file ks2_qmss_pdsp_acc48.bin downloaded for PDSP”

in the boot up log if loading of firmware to PDSP is successful.

Use of accumulated queues requires the firmware image to be present in the file system. The driver doesn’ t acc queues to the supported queue range if PDSP is not running in the SoC. The API call fails if there is a queue open request to an acc queue and PDSP is not running. So make sure to copy firmware to file system before using these queue types.

16.8 TI OMAP

16.8.1 OMAP history

This file contains documentation for running mainline kernel on omaps.

KERNEL	NEW DEPENDENCIES
v4.3+	Update is needed for custom .config files to make sure CONFIG_REGULATOR_PBIAS is enabled for MMC1 to work properly.
v4.18+	Update is needed for custom .config files to make sure CONFIG_MMC_SDHCI_OMAP is enabled for all MMC instances to work in DRA7 and K2G based boards.

16.8.2 The OMAP PM interface

This document describes the temporary OMAP PM interface. Driver authors use these functions to communicate minimum latency or throughput constraints to the kernel power management code. Over time, the intention is to merge features from the OMAP PM interface into the Linux PM QoS code.

Drivers need to express PM parameters which:

- support the range of power management parameters present in the TI SRF;
- separate the drivers from the underlying PM parameter implementation, whether it is the TI SRF or Linux PM QoS or Linux latency framework or something else;
- specify PM parameters in terms of fundamental units, such as latency and throughput, rather than units which are specific to OMAP or to particular OMAP variants;
- allow drivers which are shared with other architectures (e.g., DaVinci) to add these constraints in a way which won't affect non-OMAP systems,
- can be implemented immediately with minimal disruption of other architectures.

This document proposes the OMAP PM interface, including the following five power management functions for driver code:

1. Set the maximum MPU wakeup latency:

```
(*pdata->set_max_mpu_wakeup_lat)(struct device *dev, unsigned long t)
```

2. Set the maximum device wakeup latency:

```
(*pdata->set_max_dev_wakeup_lat)(struct device *dev, unsigned long t)
```

3. Set the maximum system DMA transfer start latency (CORE pwrdm):

```
(*pdata->set_max_sdma_lat)(struct device *dev, long t)
```

4. Set the minimum bus throughput needed by a device:

```
(*pdata->set_min_bus_tput)(struct device *dev, u8 agent_id, unsigned␣
↪long r)
```

5. Return the number of times the device has lost context:

```
(*pdata->get_dev_context_loss_count)(struct device *dev)
```

Further documentation for all OMAP PM interface functions can be found in arch/arm/plat-omap/include/mach/omap-pm.h.

The OMAP PM layer is intended to be temporary

The intention is that eventually the Linux PM QoS layer should support the range of power management features present in OMAP3. As this happens, existing drivers using the OMAP PM interface can be modified to use the Linux PM QoS code; and the OMAP PM interface can disappear.

Driver usage of the OMAP PM functions

As the 'pdata' in the above examples indicates, these functions are exposed to drivers through function pointers in driver .platform_data structures. The function pointers are initialized by the board-*.c files to point to the corresponding OMAP PM functions:

- set_max_dev_wakeup_lat will point to omap_pm_set_max_dev_wakeup_lat(), etc. Other architectures which do not support these functions should leave these function pointers set to NULL. Drivers should use the following idiom:

```
if (pdata->set_max_dev_wakeup_lat)
    (*pdata->set_max_dev_wakeup_lat)(dev, t);
```

The most common usage of these functions will probably be to specify the maximum time from when an interrupt occurs, to when the device becomes accessible. To accomplish this, driver writers should use the set_max_mpu_wakeup_lat() function to constrain the MPU wakeup latency, and the set_max_dev_wakeup_lat() function to constrain the device wakeup latency (from clk_enable() to accessibility). For example:

```
/* Limit MPU wakeup latency */
if (pdata->set_max_mpu_wakeup_lat)
    (*pdata->set_max_mpu_wakeup_lat)(dev, tc);

/* Limit device powerdomain wakeup latency */
if (pdata->set_max_dev_wakeup_lat)
    (*pdata->set_max_dev_wakeup_lat)(dev, td);

/* total wakeup latency in this example: (tc + td) */
```

The PM parameters can be overwritten by calling the function again with the new value. The settings can be removed by calling the function with a t argument of -1 (except in the case of set_max_bus_tput(), which should be called with an r argument of 0).

The fifth function above, omap_pm_get_dev_context_loss_count(), is intended as an optimization to allow drivers to determine whether the device has lost its internal context. If context has been lost, the driver must restore its internal context before proceeding.

Other specialized interface functions

The five functions listed above are intended to be usable by any device driver. DSPBridge and CPUFreq have a few special requirements. DSPBridge expresses target DSP performance levels in terms of OPP IDs. CPUFreq expresses target MPU performance levels in terms of MPU frequency. The OMAP PM interface contains functions for these specialized cases to convert that input information (OPPs/MPU frequency) into the form that the underlying power management implementation needs:

6. (*pdata->dsp_get_opp_table)(void)
7. (*pdata->dsp_set_min_opp)(u8 opp_id)
8. (*pdata->dsp_get_opp)(void)
9. (*pdata->cpu_get_freq_table)(void)
10. (*pdata->cpu_set_freq)(unsigned long f)
11. (*pdata->cpu_get_freq)(void)

Customizing OPP for platform

Defining CONFIG_PM should enable OPP layer for the silicon and the registration of OPP table should take place automatically. However, in special cases, the default OPP table may need to be tweaked, for e.g.:

- enable default OPPs which are disabled by default, but which could be enabled on a platform
- Disable an unsupported OPP on the platform
- Define and add a custom opp table entry in these cases, the board file needs to do additional steps as follows:

arch/arm/mach-omapx/board-xyz.c:

```
#include "pm.h"
....
static void __init omap_xyz_init_irq(void)
{
        ....
        /* Initialize the default table */
        omapx_opp_init();
        /* Do customization to the defaults */
        ....
}
```

NOTE: omapx_opp_init will be omap3_opp_init or as required based on the omap family.

16.8.3 OMAP2/3 Display Subsystem

This is an almost total rewrite of the OMAP FB driver in drivers/video/omap (let's call it DSS1). The main differences between DSS1 and DSS2 are DSI, TV-out and multiple display support, but there are lots of small improvements also.

The DSS2 driver (omapdss module) is in arch/arm/plat-omap/dss/, and the FB, panel and controller drivers are in drivers/video/omap2/. DSS1 and DSS2 live currently side by side, you can choose which one to use.

Features

Working and tested features include:

- MIPI DPI (parallel) output
- MIPI DSI output in command mode
- MIPI DBI (RFBI) output
- SDI output
- TV output
- All pieces can be compiled as a module or inside kernel
- Use DISPC to update any of the outputs
- Use CPU to update RFBI or DSI output
- OMAP DISPC planes
- RGB16, RGB24 packed, RGB24 unpacked
- YUV2, UYVY
- Scaling
- Adjusting DSS FCK to find a good pixel clock
- Use DSI DPLL to create DSS FCK

Tested boards include: - OMAP3 SDP board - Beagle board - N810

omapdss driver

The DSS driver does not itself have any support for Linux framebuffer, V4L or such like the current ones, but it has an internal kernel API that upper level drivers can use.

The DSS driver models OMAP's overlays, overlay managers and displays in a flexible way to enable non-common multi-display configuration. In addition to modelling the hardware overlays, omapdss supports virtual overlays and overlay managers. These can be used when updating a display with CPU or system DMA.

omapdss driver support for audio

There exist several display technologies and standards that support audio as well. Hence, it is relevant to update the DSS device driver to provide an audio interface that may be used by an audio driver or any other driver interested in the functionality.

The audio_enable function is intended to prepare the relevant IP for playback (e.g., enabling an audio FIFO, taking in/out of reset some IP, enabling companion chips, etc). It is intended to be called before audio_start. The audio_disable function performs the reverse operation and is intended to be called after audio_stop.

While a given DSS device driver may support audio, it is possible that for certain configurations audio is not supported (e.g., an HDMI display using a VESA video timing). The audio_supported function is intended to query whether the current configuration of the display supports audio.

The audio_config function is intended to configure all the relevant audio parameters of the display. In order to make the function independent of any specific DSS device driver, a struct omap_dss_audio is defined. Its purpose is to contain all the required parameters for audio configuration. At the moment, such structure contains pointers to IEC-60958 channel status word and CEA-861 audio infoframe structures. This should be enough to support HDMI and DisplayPort, as both are based on CEA-861 and IEC-60958.

The audio_enable/disable, audio_config and audio_supported functions could be implemented as functions that may sleep. Hence, they should not be called while holding a spinlock or a readlock.

The audio_start/audio_stop function is intended to effectively start/stop audio playback after the configuration has taken place. These functions are designed to be used in an atomic context. Hence, audio_start should return quickly and be called only after all the needed resources for audio playback (audio FIFOs, DMA channels, companion chips, etc) have been enabled to begin data transfers. audio_stop is designed to only stop the audio transfers. The resources used for playback are released using audio_disable.

The enum omap_dss_audio_state may be used to help the implementations of the interface to keep track of the audio state. The initial state is _DISABLED; then, the state transitions to _CONFIGURED, and then, when it is ready to play audio, to _ENABLED. The state _PLAYING is used when the audio is being rendered.

Panel and controller drivers

The drivers implement panel or controller specific functionality and are not usually visible to users except through omapfb driver. They register themselves to the DSS driver.

omapfb driver

The omapfb driver implements arbitrary number of standard linux framebuffers. These framebuffers can be routed flexibly to any overlays, thus allowing very dynamic display architecture.

The driver exports some omapfb specific ioctls, which are compatible with the ioctls in the old driver.

The rest of the non standard features are exported via sysfs. Whether the final implementation will use sysfs, or ioctls, is still open.

V4L2 drivers

V4L2 is being implemented in TI.

From omapdss point of view the V4L2 drivers should be similar to framebuffer driver.

Architecture

Some clarification what the different components do:

- Framebuffer is a memory area inside OMAP’s SRAM/SDRAM that contains the pixel data for the image. Framebuffer has width and height and color depth.
- Overlay defines where the pixels are read from and where they go on the screen. The overlay may be smaller than framebuffer, thus displaying only part of the framebuffer. The position of the overlay may be changed if the overlay is smaller than the display.
- Overlay manager combines the overlays in to one image and feeds them to display.
- Display is the actual physical display device.

A framebuffer can be connected to multiple overlays to show the same pixel data on all of the overlays. Note that in this case the overlay input sizes must be the same, but, in case of video overlays, the output size can be different. Any framebuffer can be connected to any overlay.

An overlay can be connected to one overlay manager. Also DISPC overlays can be connected only to DISPC overlay managers, and virtual overlays can be only connected to virtual overlays.

An overlay manager can be connected to one display. There are certain restrictions which kinds of displays an overlay manager can be connected:

- DISPC TV overlay manager can be only connected to TV display.
- Virtual overlay managers can only be connected to DBI or DSI displays.
- DISPC LCD overlay manager can be connected to all displays, except TV display.

Sysfs

The sysfs interface is mainly used for testing. I don't think sysfs interface is the best for this in the final version, but I don't quite know what would be the best interfaces for these things.

The sysfs interface is divided to two parts: DSS and FB.

/sys/class/graphics/fb? directory: mirror 0=off, 1=on rotate Rotation 0-3 for 0, 90, 180, 270 degrees rotate_type 0 = DMA rotation, 1 = VRFB rotation overlays List of overlay numbers to which framebuffer pixels go phys_addr Physical address of the framebuffer virt_addr Virtual address of the framebuffer size Size of the framebuffer

/sys/devices/platform/omapdss/overlay? directory: enabled 0=off, 1=on input_size width,height (ie. the framebuffer size) manager Destination overlay manager name name name output_size width,height position x,y screen_width width global_alpha global alpha 0-255 0=transparent 255=opaque

/sys/devices/platform/omapdss/manager? directory: display Destination display name alpha_blending_enabled 0=off, 1=on trans_key_enabled 0=off, 1=on trans_key_type gfx-destination, video-source trans_key_value transparency color key (RGB24) default_color default background color (RGB24)

/sys/devices/platform/omapdss/display? directory:

ctrl_name	Controller name
mirror	0=off, 1=on
update_mode	0=off, 1=auto, 2=manual
enabled	0=off, 1=on
name	
rotate	Rotation 0-3 for 0, 90, 180, 270 degrees
timings	Display timings (pixclock,xres/hfp/hbp/hsw,yres/vfp/vbp/vsw) When writing, two special timings are accepted for tv-out: "pal" and "ntsc"
panel_name	
tear_elim	Tearing elimination 0=off, 1=on
output_type	Output type (video encoder only): "composite" or "svideo"

There are also some debugfs files at <debugfs>/omapdss/ which show information about clocks and registers.

Examples

The following definitions have been made for the examples below:

```
ovl0=/sys/devices/platform/omapdss/overlay0
ovl1=/sys/devices/platform/omapdss/overlay1
ovl2=/sys/devices/platform/omapdss/overlay2

mgr0=/sys/devices/platform/omapdss/manager0
mgr1=/sys/devices/platform/omapdss/manager1

lcd=/sys/devices/platform/omapdss/display0
dvi=/sys/devices/platform/omapdss/display1
tv=/sys/devices/platform/omapdss/display2

fb0=/sys/class/graphics/fb0
fb1=/sys/class/graphics/fb1
fb2=/sys/class/graphics/fb2
```

Default setup on OMAP3 SDP

Here’s the default setup on OMAP3 SDP board. All planes go to LCD. DVI and TV-out are not in use. The columns from left to right are: framebuffers, overlays, overlay managers, displays. Framebuffers are handled by omapfb, and the rest by the DSS:

```
FB0 --- GFX  -\            DVI
FB1 --- VID1 --+- LCD ---- LCD
FB2 --- VID2 -/   TV ----- TV
```

Example: Switch from LCD to DVI

```
w=`cat $dvi/timings | cut -d "," -f 2 | cut -d "/" -f 1`
h=`cat $dvi/timings | cut -d "," -f 3 | cut -d "/" -f 1`

echo "0" > $lcd/enabled
echo "" > $mgr0/display
fbset -fb /dev/fb0 -xres $w -yres $h -vxres $w -vyres $h
# at this point you have to switch the dvi/lcd dip-switch from the omap␣
↪board
echo "dvi" > $mgr0/display
echo "1" > $dvi/enabled
```

After this the configuration looks like::

```
FB0 --- GFX  -\         -- DVI
FB1 --- VID1 --+- LCD -/   LCD
FB2 --- VID2 -/   TV ----- TV
```

Example: Clone GFX overlay to LCD and TV

```
w=`cat $tv/timings | cut -d "," -f 2 | cut -d "/" -f 1`
h=`cat $tv/timings | cut -d "," -f 3 | cut -d "/" -f 1`

echo "0" > $ovl0/enabled
echo "0" > $ovl1/enabled

echo "" > $fb1/overlays
echo "0,1" > $fb0/overlays

echo "$w,$h" > $ovl1/output_size
echo "tv" > $ovl1/manager

echo "1" > $ovl0/enabled
echo "1" > $ovl1/enabled

echo "1" > $tv/enabled
```

After this the configuration looks like (only relevant parts shown):

```
FB0 +-- GFX  ---- LCD ---- LCD
\- VID1 ---- TV  ---- TV
```

Misc notes

OMAP FB allocates the framebuffer memory using the standard dma allocator. You can enable Contiguous Memory Allocator (CONFIG_CMA) to improve the dma allocator, and if CMA is enabled, you use "cma=" kernel parameter to increase the global memory area for CMA.

Using DSI DPLL to generate pixel clock it is possible produce the pixel clock of 86.5MHz (max possible), and with that you get 1280x1024@57 output from DVI.

Rotation and mirroring currently only supports RGB565 and RGB8888 modes. VRFB does not support mirroring.

VRFB rotation requires much more memory than non-rotated framebuffer, so you probably need to increase your vram setting before using VRFB rotation. Also, many applications may not work with VRFB if they do not pay attention to all framebuffer parameters.

Kernel boot arguments

omapfb.mode=<display>:<mode>[,…]

- Default video mode for specified displays. For example, "dvi:800x400MR-24@60". See drivers/video/modedb.c. There are also two special modes: "pal" and "ntsc" that can be used to tv out.

omapfb.vram=<fbnum>:<size>[@<physaddr>][,…]

- VRAM allocated for a framebuffer. Normally omapfb allocates vram depending on the display size. With this you can manually allocate more or

define the physical address of each framebuffer. For example, “1:4M” to allocate 4M for fb1.

omapfb.debug=<y|n>

- Enable debug printing. You have to have OMAPFB debug support enabled in kernel config.

omapfb.test=<y|n>

- Draw test pattern to framebuffer whenever framebuffer settings change. You need to have OMAPFB debug support enabled in kernel config.

omapfb.vrfb=<y|n>

- Use VRFB rotation for all framebuffers.

omapfb.rotate=<angle>

- Default rotation applied to all framebuffers. 0 - 0 degree rotation 1 - 90 degree rotation 2 - 180 degree rotation 3 - 270 degree rotation

omapfb.mirror=<y|n>

- Default mirror for all framebuffers. Only works with DMA rotation.

omapdss.def_disp=<display>

- Name of default display, to which all overlays will be connected. Common examples are “lcd” or “tv”.

omapdss.debug=<y|n>

- Enable debug printing. You have to have DSS debug support enabled in kernel config.

TODO

DSS locking

Error checking

- Lots of checks are missing or implemented just as BUG()

System DMA update for DSI

- Can be used for RGB16 and RGB24P modes. Probably not for RGB24U (how to skip the empty byte?)

OMAP1 support

- Not sure if needed

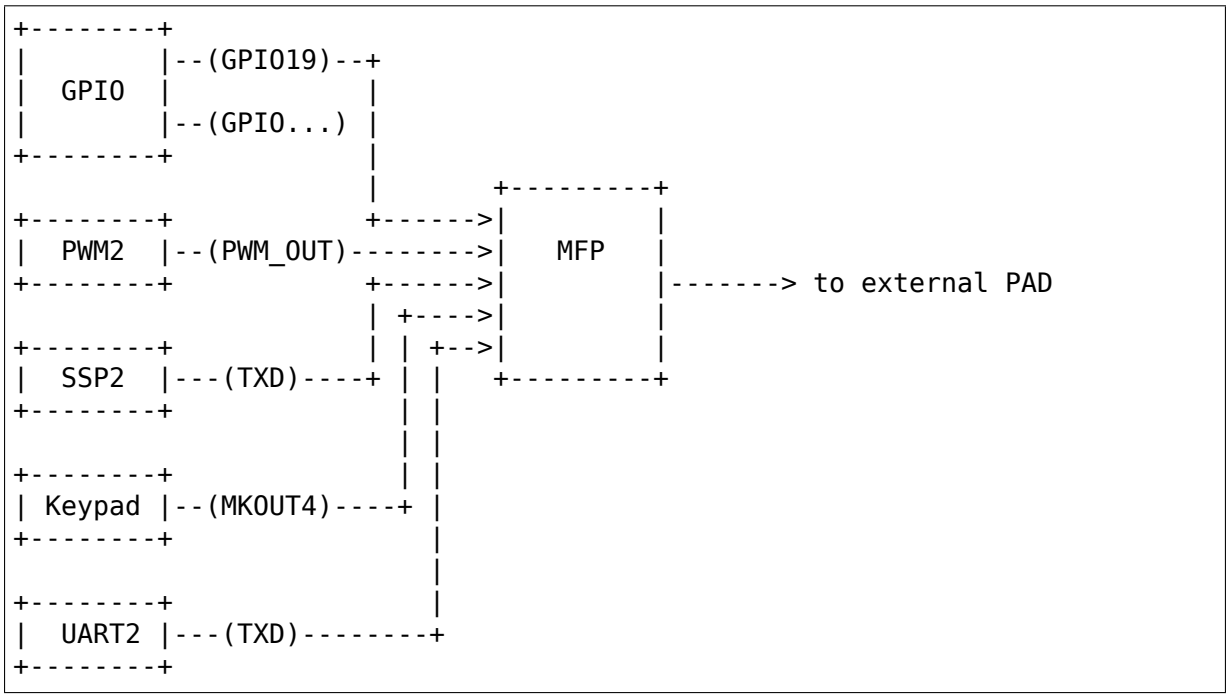
16.9 MFP Configuration for PXA2xx/PXA3xx Processors

Eric Miao <eric.miao@marvell.com>

MFP stands for Multi-Function Pin, which is the pin-mux logic on PXA3xx and later PXA series processors. This document describes the existing MFP API, and how board/platform driver authors could make use of it.

16.9.1 Basic Concept

Unlike the GPIO alternate function settings on PXA25x and PXA27x, a new MFP mechanism is introduced from PXA3xx to completely move the pin-mux functions out of the GPIO controller. In addition to pin-mux configurations, the MFP also controls the low power state, driving strength, pull-up/down and event detection of each pin. Below is a diagram of internal connections between the MFP logic and the remaining SoC peripherals:



NOTE: the external pad is named as MFP_PIN_GPIO19, it doesn't necessarily mean it's dedicated for GPIO19, only as a hint that internally this pin can be routed from GPIO19 of the GPIO controller.

To better understand the change from PXA25x/PXA27x GPIO alternate function to this new MFP mechanism, here are several key points:

1. GPIO controller on PXA3xx is now a dedicated controller, same as other internal controllers like PWM, SSP and UART, with 128 internal signals which can be routed to external through one or more MFPs (e.g. GPIO<0> can be routed through either MFP_PIN_GPIO0 as well as MFP_PIN_GPIO0_2, see arch/arm/mach-pxa/mfp-pxa300.h)

2. Alternate function configuration is removed from this GPIO controller, the remaining functions are pure GPIO-specific, i.e.

- GPIO signal level control
- GPIO direction control
- GPIO level change detection

3. Low power state for each pin is now controlled by MFP, this means the PGSRx registers on PXA2xx are now useless on PXA3xx
4. Wakeup detection is now controlled by MFP, PWER does not control the wakeup from GPIO(s) any more, depending on the sleeping state, ADxER (as defined in pxa3xx-regs.h) controls the wakeup from MFP

NOTE: with such a clear separation of MFP and GPIO, by GPIO<xx> we normally mean it is a GPIO signal, and by MFP<xxx> or pin xxx, we mean a physical pad (or ball).

16.9.2 MFP API Usage

For board code writers, here are some guidelines:

1. include ONE of the following header files in your <board>.c:
   - #include “mfp-pxa25x.h”
   - #include “mfp-pxa27x.h”
   - #include “mfp-pxa300.h”
   - #include “mfp-pxa320.h”
   - #include “mfp-pxa930.h”

   NOTE: only one file in your <board>.c, depending on the processors used, because pin configuration definitions may conflict in these file (i.e. same name, different meaning and settings on different processors). E.g. for zylonite platform, which support both PXA300/PXA310 and PXA320, two separate files are introduced: zylonite_pxa300.c and zylonite_pxa320.c (in addition to handle MFP configuration differences, they also handle the other differences between the two combinations).

   NOTE: PXA300 and PXA310 are almost identical in pin configurations (with PXA310 supporting some additional ones), thus the difference is actually covered in a single mfp-pxa300.h.

2. prepare an array for the initial pin configurations, e.g.:

```
static unsigned long mainstone_pin_config[] __initdata = {
   /* Chip Select */
   GPIO15_nCS_1,

   /* LCD - 16bpp Active TFT */
   GPIOxx_TFT_LCD_16BPP,
   GPIO16_PWM0_OUT,        /* Backlight */

   /* MMC */
   GPIO32_MMC_CLK,
   GPIO112_MMC_CMD,
```

(continues on next page)

(continued from previous page)

```
   GPIO92_MMC_DAT_0,
   GPIO109_MMC_DAT_1,
   GPIO110_MMC_DAT_2,
   GPIO111_MMC_DAT_3,

   ...

   /* GPIO */
   GPIO1_GPIO | WAKEUP_ON_EDGE_BOTH,
};
```

a) once the pin configurations are passed to pxa{2xx,3xx}_mfp_config(), and written to the actual registers, they are useless and may discard, adding '__initdata' will help save some additional bytes here.

b) when there is only one possible pin configurations for a component, some simplified definitions can be used, e.g. GPIOxx_TFT_LCD_16BPP on PXA25x and PXA27x processors

c) if by board design, a pin can be configured to wake up the system from low power state, it can be 'OR' ed with any of:

> WAKEUP_ON_EDGE_BOTH WAKEUP_ON_EDGE_RISE WAKEUP_ON_EDGE_FALL WAKEUP_ON_LEVEL_HIGH - specifically for enabling of keypad GPIOs,

to indicate that this pin has the capability of wake-up the system, and on which edge(s). This, however, doesn't necessarily mean the pin _will_ wakeup the system, it will only when set_irq_wake() is invoked with the corresponding GPIO IRQ (GPIO_IRQ(xx) or gpio_to_irq()) and eventually calls gpio_set_wake() for the actual register setting.

d) although PXA3xx MFP supports edge detection on each pin, the internal logic will only wakeup the system when those specific bits in ADxER registers are set, which can be well mapped to the corresponding peripheral, thus set_irq_wake() can be called with the peripheral IRQ to enable the wakeup.

16.9.3 MFP on PXA3xx

Every external I/O pad on PXA3xx (excluding those for special purpose) has one MFP logic associated, and is controlled by one MFP register (MFPR).

The MFPR has the following bit definitions (for PXA300/PXA310/PXA320):

```
31                      16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1 ␣
↪0
 +-------------------------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
↪+
 |         RESERVED        |PS|PU|PD|  DRIVE |SS|SD|SO|EC|EF|ER|--| AF_SEL␣
↪|
 +-------------------------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
↪+

 Bit 3:   RESERVED
```

(continues on next page)

(continued from previous page)

```
Bit 4:   EDGE_RISE_EN - enable detection of rising edge on this pin
Bit 5:   EDGE_FALL_EN - enable detection of falling edge on this pin
Bit 6:   EDGE_CLEAR   - disable edge detection on this pin
Bit 7:   SLEEP_OE_N   - enable outputs during low power modes
Bit 8:   SLEEP_DATA   - output data on the pin during low power modes
Bit 9:   SLEEP_SEL    - selection control for low power modes signals
Bit 13:  PULLDOWN_EN  - enable the internal pull-down resistor on this pin
Bit 14:  PULLUP_EN    - enable the internal pull-up resistor on this pin
Bit 15:  PULL_SEL     - pull state controlled by selected alternate␣
↪function
                        (0) or by PULL{UP,DOWN}_EN bits (1)

Bit 0 - 2: AF_SEL - alternate function selection, 8 possibilities, from 0-
↪7
Bit 10-12: DRIVE  - drive strength and slew rate
                      0b000 - fast 1mA
                      0b001 - fast 2mA
                      0b002 - fast 3mA
                      0b003 - fast 4mA
                      0b004 - slow 6mA
                      0b005 - fast 6mA
                      0b006 - slow 10mA
                      0b007 - fast 10mA
```

16.9.4 MFP Design for PXA2xx/PXA3xx

Due to the difference of pin-mux handling between PXA2xx and PXA3xx, a unified MFP API is introduced to cover both series of processors.

The basic idea of this design is to introduce definitions for all possible pin configurations, these definitions are processor and platform independent, and the actual API invoked to convert these definitions into register settings and make them effective there-after.

Files Involved

- arch/arm/mach-pxa/include/mach/mfp.h

for

1. Unified pin definitions - enum constants for all configurable pins
2. processor-neutral bit definitions for a possible MFP configuration

- arch/arm/mach-pxa/mfp-pxa3xx.h

for PXA3xx specific MFPR register bit definitions and PXA3xx common pin configurations

- arch/arm/mach-pxa/mfp-pxa2xx.h

for PXA2xx specific definitions and PXA25x/PXA27x common pin configurations

- arch/arm/mach-pxa/mfp-pxa25x.h arch/arm/mach-pxa/mfp-pxa27x.h arch/arm/mach-pxa/mfp-pxa300.h arch/arm/mach-pxa/mfp-pxa320.h arch/arm/mach-pxa/mfp-pxa930.h

for processor specific definitions

- arch/arm/mach-pxa/mfp-pxa3xx.c
- arch/arm/mach-pxa/mfp-pxa2xx.c

for implementation of the pin configuration to take effect for the actual processor.

Pin Configuration

The following comments are copied from mfp.h (see the actual source code for most updated info):

```
 /*
  * a possible MFP configuration is represented by a 32-bit␣
↪integer
  *
  * bit  0.. 9 - MFP Pin Number (1024 Pins Maximum)
  * bit 10..12 - Alternate Function Selection
  * bit 13..15 - Drive Strength
  * bit 16..18 - Low Power Mode State
  * bit 19..20 - Low Power Mode Edge Detection
  * bit 21..22 - Run Mode Pull State
  *
  * to facilitate the definition, the following macros are␣
↪provided
  *
  * MFP_CFG_DEFAULT - default MFP configuration value, with
  *             alternate function = 0,
  *             drive strength = fast 3mA (MFP_DS03X)
  *             low power mode = default
  *             edge detection = none
  *
  * MFP_CFG  - default MFPR value with alternate function
  * MFP_CFG_DRV      - default MFPR value with alternate function␣
↪and
  *             pin drive strength
  * MFP_CFG_LPM      - default MFPR value with alternate function␣
↪and
  *             low power mode
  * MFP_CFG_X        - default MFPR value with alternate function,
  *             pin drive strength and low power mode
  */

Examples of pin configurations are::

  #define GPIO94_SSP3_RXD            MFP_CFG_X(GPIO94, AF1, DS08X,
↪ FLOAT)

which reads GPIO94 can be configured as SSP3_RXD, with alternate␣
↪function
selection of 1, driving strength of 0b101, and a float state in␣
↪low power
```

(continues on next page)

(continued from previous page)

```
modes.

NOTE: this is the default setting of this pin being configured as␣
↪SSP3_RXD
which can be modified a bit in board code, though it is not␣
↪recommended to
do so, simply because this default setting is usually carefully␣
↪encoded,
and is supposed to work in most cases.
```

Register Settings

Register settings on PXA3xx for a pin configuration is actually very straight-forward, most bits can be converted directly into MFPR value in a easier way. Two sets of MFPR values are calculated: the run-time ones and the low power mode ones, to allow different settings.

The conversion from a generic pin configuration to the actual register settings on PXA2xx is a bit complicated: many registers are involved, including GAFRx, GPDRx, PGSRx, PWER, PKWR, PFER and PRER. Please see mfp-pxa2xx.c for how the conversion is made.

16.10 Intel StrongARM 1100

16.10.1 The Intel Assabet (SA-1110 evaluation) board

Please see: http://developer.intel.com

Also some notes from John G Dorsey <jd5q@andrew.cmu.edu>: http://www.cs.cmu.edu/~wearable/software/assabet.html

Building the kernel

To build the kernel with current defaults:

```
make assabet_defconfig
make oldconfig
make zImage
```

The resulting kernel image should be available in linux/arch/arm/boot/zImage.

Installing a bootloader

A couple of bootloaders able to boot Linux on Assabet are available:

BLOB (http://www.lartmaker.nl/lartware/blob/)

> BLOB is a bootloader used within the LART project. Some contributed patches were merged into BLOB to add support for Assabet.

Compaq's Bootldr + John Dorsey's patch for Assabet support (http://www.handhelds.org/Compaq/bootldr.html) (http://www.wearablegroup.org/software/bootldr/)

> Bootldr is the bootloader developed by Compaq for the iPAQ Pocket PC. John Dorsey has produced add-on patches to add support for Assabet and the JFFS filesystem.

RedBoot (http://sources.redhat.com/redboot/)

> RedBoot is a bootloader developed by Red Hat based on the eCos RTOS hardware abstraction layer. It supports Assabet amongst many other hardware platforms.

RedBoot is currently the recommended choice since it's the only one to have networking support, and is the most actively maintained.

Brief examples on how to boot Linux with RedBoot are shown below. But first you need to have RedBoot installed in your flash memory. A known to work precompiled RedBoot binary is available from the following location:

- ftp://ftp.netwinder.org/users/n/nico/
- ftp://ftp.arm.linux.org.uk/pub/linux/arm/people/nico/
- ftp://ftp.handhelds.org/pub/linux/arm/sa-1100-patches/

Look for redboot-assabet*.tgz. Some installation infos are provided in redboot-assabet*.txt.

Initial RedBoot configuration

The commands used here are explained in The RedBoot User's Guide available on-line at http://sources.redhat.com/ecos/docs.html. Please refer to it for explanations.

If you have a CF network card (my Assabet kit contained a CF+ LP-E from Socket Communications Inc.), you should strongly consider using it for TFTP file transfers. You must insert it before RedBoot runs since it can't detect it dynamically.

To initialize the flash directory:

```
fis init -f
```

To initialize the non-volatile settings, like whether you want to use BOOTP or a static IP address, etc, use this command:

```
fconfig -i
```

Writing a kernel image into flash

First, the kernel image must be loaded into RAM. If you have the zImage file available on a TFTP server:

```
load zImage -r -b 0x100000
```

If you rather want to use Y-Modem upload over the serial port:

```
load -m ymodem -r -b 0x100000
```

To write it to flash:

```
fis create "Linux kernel" -b 0x100000 -l 0xc0000
```

Booting the kernel

The kernel still requires a filesystem to boot. A ramdisk image can be loaded as follows:

```
load ramdisk_image.gz -r -b 0x800000
```

Again, Y-Modem upload can be used instead of TFTP by replacing the file name by '-y ymodem' .

Now the kernel can be retrieved from flash like this:

```
fis load "Linux kernel"
```

or loaded as described previously. To boot the kernel:

```
exec -b 0x100000 -l 0xc0000
```

The ramdisk image could be stored into flash as well, but there are better solutions for on-flash filesystems as mentioned below.

Using JFFS2

Using JFFS2 (the Second Journalling Flash File System) is probably the most convenient way to store a writable filesystem into flash. JFFS2 is used in conjunction with the MTD layer which is responsible for low-level flash management. More information on the Linux MTD can be found on-line at: http://www.linux-mtd.infradead.org/. A JFFS howto with some infos about creating JFFS/JFFS2 images is available from the same site.

For instance, a sample JFFS2 image can be retrieved from the same FTP sites mentioned below for the precompiled RedBoot image.

To load this file:

```
load sample_img.jffs2 -r -b 0x100000
```

The result should look like:

```
RedBoot> load sample_img.jffs2 -r -b 0x100000
Raw file loaded 0x00100000-0x00377424
```

Now we must know the size of the unallocated flash:

```
fis free
```

Result:

```
RedBoot> fis free
  0x500E0000 .. 0x503C0000
```

The values above may be different depending on the size of the filesystem and the type of flash. See their usage below as an example and take care of substituting yours appropriately.

We must determine some values:

```
size of unallocated flash:      0x503c0000 - 0x500e0000 = 0x2e0000
size of the filesystem image:   0x00377424 - 0x00100000 = 0x277424
```

We want to fit the filesystem image of course, but we also want to give it all the remaining flash space as well. To write it:

```
fis unlock -f 0x500E0000 -l 0x2e0000
fis erase -f 0x500E0000 -l 0x2e0000
fis write -b 0x100000 -l 0x277424 -f 0x500E0000
fis create "JFFS2" -n -f 0x500E0000 -l 0x2e0000
```

Now the filesystem is associated to a MTD “partition” once Linux has discovered what they are in the boot process. From Redboot, the ‘fis list’ command displays them:

```
RedBoot> fis list
Name              FLASH addr  Mem addr    Length      Entry point
RedBoot           0x50000000  0x50000000  0x00020000  0x00000000
RedBoot config    0x503C0000  0x503C0000  0x00020000  0x00000000
FIS directory     0x503E0000  0x503E0000  0x00020000  0x00000000
Linux kernel      0x50020000  0x00100000  0x000C0000  0x00000000
JFFS2             0x500E0000  0x500E0000  0x002E0000  0x00000000
```

However Linux should display something like:

```
SA1100 flash: probing 32-bit flash bus
SA1100 flash: Found 2 x16 devices at 0x0 in 32-bit mode
Using RedBoot partition definition
Creating 5 MTD partitions on "SA1100 flash":
0x00000000-0x00020000 : "RedBoot"
0x00020000-0x000e0000 : "Linux kernel"
0x000e0000-0x003c0000 : "JFFS2"
0x003c0000-0x003e0000 : "RedBoot config"
0x003e0000-0x00400000 : "FIS directory"
```

What’s important here is the position of the partition we are interested in, which is the third one. Within Linux, this correspond to /dev/mtdblock2. Therefore to

boot Linux with the kernel and its root filesystem in flash, we need this RedBoot command:

```
fis load "Linux kernel"
exec -b 0x100000 -l 0xc0000 -c "root=/dev/mtdblock2"
```

Of course other filesystems than JFFS might be used, like cramfs for example. You might want to boot with a root filesystem over NFS, etc. It is also possible, and sometimes more convenient, to flash a filesystem directly from within Linux while booted from a ramdisk or NFS. The Linux MTD repository has many tools to deal with flash memory as well, to erase it for example. JFFS2 can then be mounted directly on a freshly erased partition and files can be copied over directly. Etc…

RedBoot scripting

All the commands above aren't so useful if they have to be typed in every time the Assabet is rebooted. Therefore it's possible to automate the boot process using RedBoot's scripting capability.

For example, I use this to boot Linux with both the kernel and the ramdisk images retrieved from a TFTP server on the network:

```
RedBoot> fconfig
Run script at boot: false true
Boot script:
Enter script, terminate with empty line
>> load zImage -r -b 0x100000
>> load ramdisk_ks.gz -r -b 0x800000
>> exec -b 0x100000 -l 0xc0000
>>
Boot script timeout (1000ms resolution): 3
Use BOOTP for network configuration: true
GDB connection port: 9000
Network debug at boot time: false
Update RedBoot non-volatile configuration - are you sure (y/n)? y
```

Then, rebooting the Assabet is just a matter of waiting for the login prompt.

Nicolas Pitre nico@fluxnic.net

June 12, 2001

Status of peripherals in -rmk tree (updated 14/10/2001)

Assabet:

Serial ports:

Radio: TX, RX, CTS, DSR, DCD, RI

- PM: Not tested.
- COM: TX, RX, CTS, DSR, DCD, RTS, DTR, PM
- PM: Not tested.
- I2C: Implemented, not fully tested.

- L3: Fully tested, pass.
- PM: Not tested.

Video:

- LCD: Fully tested. PM

  (LCD doesn't like being blanked with neponset connected)

- Video out: Not fully

Audio: UDA1341: - Playback: Fully tested, pass. - Record: Implemented, not tested. - PM: Not tested.

UCB1200: - Audio play: Implemented, not heavily tested. - Audio rec: Implemented, not heavily tested. - Telco audio play: Implemented, not heavily tested. - Telco audio rec: Implemented, not heavily tested. - POTS control: No - Touchscreen: Yes - PM: Not tested.

Other:

- PCMCIA:
- LPE: Fully tested, pass.
- USB: No
- IRDA:
- SIR: Fully tested, pass.
- FIR: Fully tested, pass.
- PM: Not tested.

Neponset:

Serial ports:

- COM1,2: TX, RX, CTS, DSR, DCD, RTS, DTR
- PM: Not tested.
- USB: Implemented, not heavily tested.
- PCMCIA: Implemented, not heavily tested.
- CF: Implemented, not heavily tested.
- PM: Not tested.

More stuff can be found in the -np (Nicolas Pitre's) tree.

16.10.2 CerfBoard/Cube

*** The StrongARM version of the CerfBoard/Cube has been discontinued ***

The Intrinsyc CerfBoard is a StrongARM 1110-based computer on a board that measures approximately 2” square. It includes an Ethernet controller, an RS232-compatible serial port, a USB function port, and one CompactFlash+ slot on the back. Pictures can be found at the Intrinsyc website, http://www.intrinsyc.com.

This document describes the support in the Linux kernel for the Intrinsyc CerfBoard.

Supported in this version

- CompactFlash+ slot (select PCMCIA in General Setup and any options that may be required)
- Onboard Crystal CS8900 Ethernet controller (Cerf CS8900A support in Network Devices)
- Serial ports with a serial console (hardcoded to 38400 8N1)

In order to get this kernel onto your Cerf, you need a server that runs both BOOTP and TFTP. Detailed instructions should have come with your evaluation kit on how to use the bootloader. This series of commands will suffice:

```
make ARCH=arm CROSS_COMPILE=arm-linux- cerfcube_defconfig
make ARCH=arm CROSS_COMPILE=arm-linux- zImage
make ARCH=arm CROSS_COMPILE=arm-linux- modules
cp arch/arm/boot/zImage <TFTP directory>
```

support@intrinsyc.com

16.10.3 Linux Advanced Radio Terminal (LART)

The LART is a small (7.5 x 10cm) SA-1100 board, designed for embedded applications. It has 32 MB DRAM, 4MB Flash ROM, double RS232 and all other StrongARM-gadgets. Almost all SA signals are directly accessible through a number of connectors. The powersupply accepts voltages between 3.5V and 16V and is overdimensioned to support a range of daughterboards. A quad Ethernet / IDE / PS2 / sound daughterboard is under development, with plenty of others in different stages of planning.

The hardware designs for this board have been released under an open license; see the LART page at http://www.lartmaker.nl/ for more information.

16.10.4 SA1100 serial port

The SA1100 serial port had its major/minor numbers officially assigned:

```
> Date: Sun, 24 Sep 2000 21:40:27 -0700
> From: H. Peter Anvin <hpa@transmeta.com>
> To: Nicolas Pitre <nico@CAM.ORG>
> Cc: Device List Maintainer <device@lanana.org>
> Subject: Re: device
>
> Okay.  Note that device numbers 204 and 205 are used for "low density
> serial devices", so you will have a range of minors on those majors (the
> tty device layer handles this just fine, so you don't have to worry about
> doing anything special.)
>
> So your assignments are:
>
> 204 char        Low-density serial ports
>                   5 = /dev/ttySA0               SA1100 builtin serial␣
 ↪port 0
>                   6 = /dev/ttySA1               SA1100 builtin serial␣
 ↪port 1
>                   7 = /dev/ttySA2               SA1100 builtin serial␣
 ↪port 2
>
> 205 char        Low-density serial ports (alternate device)
>                   5 = /dev/cusa0                Callout device for ttySA0
>                   6 = /dev/cusa1                Callout device for ttySA1
>                   7 = /dev/cusa2                Callout device for ttySA2
>
```

You must create those inodes in /dev on the root filesystem used by your SA1100-based device:

```
mknod ttySA0 c 204 5
mknod ttySA1 c 204 6
mknod ttySA2 c 204 7
mknod cusa0 c 205 5
mknod cusa1 c 205 6
mknod cusa2 c 205 7
```

In addition to the creation of the appropriate device nodes above, you must ensure your user space applications make use of the correct device name. The classic example is the content of the /etc/inittab file where you might have a getty process started on ttyS0.

In this case:

- replace occurrences of ttyS0 with ttySA0, ttyS1 with ttySA1, etc.
- don’t forget to add ‘ttySA0’, ‘console’, or the appropriate tty name in /etc/securetty for root to be allowed to login as well.

16.11 STM32F746 Overview

16.11.1 Introduction

The STM32F746 is a Cortex-M7 MCU aimed at various applications. It features:

- Cortex-M7 core running up to @216MHz
- 1MB internal flash, 320KBytes internal RAM (+4KB of backup SRAM)
- FMC controller to connect SDRAM, NOR and NAND memories
- Dual mode QSPI
- SD/MMC/SDIO support
- Ethernet controller
- USB OTFG FS & HS controllers
- I2C, SPI, CAN busses support
- Several 16 & 32 bits general purpose timers
- Serial Audio interface
- LCD controller
- HDMI-CEC
- SPDIFRX

16.11.2 Resources

Datasheet and reference manual are publicly available on ST website (STM32F746).

Authors Alexandre Torgue <alexandre.torgue@st.com>

16.12 STM32 ARM Linux Overview

16.12.1 Introduction

The STMicroelectronics STM32 family of Cortex-A microprocessors (MPUs) and Cortex-M microcontrollers (MCUs) are supported by the 'STM32' platform of ARM Linux.

16.12.2 Configuration

For MCUs, use the provided default configuration: make stm32_defconfig

For MPUs, use multi_v7 configuration: make multi_v7_defconfig

16.12.3 Layout

All the files for multiple machine families are located in the platform code contained in arch/arm/mach-stm32

There is a generic board board-dt.c in the mach folder which support Flattened Device Tree, which means, it works with any compatible board with Device Trees.

Authors

- Maxime Coquelin <mcoquelin.stm32@gmail.com>
- Ludovic Barre <ludovic.barre@st.com>
- Gerald Baeza <gerald.baeza@st.com>

16.13 STM32H743 Overview

16.13.1 Introduction

The STM32H743 is a Cortex-M7 MCU aimed at various applications. It features:

- Cortex-M7 core running up to @400MHz
- 2MB internal flash, 1MBytes internal RAM
- FMC controller to connect SDRAM, NOR and NAND memories
- Dual mode QSPI
- SD/MMC/SDIO support
- Ethernet controller
- USB OTFG FS & HS controllers
- I2C, SPI, CAN busses support
- Several 16 & 32 bits general purpose timers
- Serial Audio interface
- LCD controller
- HDMI-CEC
- SPDIFRX
- DFSDM

16.13.2 Resources

Datasheet and reference manual are publicly available on ST website (STM32H743).

Authors Alexandre Torgue <alexandre.torgue@st.com>

16.14 STM32F769 Overview

16.14.1 Introduction

The STM32F769 is a Cortex-M7 MCU aimed at various applications. It features:

- Cortex-M7 core running up to @216MHz
- 2MB internal flash, 512KBytes internal RAM (+4KB of backup SRAM)
- FMC controller to connect SDRAM, NOR and NAND memories
- Dual mode QSPI
- SD/MMC/SDIO support*2
- Ethernet controller
- USB OTFG FS & HS controllers
- I2C*4, SPI*6, CAN*3 busses support
- Several 16 & 32 bits general purpose timers
- Serial Audio interface*2
- LCD controller
- HDMI-CEC
- DSI
- SPDIFRX
- MDIO salave interface

16.14.2 Resources

Datasheet and reference manual are publicly available on ST website (STM32F769).

Authors Alexandre Torgue <alexandre.torgue@st.com>

16.15 STM32F429 Overview

16.15.1 Introduction

The STM32F429 is a Cortex-M4 MCU aimed at various applications. It features:

- ARM Cortex-M4 up to 180MHz with FPU
- 2MB internal Flash Memory
- External memory support through FMC controller (PSRAM, SDRAM, NOR, NAND)
- I2C, SPI, SAI, CAN, USB OTG, Ethernet controllers
- LCD controller & Camera interface
- Cryptographic processor

16.15.2 Resources

Datasheet and reference manual are publicly available on ST website (STM32F429).

Authors Maxime Coquelin <mcoquelin.stm32@gmail.com>

16.16 STM32MP157 Overview

16.16.1 Introduction

The STM32MP157 is a Cortex-A MPU aimed at various applications. It features:

- Dual core Cortex-A7 application core
- 2D/3D image composition with GPU
- Standard memories interface support
- Standard connectivity, widely inherited from the STM32 MCU family
- Comprehensive security support

Authors

- Ludovic Barre <ludovic.barre@st.com>
- Gerald Baeza <gerald.baeza@st.com>

16.17 ARM Allwinner SoCs

This document lists all the ARM Allwinner SoCs that are currently supported in mainline by the Linux kernel. This document will also provide links to documentation and/or datasheet for these SoCs.

16.17.1 SunXi family

Linux kernel mach directory: arch/arm/mach-sunxi

Flavors:

- ARM926 based SoCs - Allwinner F20 (sun3i)
  - Not Supported
- ARM Cortex-A8 based SoCs - Allwinner A10 (sun4i)
  - Datasheet

    http://dl.linux-sunxi.org/A10/A10%20Datasheet%20-%20v1.21%20%282012-04-06%29.pdf
  - User Manual

    http://dl.linux-sunxi.org/A10/A10%20User%20Manual%20-%20v1.20%20%282012-04-09%2c%20DECRYPTED%29.pdf
  - Allwinner A10s (sun5i)
    - Datasheet

      http://dl.linux-sunxi.org/A10s/A10s%20Datasheet%20-%20v1.20%20%282012-03-27%29.pdf
  - Allwinner A13 / R8 (sun5i)
    - Datasheet

      http://dl.linux-sunxi.org/A13/A13%20Datasheet%20-%20v1.12%20%282012-03-29%29.pdf
    - User Manual

      http://dl.linux-sunxi.org/A13/A13%20User%20Manual%20-%20v1.2%20%282013-01-08%29.pdf
  - Next Thing Co GR8 (sun5i)
- Single ARM Cortex-A7 based SoCs - Allwinner V3s (sun8i)
  - Datasheet

    http://linux-sunxi.org/File:Allwinner_V3s_Datasheet_V1.0.pdf
- Dual ARM Cortex-A7 based SoCs - Allwinner A20 (sun7i)
  - User Manual

    http://dl.linux-sunxi.org/A20/A20%20User%20Manual%202013-03-22.pdf

  - Allwinner A23 (sun8i)
    * Datasheet

      http://dl.linux-sunxi.org/A23/A23%20Datasheet%20V1.0%2020130830.pdf

    * User Manual

      http://dl.linux-sunxi.org/A23/A23%20User%20Manual%20V1.0%2020130830.pdf

- Quad ARM Cortex-A7 based SoCs - Allwinner A31 (sun6i)
  - Datasheet

    http://dl.linux-sunxi.org/A31/A3x_release_document/A31/IC/A31%20datasheet%20V1.3%2020131106.pdf

  - User Manual

    http://dl.linux-sunxi.org/A31/A3x_release_document/A31/IC/A31%20user%20manual%20V1.1%2020130630.pdf

  - Allwinner A31s (sun6i)
    * Datasheet

      http://dl.linux-sunxi.org/A31/A3x_release_document/A31s/IC/A31s%20datasheet%20V1.3%2020131106.pdf

    * User Manual

      http://dl.linux-sunxi.org/A31/A3x_release_document/A31s/IC/A31s%20User%20Manual%20%20V1.0%2020130322.pdf

  - Allwinner A33 (sun8i)
    * Datasheet

      http://dl.linux-sunxi.org/A33/A33%20Datasheet%20release%201.1.pdf

    * User Manual

      http://dl.linux-sunxi.org/A33/A33%20user%20manual%20release%201.1.pdf

  - Allwinner H2+ (sun8i)
    * No document available now, but is known to be working properly with H3 drivers and memory map.

  - Allwinner H3 (sun8i)
    * Datasheet

      http://dl.linux-sunxi.org/H3/Allwinner_H3_Datasheet_V1.0.pdf

  - Allwinner R40 (sun8i)

    * Datasheet

      https://github.com/tinalinux/docs/raw/r40-v1.y/R40_Datasheet_V1.0.pdf

    * User Manual

      https://github.com/tinalinux/docs/raw/r40-v1.y/Allwinner_R40_User_Manual_V1.0.pdf

- Quad ARM Cortex-A15, Quad ARM Cortex-A7 based SoCs - Allwinner A80
  - Datasheet

    http://dl.linux-sunxi.org/A80/A80_Datasheet_Revision_1.0_0404.pdf

- Octa ARM Cortex-A7 based SoCs - Allwinner A83T
  - Datasheet

    https://github.com/allwinner-zh/documents/raw/master/A83T/A83T_Datasheet_v1.3_20150510.pdf

  - User Manual

    https://github.com/allwinner-zh/documents/raw/master/A83T/A83T_User_Manual_v1.5.1_20150513.pdf

- Quad ARM Cortex-A53 based SoCs - Allwinner A64
  - Datasheet

    http://dl.linux-sunxi.org/A64/A64_Datasheet_V1.1.pdf

  - User Manual

    http://dl.linux-sunxi.org/A64/Allwinner%20A64%20User%20Manual%20v1.0.pdf

16.18 Samsung SoC

16.18.1 Samsung GPIO implementation

Introduction

This outlines the Samsung GPIO implementation and the architecture specific calls provided alongside the drivers/gpio core.

S3C24XX (Legacy)

See Documentation/arm/samsung-s3c24xx/gpio.rst for more information about these devices. Their implementation has been brought into line with the core samsung implementation described in this document.

GPIOLIB integration

The gpio implementation uses gpiolib as much as possible, only providing specific calls for the items that require Samsung specific handling, such as pin special-function or pull resistor control.

GPIO numbering is synchronised between the Samsung and gpiolib system.

PIN configuration

Pin configuration is specific to the Samsung architecture, with each SoC registering the necessary information for the core gpio configuration implementation to configure pins as necessary.

The s3c_gpio_cfgpin() and s3c_gpio_setpull() provide the means for a driver or machine to change gpio configuration.

See arch/arm/plat-samsung/include/plat/gpio-cfg.h for more information on these functions.

16.18.2 Interface between kernel and boot loaders on Exynos boards

Author: Krzysztof Kozlowski

Date : 6 June 2015

The document tries to describe currently used interface between Linux kernel and boot loaders on Samsung Exynos based boards. This is not a definition of interface but rather a description of existing state, a reference for information purpose only.

In the document "boot loader" means any of following: U-boot, proprietary SBOOT or any other firmware for ARMv7 and ARMv8 initializing the board before executing kernel.

1. Non-Secure mode

Address: sysram_ns_base_addr

Offset	Value	Purpose
0x08	exynos_cpu_resume_ns, mcpm_entry_point	System suspend
0x0c	0x00000bad (Magic cookie)	System suspend
0x1c	exynos4_secondary_startup	Secondary CPU boot
0x1c + 4*cpu	exynos4_secondary_startup (Exynos4412)	Secondary CPU boot
0x20	0xfcba0d10 (Magic cookie)	AFTR
0x24	exynos_cpu_resume_ns	AFTR
0x28 + 4*cpu	0x8 (Magic cookie, Exynos3250)	AFTR
0x28	0x0 or last value during resume (Exynos542x)	System suspend

2. Secure mode

Address: sysram_base_addr

Offset	Value	Purpose
0x00	exynos4_secondary_startup	Secondary CPU boot
0x04	exynos4_secondary_startup (Exynos542x)	Secondary CPU boot
4*cpu	exynos4_secondary_startup (Exynos4412)	Secondary CPU boot
0x20	exynos_cpu_resume (Exynos4210 r1.0)	AFTR
0x24	0xfcba0d10 (Magic cookie, Exynos4210 r1.0)	AFTR

Address: pmu_base_addr

Offset	Value	Purpose
0x0800	exynos_cpu_resume	AFTR, suspend
0x0800	mcpm_entry_point (Exynos542x with MCPM)	AFTR, suspend
0x0804	0xfcba0d10 (Magic cookie)	AFTR
0x0804	0x00000bad (Magic cookie)	System suspend
0x0814	exynos4_secondary_startup (Exynos4210 r1.1)	Secondary CPU boot
0x0818	0xfcba0d10 (Magic cookie, Exynos4210 r1.1)	AFTR
0x081C	exynos_cpu_resume (Exynos4210 r1.1)	AFTR

3. Other (regardless of secure/non-secure mode)

Address: pmu_base_addr

Offset	Value	Purpose
0x0908	Non-zero	Secondary CPU boot up indicator on Exynos3250 and Exynos542x

4. Glossary

AFTR - ARM Off Top Running, a low power mode, Cortex cores and many other modules are power gated, except the TOP modules MCPM - Multi-Cluster Power Management

16.18.3 Samsung ARM Linux Overview

Introduction

The Samsung range of ARM SoCs spans many similar devices, from the initial ARM9 through to the newest ARM cores. This document shows an overview of the current kernel support, how to use it and where to find the code that supports this.

The currently supported SoCs are:

- S3C24XX: See Documentation/arm/samsung-s3c24xx/overview.rst for full list
- S3C64XX: S3C6400 and S3C6410
- S5PC110 / S5PV210

S3C24XX Systems

There is still documentation in Documnetation/arm/Samsung-S3C24XX/ which deals with the architecture and drivers specific to these devices.

See Documentation/arm/samsung-s3c24xx/overview.rst for more information on the implementation details and specific support.

Configuration

A number of configurations are supplied, as there is no current way of unifying all the SoCs into one kernel.

s5pc110_defconfig

- S5PC110 specific default configuration

s5pv210_defconfig

- S5PV210 specific default configuration

Layout

The directory layout is currently being restructured, and consists of several platform directories and then the machine specific directories of the CPUs being built for.

plat-samsung provides the base for all the implementations, and is the last in the line of include directories that are processed for the build specific information. It contains the base clock, GPIO and device definitions to get the system running.

plat-s3c24xx is for s3c24xx specific builds, see the S3C24XX docs.

plat-s5p is for s5p specific builds, and contains common support for the S5P specific systems. Not all S5Ps use all the features in this directory due to differences in the hardware.

Layout changes

The old plat-s3c and plat-s5pc1xx directories have been removed, with support moved to either plat-samsung or plat-s5p as necessary. These moves where to simplify the include and dependency issues involved with having so many different platform directories.

Port Contributors

Ben Dooks (BJD) Vincent Sanders Herbert Potzl Arnaud Patard (RTP) Roc Wu Klaus Fetscher Dimitry Andric Shannon Holland Guillaume Gourat (NexVision) Christer Weinigel (wingel) (Acer N30) Lucas Correia Villa Real (S3C2400 port)

Document Author

Copyright 2009-2010 Ben Dooks <ben-linux@fluff.org>

16.19 Samsung S3C24XX SoC Family

16.19.1 HP IPAQ H1940

http://www.handhelds.org/projects/h1940.html

Introduction

The HP H1940 is a S3C2410 based handheld device, with bluetooth connectivity.

Support

A variety of information is available

handhelds.org project page:

http://www.handhelds.org/projects/h1940.html

handhelds.org wiki page:

http://handhelds.org/moin/moin.cgi/HpIpaqH1940

Herbert Pötzl pages:

http://vserver.13thfloor.at/H1940/

Maintainers

This project is being maintained and developed by a variety of people, including Ben Dooks, Arnaud Patard, and Herbert Pötzl.

Thanks to the many others who have also provided support.

(c) 2005 Ben Dooks

16.19.2 S3C24XX GPIO Control

Introduction

The s3c2410 kernel provides an interface to configure and manipulate the state of the GPIO pins, and find out other information about them.

There are a number of conditions attached to the configuration of the s3c2410 GPIO system, please read the Samsung provided data-sheet/users manual to find out the complete list.

See Documentation/arm/samsung/gpio.rst for the core implementation.

GPIOLIB

With the event of the GPIOLIB in drivers/gpio, support for some of the GPIO functions such as reading and writing a pin will be removed in favour of this common access method.

Once all the extant drivers have been converted, the functions listed below will be removed (they may be marked as __deprecated in the near future).

The following functions now either have a s3c_ specific variant or are merged into gpiolib. See the definitions in arch/arm/plat-samsung/include/plat/gpio-cfg.h:

- s3c2410_gpio_setpin() gpio_set_value() or gpio_direction_output()
- s3c2410_gpio_getpin() gpio_get_value() or gpio_direction_input()
- s3c2410_gpio_getirq() gpio_to_irq()
- s3c2410_gpio_cfgpin() s3c_gpio_cfgpin()
- s3c2410_gpio_getcfg() s3c_gpio_getcfg()
- s3c2410_gpio_pullup() s3c_gpio_setpull()

GPIOLIB conversion

If you need to convert your board or driver to use gpiolib from the phased out s3c2410 API, then here are some notes on the process.

1) If your board is exclusively using an GPIO, say to control peripheral power, then it will require to claim the gpio with gpio_request() before it can use it.

   It is recommended to check the return value, with at least WARN_ON() during initialisation.

2) The s3c2410_gpio_cfgpin() can be directly replaced with s3c_gpio_cfgpin() as they have the same arguments, and can either take the pin specific values, or the more generic special-function-number arguments.

3) s3c2410_gpio_pullup() changes have the problem that while the s3c2410_gpio_pullup(x, 1) can be easily translated to the s3c_gpio_setpull(x, S3C_GPIO_PULL_NONE), the s3c2410_gpio_pullup(x, 0) are not so easy.

   The s3c2410_gpio_pullup(x, 0) case enables the pull-up (or in the case of some of the devices, a pull-down) and as such the new API distinguishes between the UP and DOWN case. There is currently no 'just turn on' setting which may be required if this becomes a problem.

4) s3c2410_gpio_setpin() can be replaced by gpio_set_value(), the old call does not implicitly configure the relevant gpio to output. The gpio direction should be changed before using gpio_set_value().

5) s3c2410_gpio_getpin() is replaceable by gpio_get_value() if the pin has been set to input. It is currently unknown what the behaviour is when using gpio_get_value() on an output pin (s3c2410_gpio_getpin would return the value the pin is supposed to be outputting).

6) s3c2410_gpio_getirq() should be directly replaceable with the gpio_to_irq() call.

The s3c2410_gpio and gpio_ calls have always operated on the same gpio numberspace, so there is no problem with converting the gpio numbering between the calls.

Headers

See arch/arm/mach-s3c24xx/include/mach/regs-gpio.h for the list of GPIO pins, and the configuration values for them. This is included by using #include <mach/regs-gpio.h>

PIN Numbers

Each pin has an unique number associated with it in regs-gpio.h, e.g. S3C2410_GPA(0) or S3C2410_GPF(1). These defines are used to tell the GPIO functions which pin is to be used.

With the conversion to gpiolib, there is no longer a direct conversion from gpio pin number to register base address as in earlier kernels. This is due to the number space required for newer SoCs where the later GPIOs are not contiguous.

Configuring a pin

The following function allows the configuration of a given pin to be changed.

```
void s3c_gpio_cfgpin(unsigned int pin, unsigned int function);
```

e.g.:

```
s3c_gpio_cfgpin(S3C2410_GPA(0),  S3C_GPIO_SFN(1));
s3c_gpio_cfgpin(S3C2410_GPE(8), S3C_GPIO_SFN(2));
```

which would turn GPA(0) into the lowest Address line A0, and set GPE(8) to be connected to the SDIO/MMC controller's SD-DAT1 line.

Reading the current configuration

The current configuration of a pin can be read by using standard gpiolib function:

s3c_gpio_getcfg(unsigned int pin);

The return value will be from the same set of values which can be passed to s3c_gpio_cfgpin().

Configuring a pull-up resistor

A large proportion of the GPIO pins on the S3C2410 can have weak pull-up resistors enabled. This can be configured by the following function:

```
void s3c_gpio_setpull(unsigned int pin, unsigned int to);
```

Where the to value is S3C_GPIO_PULL_NONE to set the pull-up off, and S3C_GPIO_PULL_UP to enable the specified pull-up. Any other values are currently undefined.

Getting and setting the state of a PIN

These calls are now implemented by the relevant gpiolib calls, convert your board or driver to use gpiolib.

Getting the IRQ number associated with a PIN

A standard gpiolib function can map the given pin number to an IRQ number to pass to the IRQ system.

```
int gpio_to_irq(unsigned int pin);
```

Note, not all pins have an IRQ.

Author

Ben Dooks, 03 October 2004 Copyright 2004 Ben Dooks, Simtec Electronics

16.19.3 S3C24XX CPUfreq support

Introduction

The S3C24XX series support a number of power saving systems, such as the ability to change the core, memory and peripheral operating frequencies. The core control is exported via the CPUFreq driver which has a number of different manual or automatic controls over the rate the core is running at.

There are two forms of the driver depending on the specific CPU and how the clocks are arranged. The first implementation used as single PLL to feed the ARM, memory and peripherals via a series of dividers and muxes and this is the implementation that is documented here. A newer version where there is a separate PLL and clock divider for the ARM core is available as a separate driver.

Layout

The code core manages the CPU specific drivers, any data that they need to register and the interface to the generic drivers/cpufreq system. Each CPU registers a driver to control the PLL, clock dividers and anything else associated with it. Any board that wants to use this framework needs to supply at least basic details of what is required.

The core registers with drivers/cpufreq at init time if all the data necessary has been supplied.

CPU support

The support for each CPU depends on the facilities provided by the SoC and the driver as each device has different PLL and clock chains associated with it.

Slow Mode

The SLOW mode where the PLL is turned off altogether and the system is fed by the external crystal input is currently not supported.

sysfs

The core code exports extra information via sysfs in the directory devices/system/cpu/cpu0/arch-freq.

Board Support

Each board that wants to use the cpufreq code must register some basic information with the core driver to provide information about what the board requires and any restrictions being placed on it.

The board needs to supply information about whether it needs the IO bank timings changing, any maximum frequency limits and information about the SDRAM refresh rate.

Document Author

Ben Dooks, Copyright 2009 Simtec Electronics Licensed under GPLv2

16.19.4 S3C24XX Suspend Support

Introduction

The S3C24XX supports a low-power suspend mode, where the SDRAM is kept in Self-Refresh mode, and all but the essential peripheral blocks are powered down. For more information on how this works, please look at the relevant CPU datasheet from Samsung.

Requirements

1) A bootloader that can support the necessary resume operation
2) Support for at least 1 source for resume
3) CONFIG_PM enabled in the kernel
4) Any peripherals that are going to be powered down at the same time require suspend/resume support.

Resuming

The S3C2410 user manual defines the process of sending the CPU to sleep and how it resumes. The default behaviour of the Linux code is to set the GSTATUS3 register to the physical address of the code to resume Linux operation.

GSTATUS4 is currently left alone by the sleep code, and is free to use for any other purposes (for example, the EB2410ITX uses this to save memory configuration in).

Machine Support

The machine specific functions must call the s3c_pm_init() function to say that its bootloader is capable of resuming. This can be as simple as adding the following to the machine's definition:

INITMACHINE(s3c_pm_init)

A board can do its own setup before calling s3c_pm_init, if it needs to setup anything else for power management support.

There is currently no support for over-riding the default method of saving the resume address, if your board requires it, then contact the maintainer and discuss what is required.

Note, the original method of adding an late_initcall() is wrong, and will end up initialising all compiled machines' pm init!

The following is an example of code used for testing wakeup from an falling edge on IRQ_EINT0:

```
static irqreturn_t button_irq(int irq, void *pw)
{
    return IRQ_HANDLED;
}

statuc void __init machine_init(void)
{
    ...

    request_irq(IRQ_EINT0, button_irq, IRQF_TRIGGER_FALLING,
                "button-irq-eint0", NULL);

    enable_irq_wake(IRQ_EINT0);

    s3c_pm_init();
}
```

Debugging

There are several important things to remember when using PM suspend:

1) The uart drivers will disable the clocks to the UART blocks when suspending, which means that use of printascii() or similar direct access to the UARTs will cause the debug to stop.

2) While the pm code itself will attempt to re-enable the UART clocks, care should be taken that any external clock sources that the UARTs rely on are still enabled at that point.

3) If any debugging is placed in the resume path, then it must have the relevant clocks and peripherals setup before use (ie, bootloader).

   For example, if you transmit a character from the UART, the baud rate and uart controls must be setup beforehand.

Configuration

The S3C2410 specific configuration in System Type defines various aspects of how the S3C2410 suspend and resume support is configured

S3C2410 PM Suspend debug

> This option prints messages to the serial console before and after the actual suspend, giving detailed information on what is happening

S3C2410 PM Suspend Memory CRC

> Allows the entire memory to be checksummed before and after the suspend to see if there has been any corruption of the contents.
>
> Note, the time to calculate the CRC is dependent on the CPU speed and the size of memory. For an 64Mbyte RAM area on

an 200MHz S3C2410, this can take approximately 4 seconds to complete.

This support requires the CRC32 function to be enabled.

S3C2410 PM Suspend CRC Chunksize (KiB)

Defines the size of memory each CRC chunk covers. A smaller value will mean that the CRC data block will take more memory, but will identify any faults with better precision

Document Author

Ben Dooks, Copyright 2004 Simtec Electronics

16.19.5 S3C24XX USB Host support

Introduction

This document details the S3C2410/S3C2440 in-built OHCI USB host support.

Configuration

Enable at least the following kernel options:

menuconfig:

```
Device Drivers  --->
  USB support  --->
    <*> Support for Host-side USB
    <*>   OHCI HCD support
```

.config:

- CONFIG_USB
- CONFIG_USB_OHCI_HCD

Once these options are configured, the standard set of USB device drivers can be configured and used.

Board Support

The driver attaches to a platform device, which will need to be added by the board specific support file in linux/arch/arm/mach-s3c2410, such as mach-bast.c or mach-smdk2410.c

The platform device' s platform_data field is only needed if the board implements extra power control or over-current monitoring.

The OHCI driver does not ensure the state of the S3C2410' s MISCCTRL register, so if both ports are to be used for the host, then it is the board

support file's responsibility to ensure that the second port is configured to be connected to the OHCI core.

Platform Data

See arch/arm/mach-s3c2410/include/mach/usb-control.h for the descriptions of the platform device data. An implementation can be found in linux/arch/arm/mach-s3c2410/usb-simtec.c .

The struct s3c2410_hcd_info contains a pair of functions that get called to enable over-current detection, and to control the port power status.

The ports are numbered 0 and 1.

power_control: Called to enable or disable the power on the port.

enable_oc: Called to enable or disable the over-current monitoring. This should claim or release the resources being used to check the power condition on the port, such as an IRQ.

report_oc: The OHCI driver fills this field in for the over-current code to call when there is a change to the over-current state on an port. The ports argument is a bitmask of 1 bit per port, with bit X being 1 for an over-current on port X.

The function s3c2410_usb_report_oc() has been provided to ensure this is called correctly.

port[x]: This is struct describes each port, 0 or 1. The platform driver should set the flags field of each port to S3C_HCDFLG_USED if the port is enabled.

Document Author

Ben Dooks, Copyright 2005 Simtec Electronics

16.19.6 S3C2412 ARM Linux Overview

Introduction

The S3C2412 is part of the S3C24XX range of ARM9 System-on-Chip CPUs from Samsung. This part has an ARM926-EJS core, capable of running up to 266MHz (see data-sheet for more information)

Clock

The core clock code provides a set of clocks to the drivers, and allows for source selection and a number of other features.

Power

No support for suspend/resume to RAM in the current system.

DMA

No current support for DMA.

GPIO

There is support for setting the GPIO to input/output/special function and reading or writing to them.

UART

The UART hardware is similar to the S3C2440, and is supported by the s3c2410 driver in the drivers/serial directory.

NAND

The NAND hardware is similar to the S3C2440, and is supported by the s3c2410 driver in the drivers/mtd/nand/raw directory.

USB Host

The USB hardware is similar to the S3C2410, with extended clock source control. The OHCI portion is supported by the ohci-s3c2410 driver, and the clock control selection is supported by the core clock code.

USB Device

No current support in the kernel

IRQs

All the standard, and external interrupt sources are supported. The extra sub-sources are not yet supported.

RTC

The RTC hardware is similar to the S3C2410, and is supported by the s3c2410-rtc driver.

Watchdog

The watchdog hardware is the same as the S3C2410, and is supported by the s3c2410_wdt driver.

MMC/SD/SDIO

No current support for the MMC/SD/SDIO block.

IIC

The IIC hardware is the same as the S3C2410, and is supported by the i2c-s3c24xx driver.

IIS

No current support for the IIS interface.

SPI

No current support for the SPI interfaces.

ATA

No current support for the on-board ATA block.

Document Author

Ben Dooks, Copyright 2006 Simtec Electronics

16.19.7 Simtec Electronics EB2410ITX (BAST)

http://www.simtec.co.uk/products/EB2410ITX/

Introduction

The EB2410ITX is a S3C2410 based development board with a variety of peripherals and expansion connectors. This board is also known by the shortened name of Bast.

Configuration

To set the default configuration, use make bast_defconfig which supports the commonly used features of this board.

Support

Official support information can be found on the Simtec Electronics website, at the product page http://www.simtec.co.uk/products/EB2410ITX/

Useful links:

- Resources Page http://www.simtec.co.uk/products/EB2410ITX/resources.html
- Board FAQ at http://www.simtec.co.uk/products/EB2410ITX/faq.html
- Bootloader info http://www.simtec.co.uk/products/SWABLE/resources.html and FAQ http://www.simtec.co.uk/products/SWABLE/faq.html

MTD

The NAND and NOR support has been merged from the linux-mtd project. Any problems, see http://www.linux-mtd.infradead.org/ for more information or up-to-date versions of linux-mtd.

IDE

Both onboard IDE ports are supported, however there is no support for changing speed of devices, PIO Mode 4 capable drives should be used.

Maintainers

This board is maintained by Simtec Electronics.

Copyright 2004 Ben Dooks, Simtec Electronics

16.19.8 S3C24XX NAND Support

Introduction

Small Page NAND

The driver uses a 512 byte (1 page) ECC code for this setup. The ECC code is not directly compatible with the default kernel ECC code, so the driver enforces its own OOB layout and ECC parameters

Large Page NAND

The driver is capable of handling NAND flash with a 2KiB page size, with support for hardware ECC generation and correction.

Unlike the 512byte page mode, the driver generates ECC data for each 256 byte block in an 2KiB page. This means that more than one error in a page can be rectified. It also means that the OOB layout remains the default kernel layout for these flashes.

Document Author

Ben Dooks, Copyright 2007 Simtec Electronics

16.19.9 Samsung/Meritech SMDK2440

Introduction

The SMDK2440 is a two part evaluation board for the Samsung S3C2440 processor. It includes support for LCD, SmartMedia, Audio, SD and 10MBit Ethernet, and expansion headers for various signals, including the camera and unused GPIO.

Configuration

To set the default configuration, use make smdk2440_defconfig which will configure the common features of this board, or use make s3c2410_config to include support for all s3c2410/s3c2440 machines

Support

Ben Dooks'SMDK2440 site at http://www.fluff.org/ben/smdk2440/ which includes linux based USB download tools.

Some of the h1940 patches that can be found from the H1940 project site at http://www.handhelds.org/projects/h1940.html can also be applied to this board.

Peripherals

There is no current support for any of the extra peripherals on the baseboard itself.

MTD

The NAND flash should be supported by the in kernel MTD NAND support, NOR flash will be added later.

Maintainers

This board is being maintained by Ben Dooks, for more info, see http://www.fluff.org/ben/smdk2440/

Many thanks to Dimitry Andric of TomTom for the loan of the SMDK2440, and to Simtec Electronics for allowing me time to work on this.

(c) 2004 Ben Dooks

16.19.10 S3C2413 ARM Linux Overview

Introduction

The S3C2413 is an extended version of the S3C2412, with an camera interface and mobile DDR memory support. See the S3C2412 support documentation for more information.

Camera Interface

This block is currently not supported.

Document Author

Ben Dooks, Copyright 2006 Simtec Electronics

16.19.11 S3C24XX ARM Linux Overview

Introduction

The Samsung S3C24XX range of ARM9 System-on-Chip CPUs are supported by the 's3c2410' architecture of ARM Linux. Currently the S3C2410, S3C2412, S3C2413, S3C2416, S3C2440, S3C2442, S3C2443 and S3C2450 devices are supported.

Support for the S3C2400 and S3C24A0 series was never completed and the corresponding code has been removed after a while. If someone wishes to revive this effort, partial support can be retrieved from earlier Linux versions.

The S3C2416 and S3C2450 devices are very similar and S3C2450 support is included under the arch/arm/mach-s3c2416 directory. Note, while core support for these SoCs is in, work on some of the extra peripherals and extra interrupts is still ongoing.

Configuration

A generic S3C2410 configuration is provided, and can be used as the default by make s3c2410_defconfig. This configuration has support for all the machines, and the commonly used features on them.

Certain machines may have their own default configurations as well, please check the machine specific documentation.

Layout

The core support files are located in the platform code contained in arch/arm/plat-s3c24xx with headers in include/asm-arm/plat-s3c24xx. This directory should be kept to items shared between the platform code (arch/arm/plat-s3c24xx) and the arch/arm/mach-s3c24* code.

Each cpu has a directory with the support files for it, and the machines that carry the device. For example S3C2410 is contained in arch/arm/mach-s3c2410 and S3C2440 in arch/arm/mach-s3c2440

Register, kernel and platform data definitions are held in the arch/arm/mach-s3c2410 directory./include/mach

arch/arm/plat-s3c24xx:

Files in here are either common to all the s3c24xx family, or are common to only some of them with names to indicate this status. The files that are not common to all are generally named with the initial cpu they support in the series to ensure a short name without any possibility of confusion with newer devices.

As an example, initially s3c244x would cover s3c2440 and s3c2442, but with the s3c2443 which does not share many of the same drivers in this directory, the name becomes invalid. We stick to s3c2440-<x> to indicate a driver that is s3c2440 and s3c2442 compatible.

This does mean that to find the status of any given SoC, a number of directories may need to be searched.

Machines

The currently supported machines are as follows:

Simtec Electronics EB2410ITX (BAST)

A general purpose development board, see EB2410ITX.txt for further details

Simtec Electronics IM2440D20 (Osiris)

CPU Module from Simtec Electronics, with a S3C2440A CPU, nand flash and a PCMCIA controller.

Samsung SMDK2410

Samsung's own development board, geared for PDA work.

Samsung/Aiji SMDK2412

The S3C2412 version of the SMDK2440.

Samsung/Aiji SMDK2413

The S3C2412 version of the SMDK2440.

Samsung/Meritech SMDK2440

The S3C2440 compatible version of the SMDK2440, which has the option of an S3C2440 or S3C2442 CPU module.

Thorcom VR1000

Custom embedded board

HP IPAQ 1940

Handheld (IPAQ), available in several varieties

HP iPAQ rx3715

S3C2440 based IPAQ, with a number of variations depending on features shipped.

Acer N30

A S3C2410 based PDA from Acer. There is a Wiki page at http://handhelds.org/moin/moin.cgi/AcerN30Documentation .

AML M5900

American Microsystems' M5900

Nex Vision Nexcoder Nex Vision Otom

Two machines by Nex Vision

Adding New Machines

The architecture has been designed to support as many machines as can be configured for it in one kernel build, and any future additions should keep this in mind before altering items outside of their own machine files.

Machine definitions should be kept in linux/arch/arm/mach-s3c2410, and there are a number of examples that can be looked at.

Read the kernel patch submission policies as well as the Documentation/arm directory before submitting patches. The ARM kernel series is managed by Russell King, and has a patch system located at http://www.arm.linux.org.uk/developer/patches/ as well as mailing lists that can be found from the same site.

As a courtesy, please notify <ben-linux@fluff.org> of any new machines or other modifications.

Any large scale modifications, or new drivers should be discussed on the ARM kernel mailing list (linux-arm-kernel) before being attempted. See http://www.arm.linux.org.uk/mailinglists/ for the mailing list information.

I2C

The hardware I2C core in the CPU is supported in single master mode, and can be configured via platform data.

RTC

Support for the onboard RTC unit, including alarm function.

This has recently been upgraded to use the new RTC core, and the module has been renamed to rtc-s3c to fit in with the new rtc naming scheme.

Watchdog

The onchip watchdog is available via the standard watchdog interface.

NAND

The current kernels now have support for the s3c2410 NAND controller. If there are any problems the latest linux-mtd code can be found from http://www.linux-mtd.infradead.org/

For more information see Documentation/arm/samsung-s3c24xx/nand.rst

SD/MMC

The SD/MMC hardware pre S3C2443 is supported in the current kernel, the driver is drivers/mmc/host/s3cmci.c and supports 1 and 4 bit SD or MMC cards.

The SDIO behaviour of this driver has not been fully tested. There is no current support for hardware SDIO interrupts.

Serial

The s3c2410 serial driver provides support for the internal serial ports. These devices appear as /dev/ttySAC0 through 3.

To create device nodes for these, use the following commands

> mknod ttySAC0 c 204 64 mknod ttySAC1 c 204 65 mknod ttySAC2 c 204 66

GPIO

The core contains support for manipulating the GPIO, see the documentation in GPIO.txt in the same directory as this file.

Newer kernels carry GPIOLIB, and support is being moved towards this with some of the older support in line to be removed.

As of v2.6.34, the move towards using gpiolib support is almost complete, and very little of the old calls are left.

See Documentation/arm/samsung-s3c24xx/gpio.rst for the S3C24XX specific support and Documentation/arm/samsung/gpio.rst for the core Samsung implementation.

Clock Management

The core provides the interface defined in the header file include/asm-arm/hardware/clock.h, to allow control over the various clock units

Suspend to RAM

For boards that provide support for suspend to RAM, the system can be placed into low power suspend.

See Suspend.txt for more information.

SPI

SPI drivers are available for both the in-built hardware (although there is no DMA support yet) and a generic GPIO based solution.

LEDs

There is support for GPIO based LEDs via a platform driver in the LED subsystem.

Platform Data

Whenever a device has platform specific data that is specified on a per-machine basis, care should be taken to ensure the following:

1) that default data is not left in the device to confuse the driver if a machine does not set it at startup

2) the data should (if possible) be marked as __initdata, to ensure that the data is thrown away if the machine is not the one currently in use.

   The best way of doing this is to make a function that kmalloc()s an area of memory, and copies the __initdata and then sets the relevant device's platform data. Making the function __init takes care of ensuring it is discarded with the rest of the initialisation code:

```
 static __init void s3c24xx_xxx_set_platdata(struct xxx_data␣
↪*pd)
 {
     struct s3c2410_xxx_mach_info *npd;

   npd = kmalloc(sizeof(struct s3c2410_xxx_mach_info), GFP_
↪KERNEL);
   if (npd) {
      memcpy(npd, pd, sizeof(struct s3c2410_xxx_mach_info));
      s3c_device_xxx.dev.platform_data = npd;
   } else {
        printk(KERN_ERR "no memory for xxx platform data\n");
   }
```

(continues on next page)

(continued from previous page)

```
}

Note, since the code is marked as __init, it should not be
exported outside arch/arm/mach-s3c2410/, or exported to
modules via EXPORT_SYMBOL() and related functions.
```

Port Contributors

Ben Dooks (BJD) Vincent Sanders Herbert Potzl Arnaud Patard (RTP) Roc Wu Klaus Fetscher Dimitry Andric Shannon Holland Guillaume Gourat (NexVision) Christer Weinigel (wingel) (Acer N30) Lucas Correia Villa Real (S3C2400 port)

Document Author

Ben Dooks, Copyright 2004-2006 Simtec Electronics

16.20 Frequently asked questions about the sunxi clock system

This document contains useful bits of information that people tend to ask about the sunxi clock system, as well as accompanying ASCII art when adequate.

Q: Why is the main 24MHz oscillator gatable? Wouldn't that break the system?

A: The 24MHz oscillator allows gating to save power. Indeed, if gated carelessly the system would stop functioning, but with the right steps, one can gate it and keep the system running. Consider this simplified suspend example:

While the system is operational, you would see something like:

```
24MHz         32kHz
 |
PLL1
 \
  \_ CPU Mux
      |
     [CPU]
```

When you are about to suspend, you switch the CPU Mux to the 32kHz oscillator:

```
  24Mhz         32kHz
   |              |
  PLL1            |
                  /
       CPU Mux  _/
         |
```

(continues on next page)

(continued from previous page)

```
       [CPU]

Finally you can gate the main oscillator::

                32kHz
                  |
                  |
                 /
       CPU Mux _/
         |
       [CPU]
```

Q: Were can I learn more about the sunxi clocks?

A: The linux-sunxi wiki contains a page documenting the clock registers, you can find it at

http://linux-sunxi.org/A10/CCM

The authoritative source for information at this time is the ccmu driver released by Allwinner, you can find it at

https://github.com/linux-sunxi/linux-sunxi/tree/sunxi-3.0/arch/arm/mach-sun4i/clock/ccmu

16.21 SPEAr ARM Linux Overview

16.21.1 Introduction

SPEAr (Structured Processor Enhanced Architecture). weblink : http://www.st.com/spear

The ST Microelectronics SPEAr range of ARM9/CortexA9 System-on-Chip CPUs are supported by the 'spear' platform of ARM Linux. Currently SPEAr1310, SPEAr1340, SPEAr300, SPEAr310, SPEAr320 and SPEAr600 SOCs are supported.

Hierarchy in SPEAr is as follows:

SPEAr (Platform)

- **SPEAr3XX (3XX SOC series, based on ARM9)**
  - **SPEAr300 (SOC)**
    - SPEAr300 Evaluation Board
  - **SPEAr310 (SOC)**
    - SPEAr310 Evaluation Board
  - **SPEAr320 (SOC)**
    - SPEAr320 Evaluation Board
- **SPEAr6XX (6XX SOC series, based on ARM9)**
  - **SPEAr600 (SOC)**

        * SPEAr600 Evaluation Board

- **SPEAr13XX (13XX SOC series, based on ARM CORTEXA9)**
  - **SPEAr1310 (SOC)**
    * SPEAr1310 Evaluation Board
  - **SPEAr1340 (SOC)**
    * SPEAr1340 Evaluation Board

16.21.2 Configuration

A generic configuration is provided for each machine, and can be used as the default by:

```
make spear13xx_defconfig
make spear3xx_defconfig
make spear6xx_defconfig
```

16.21.3 Layout

The common files for multiple machine families (SPEAr3xx, SPEAr6xx and SPEAr13xx) are located in the platform code contained in arch/arm/plat-spear with headers in plat/.

Each machine series have a directory with name arch/arm/mach-spear followed by series name. Like mach-spear3xx, mach-spear6xx and mach-spear13xx.

Common file for machines of spear3xx family is mach-spear3xx/spear3xx.c, for spear6xx is mach-spear6xx/spear6xx.c and for spear13xx family is mach-spear13xx/spear13xx.c. mach-spear* also contain soc/machine specific files, like spear1310.c, spear1340.c spear300.c, spear310.c, spear320.c and spear600.c. mach-spear* doesn't contains board specific files as they fully support Flattened Device Tree.

16.21.4 Document Author

Viresh Kumar <vireshk@kernel.org>, (c) 2010-2012 ST Microelectronics

16.22 STiH416 Overview

16.22.1 Introduction

The STiH416 is the next generation of HD, AVC set-top box processors for satellite, cable, terrestrial and IP-STB markets.

Features - ARM Cortex-A9 1.2 GHz dual core CPU - SATA2x2,USB 2.0x3, PCIe, Gbit Ethernet MACx2

16.23 STiH407 Overview

16.23.1 Introduction

The STiH407 is the new generation of SoC for Multi-HD, AVC set-top boxes and server/connected client application for satellite, cable, terrestrial and IP-STB markets.

Features - ARM Cortex-A9 1.5 GHz dual core CPU (28nm) - SATA2, USB 3.0, PCIe, Gbit Ethernet

16.23.2 Document Author

Maxime Coquelin <maxime.coquelin@st.com>, (c) 2014 ST Microelectronics

16.24 STiH418 Overview

16.24.1 Introduction

The STiH418 is the new generation of SoC for UHDp60 set-top boxes and server/connected client application for satellite, cable, terrestrial and IP-STB markets.

Features - ARM Cortex-A9 1.5 GHz quad core CPU (28nm) - SATA2, USB 3.0, PCIe, Gbit Ethernet - HEVC L5.1 Main 10 - VP9

16.24.2 Document Author

Maxime Coquelin <maxime.coquelin@st.com>, (c) 2015 ST Microelectronics

16.25 STi ARM Linux Overview

16.25.1 Introduction

The ST Microelectronics Multimedia and Application Processors range of CortexA9 System-on-Chip are supported by the 'STi' platform of ARM Linux. Currently STiH415, STiH416 SOCs are supported with both B2000 and B2020 Reference boards.

16.25.2 configuration

A generic configuration is provided for both STiH415/416, and can be used as the default by:

```
make stih41x_defconfig
```

16.25.3 Layout

All the files for multiple machine families (STiH415, STiH416, and STiG125) are located in the platform code contained in arch/arm/mach-sti

There is a generic board board-dt.c in the mach folder which support Flattened Device Tree, which means, It works with any compatible board with Device Trees.

16.25.4 Document Author

Srinivas Kandagatla <srinivas.kandagatla@st.com>, (c) 2013 ST Microelectronics

16.26 STiH415 Overview

16.26.1 Introduction

The STiH415 is the next generation of HD, AVC set-top box processors for satellite, cable, terrestrial and IP-STB markets.

Features:

- ARM Cortex-A9 1.0 GHz, dual-core CPU
- SATA2x2,USB 2.0x3, PCIe, Gbit Ethernet MACx2

16.27 Release notes for Linux Kernel VFP support code

Date: 20 May 2004

Author: Russell King

This is the first release of the Linux Kernel VFP support code. It provides support for the exceptions bounced from VFP hardware found on ARM926EJ-S.

This release has been validated against the SoftFloat-2b library by John R. Hauser using the TestFloat-2a test suite. Details of this library and test suite can be found at:

http://www.jhauser.us/arithmetic/SoftFloat.html

The operations which have been tested with this package are:

- fdiv
- fsub
- fadd
- fmul
- fcmp
- fcmpe
- fcvtd
- fcvts
- fsito
- ftosi
- fsqrt

All the above pass softfloat tests with the following exceptions:

- fadd/fsub shows some differences in the handling of +0 / -0 results when input operands differ in signs.
- the handling of underflow exceptions is slightly different. If a result underflows before rounding, but becomes a normalised number after rounding, we do not signal an underflow exception.

Other operations which have been tested by basic assembly-only tests are:

- fcpy
- fabs
- fneg
- ftoui
- ftosiz
- ftouiz

The combination operations have not been tested:

- fmac

- fnmac
- fmsc
- fnmsc
- fnmul