参考資料

TI Designs: TIDEP-0095
仮想化: AM572x上で動作するJailhouseハイパーバイザのリファレンス・デザイン

TEXAS INSTRUMENTS

概要

産業用組み込みシステムは、ベアメタルやリアルタイム・オペレーティング・システム(RTOS)をベースにした従来の定評あるリアルタイム・ソリューションと新たなニーズのバランスをとって、クラウド接続機能や先進的なグラフィカル・インターフェイスを追加しています。 Linux®は多くの場合、高度で安全なクラウド接続機能を提供し、先進的なヒューマン・マシン・インターフェイス(HMI)を実現する最も効率的な方法とされています。最新の組み込みプロセッサ(Sitara™ AM5728など)では、リアルタイム・アプリケーションとLinuxアプリケーションの機能を統合することができます。このリファレンス・デザインは、ARM® Cortex®-A15コアとJailhouseというオープン・ソースの静的ハイパーバイザを使用して、リアルタイム・アプリケーションとLinuxアプリケーションの共存をサポートしています。

リソース

TIDEP-0095	デザイン・フォルダ
AM5728	プロダクト・フォルダ
TMDXIDK5728	ツール・フォルダ
TMDSEVM572X	ツール・フォルダ
PROCESSOR-SDK-AM57X	ツール・フォルダ

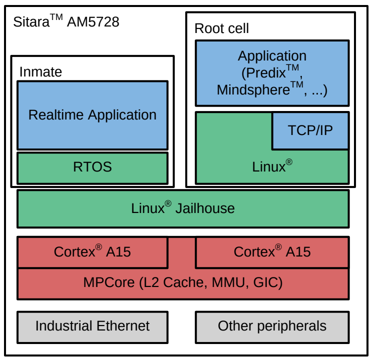
TI E2E™ Community　E2Eエキスパートに質問

特長

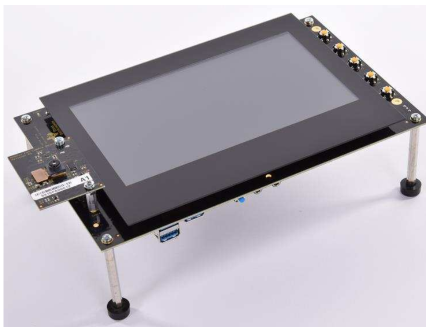
- Sitara AM572x上でJailhouseハイパーバイザが動作し、1つのARM Cortex-A15コアでLinux、もう1つのコアでベアメタルを実行
- Linux/ベアメタル間でのAM572xペリフェラルの静的パーティショニング
- 2つ目のコアでベアメタル・バイナリとRTOSベース・バイナリの実行をサポート
- Linuxでの処理負荷がある場合/ない場合の仮想ベアメタル・システムの性能(割り込みレイテンシ)測定
- TMDXIDK5728およびTMDSEVM5728ボードでテスト済み

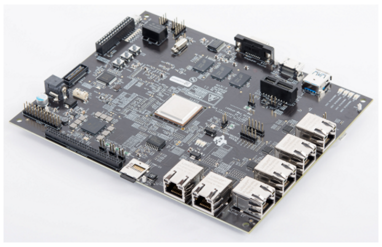
アプリケーション

- ファクトリ・オートメーション/制御—プログラマブル・ロジック・コントローラ(PLC)、DCS、PAC—CPU (PLCコントローラ)
- ファクトリオートメーション/制御—HMI—シングル・ボード・コンピュータ
- グリッド・インフラストラクチャ—保護リレー—マルチファンクション・リレー
- グリッド・インフラストラクチャ—グリッド・オートメーション—プロセス・バス
- グリッド・インフラストラクチャ—グリッド・オートメーション—システム・バス



Copyright © 2017, Texas Instruments Incorporated





1 System Description

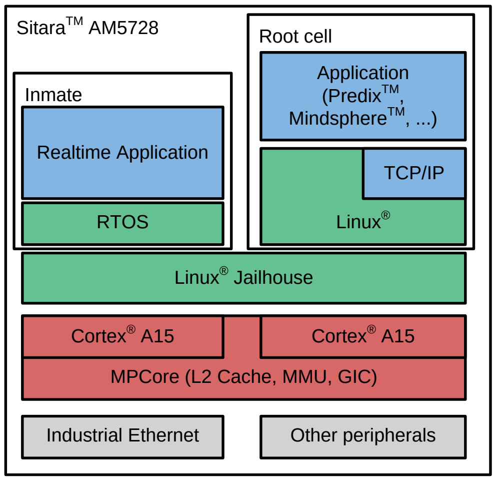
Embedded systems have different concerns and priorities from enterprise or data center server infrastructure or laptops. Typically the main difference is the priority of meeting the real-time requirement, as opposed to meeting a metric like average throughput or transactions per second. For embedded systems meeting the worst-case deadline is part of the correctness of the program, which is equally important to logical correctness. There is often no value in going beyond the required performance (for example, completing the task on average ahead of the deadline)or not meeting the required performance, which makes the system useless. In practice typical software (SW) architecture, such as RTOS or even a bare-metal-based periodic control loop triggered by either data input or a timer expiring, meets this requirement. Systems based on this pattern of SW architecture have been proven to work over the past several decades. Disrupting this established approach paradigm shifts, like Industry 4.0 and Industrial Internet of Things (IIoT), drive the requirement for a full-featured, cloud connectivity solution.

This solution has led many embedded systems to run Linux for the convenience of having a full-featured, networking stack with latest constantly updated security features and application-level protocols written in high-level languages like Python or Java. GE®’s Predix cloud platform and Siemens® Mindsphere® are examples of these protocols. However, for all the goodness of Linux, even with real-time (RT) patches, Linux might not satisfy the real-time constraints of toggling IO pins and counting nanoseconds. In these cases, being able to isolate a deterministic real-time OS or a bare-metal control loop alongside Linux is required. In some sense this is simpler than the traditional virtualization, which often has further goals beyond isolation, like consolidation of multiple-guest operating systems on to a single processor core or live migration of guests from a server or to another server. In embedded systems these use cases either do not exist or are of secondary value compared to meeting the real-time performance.

Using virtualization is not the only approach to design a system that combines a real-time application with a Linux application. A discrete solution with separate devices can sometimes make sense, but integrating the digital processing to one system-on-chip (SoC) brings size, communication latency, and often cost advantages. For some real-time applications using one or more of the embedded ARM Cortex-M4 or C66x DSP cores in the AM5728 SoC can be a more efficient solution than virtualizing a Cortex-A15 core. The right engineering tradeoff depends on the application, which includes the expected future evolution of additional features and capability. One alternative solution is moving the real-time application to RT Linux and using core affinity to assign the real-time threads to the second Cortex-A15 core while everything else is handled on the first core. The drawback with RT Linux is an order of magnitude higher worst case interrupt latency compared to an RTOS or a static hypervisor solution like Jailhouse.

2 System Overview

This reference design is a template products that require the integration of a real-time component built using an RTOS and a Linux component, such as cloud connectivity or a graphics interface. An example of an RTOS application is a fieldbus protocol stack, such as PROFINET™ or EtherCAT®. Linux Jailhouse manages boot and initialization of the system and peripherals using the two-stage hardware (HW) MMU isolates core 1 and the required peripherals for the inmate.



Copyright © 2017, Texas Instruments Incorporated

図 1. TIDEP-0095 Block Diagram

2.1 Multi-Core AM572x Processor

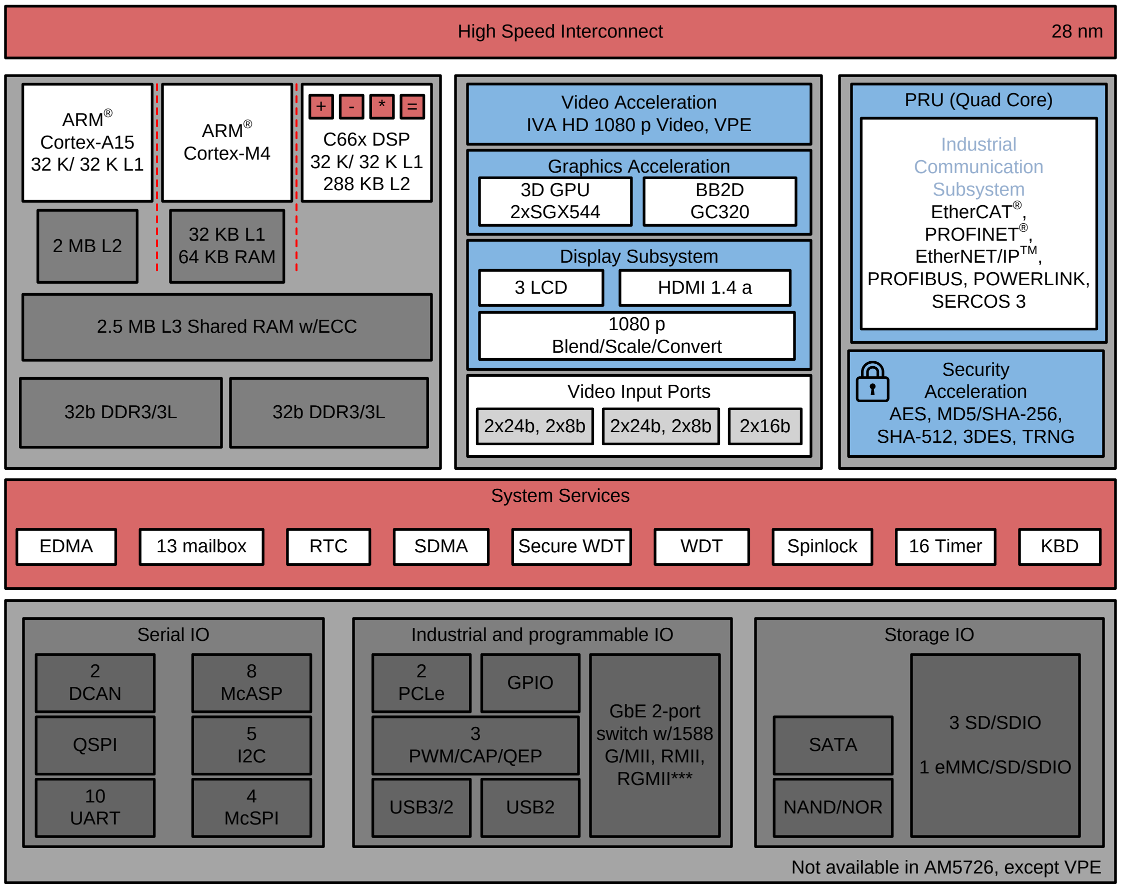

High Speed Interconnect
28 nm
ARM® Cortex-A15 32 K/ 32 K L1
ARM® Cortex-M4
+ - * =
C66x DSP 32 K/ 32 K L1 288 KB L2
2 MB L2
32 KB L1 64 KB RAM
2.5 MB L3 Shared RAM w/ECC
32b DDR3/3L
32b DDR3/3L
Video Acceleration IVA HD 1080 p Video, VPE
Graphics Acceleration
3D GPU 2xSGX544
BB2D GC320
Display Subsystem
3 LCD
HDMI 1.4 a
1080 p Blend/Scale/Convert
Video Input Ports
2x24b, 2x8b
2x24b, 2x8b
2x16b
PRU (Quad Core)
Industrial Communication Subsystem
EtherCAT®, PROFINET®, EtherNET/IP™, PROFIBUS, POWERLINK, SERCOS 3
Security Acceleration AES, MD5/SHA-256, SHA-512, 3DES, TRNG
System Services
EDMA
13 mailbox
RTC
SDMA
Secure WDT
WDT
Spinlock
16 Timer
KBD
Serial IO
2 DCAN
8 McASP
QSPI
5 I2C
10 UART
4 McSPI
Industrial and programmable IO
2 PCLe
GPIO
GbE 2-port switch w/1588 G/MII, RMII, RGMII***
3 PWM/CAP/QEP
USB3/2
USB2
Storage IO
SATA
3 SD/SDIO
1 eMMC/SD/SDIO
NAND/NOR
Not available in AM5726, except VPE

図 2. Sitara™ AM5728 Block Diagram

The AM572x is a high-performance, Sitara processor based on two ARM Cortex-A15s and two TI C66x DSPs. TheAM572x is designed for embedded applications including PLCs, industrial network switches, industrial gateways for protocol translation, HMI, grid infrastructure protection and communications, and other industrial use applications. The device includes the following subsystems:

- ARM Cortex-A15 microprocessor unit (MPU) subsystem, including two ARM Cortex-A15 cores
- Two DSP C66x cores
- Two Cortex-M4 subsystems, each including two ARM Cortex-M4 cores
- Two dual-core PRU-ICSS (capable of industrial protocols such as EtherCAT, PROFINET, EtherNet/IP, PROFIBUS™, Ethernet POWERLINK, SERCOS III, HSR, PRP)
- Graphics, video, real-time clock (RTC) and debug subsystems

The device supports memory management unit (MMU) and MPU:

- MMU used for key masters (Cortex-A15 MPU, Cortex-M4, C66x DSP, EDMA)
- Memory protection of C66x cores
- MMU inside the dynamic memory manager

The device also integrates:

- On-chip memory

External Memory Faces:

- Memory management
- Level 3 (L3) and level 4 (L4) interconnects
- System and serial peripherals

2.2 Design Considerations

Jailhouse is a static partitioning hypervisor based on Linux. Jailhouse can run bare-metal applications or (adapted) operating systems aside from Linux. For this purpose, Jailhouse configures CPU and device virtualization features of the hardware platform in a way that none of these domains, called *cells*, can interfere with each other in an unacceptable way. The overriding approach in Jailhouse is optimization for simplicity rather than feature richness. Unlike full-featured, Linux-based hypervisors like KVM or Xen, Jailhouse does not support overcommitment of resources like CPUs, RAM, or devices. All hardware resources, such as memory, CPUs, and peripherals, are statically assigned to guest operating system. Jailhouse hypervisor performs no scheduling and only virtualizes resources in software that are essential for a platform and cannot be partitioned in hardware. An example of shared hardware with ARM Cortex-A based SoC's is the interrupt controller or GIC.

Once Jailhouse is activated, the hypervisor runs bare-metal, that is, it takes full control over the hardware and requires no external support. However, in contrast to other bare-metal hypervisors, Jailhouse is loaded and configured by a normal Linux system with a management interface based on Linux infrastructure. Boot Linux first, next enable Jailhouse, and finally split off parts of the system's resources and assign them to additional cells. In the case of AM5728, it only makes sense to have two cells, the root cell Linux on A15 core 0 and another cell typically running an RTOS or bare-metal on A15 core 1. Typically a real-time inmate would directly own a couple peripherals (for example, the industrial Ethernet port implementing a fieldbus). The rest of the peripherals, such as USB or displays, would be owned and used by standard Linux drivers. Networking peripherals would typically be shared—at a minimum—with the configuration part of the paravirtualized driver, which sets up the hardware, so at least the real-time traffic goes directly to the inmate's memory and best-effort, TCP traffic and any management traffic goes to the Linux networking stack. Jailhouse consists of three parts: kernel module, hypervisor firmware, and tools. These are used to enable the hypervisor, create a cell, load an inmate binary, run it and stop it. On AM572x, which has two ARM Cortex-A15 cores, Linux boots in SMP mode using both cores. After enabled, the hypervisor moves Linux to the root-cell. At this point the root cell still uses both ARM cores. When a new cell is created, the hypervisor calls cpu_down() for the second core, which leaves Linux to run on core 0 only. The new cell will use core 1 and hardware resources dedicated for this cell in the cell configuration file.

Jailhouse is based on static partitioning of memory and peripherals. On AM5728 one core will run Linux, and the inmate will run an RTOS or bare metal, which is managed with device tree. Those peripherals and memory allocated for the inmate are carved out with additional configuration to the Linux device tree. 3.1.1 describes the example device tree used in this reference where 240MB of physical memory, one timer, and one serial port (UART) are statically partitioned for the inmate. A further 16MB of memory is carved out for the hypervisor itself—primarily used to store the second stage virtual address translation page tables. The .cell configuration files are compiled files that describe the memory and peripherals allocated to the root-cell and each inmate.

2.3 Highlighted Products

2.3.1 TMDXIDK5728

The TMDXIDK5728 is the Sitara AM5728 industrial development kit (IDK) The TMDXIDK5728 can be used for both of the examples in this reference design and any application development that requires the usage of the programmable real-time units for industrial communication (PRU-ICSS). If developing HMI applications the display, TMDXIDK57X-LCD is also required. The single, Cortex-A15 core based AM571x and AM570x devices and evaluation boards are not applicable as Jailhouse requires two or more ARM Cortex-A cores.

The AM572x IDK is a development platform for evaluating the industrial communication and control capabilities of Sitara AM572x processors for applications in factory automation, drives, robotics, grid infrastructure, and more. AM572x processors include dual PRU-ICSS subsystems, which can be used for industrial Ethernet protocols, such as PROFINET, EtherCAT, Ethernet/IP, and others. The TMDXIDK5728 breaks out six ports of Ethernet, four of which can be used concurrently—2x Gb Ethernet ports and 2x 10/100 Ethernet ports from the PRU-ICSS subsystems.

2.3.2 TMDSEVM572X

The TMDSEVM572X is the Sitara AM5728 general purpose evaluation module (GP-EVM). TMDSEVM572X can be used to run the UART and timer example application and for developing applications that do not require the use of the PRU-ICSS for industrial communication. The GP-EVM includes a display.

The AM572x EVM provides an affordable platform to quickly start evaluation of Sitara ARM Cortex-A15 AM57x processors (AM5728, AM5726, AM5718, AM5716) and accelerate development for HMI, machine vision, networking, medical imaging, and many other industrial applications. The EVM is a development platform based on the dual ARM Cortex-A15, dual C66x DSP processor that is integrated with tons of connectivity, such as PCIe, SATA, HDMI, USB 3.0 or 2.0, dual Gb Ethernet, and more.

The AM572x EVM also integrates video and 3D or 2D graphics acceleration as well as two PRU-ICSS and dual ARM Cortex-M4 cores.

2.4 System Design Theory

The peripherals will be owned by a inmate depending on the application. A good general principle is to keep the number of peripherals and amount of resources owned by the inmate as small as possible. This sizing is both to manage the complexity of meeting real-time and to minimize software engineering effort by using existing and maintained Linux drivers.

2.4.1 Example Device Tree

```
/* append to the existing device tree used with Linux, am57xx-evm-
reva3.dts is the device tree for the GP EVM */
#include "am57xx-evm-reva3.dts"
/* if you are using the IDK without a display use am572x-
idk.dts, if using an IDK with display use appropriate device tree file such as am572x-idk-lcd-
osd101t2045.dts */

/ {
        reserved-memory {
 /* reserve physical memory for the hypervisor privilege SW, 16MB (0x1000000) */
                jailhouse: jailhouse@ef000000 {
                        reg = <0x0 0xef000000 0x0 0x1000000>;
                        no-map;
                        status = "okay";
                };
 /* reserve physical memory for the inmate SW, 256MB (0xF000000) */
                jh_inmate: jh_inmate@ee000000 {
                        reg = <0x0 0xe0000000 0x0 0xf000000>;
                        no-map;
                        status = "okay";
                };
        };
};
/* assign timer8 to inmate, but allow Linux to configure clocking for it */
&timer8 {
        status = "disabled";
        ti,no-idle;
};
/* assign uart9 to inmate, but allow Linux to configure clocking for it */
&uart9 {
        status = "disabled";
        ti,no-idle;
};
/* configure the interrupt crossbar so the interrupts related to the above are not used by Linux.
Linux on AM5728 dynamically manages the interrupt crossbars, we want these to be statically
assigned to inmate */
/ {
        ocp {
                crossbar_mpu: crossbar@4a002a48 {
                ti,irqs-skip = <10 133 134 135 139 140>;
                };
        };
};
```

2.4.2 Sharing Peripherals

For some peripherals, such as Ethernet or storage, there is a requirement for both Linux and the real-time inmate to have access to the interface. To achieve this with Jailhouse, a paravirtualized driver is required. For example the real-time fieldbus, such as PROFINET, might be used for best-effort, Ethernet traffic coming from and going to the Linux networking stack. In this case the low-level control is typically in the real-time inmate with a paravirtualized Ethernet driver in the Linux side. In the case of storage, it might be more common to leverage the Linux driver and provide an interface for the inmate to read and write through a proxy application.

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware and Software

3.1.1 UART and Timer Application Example

This section contains details of setting up Jailhouse on Sitara AM5728 with a bare-metal inmate on Cortex A15 core 1 and with Linux running on core 0. Further information on setting up Jailhouse can be found in *TI Processors Wiki* . The demonstration application is *ti-app*, which is a simple, forever loop that configures a timer and related interrupt, printing out values read from the timer periodically on the console (UART).

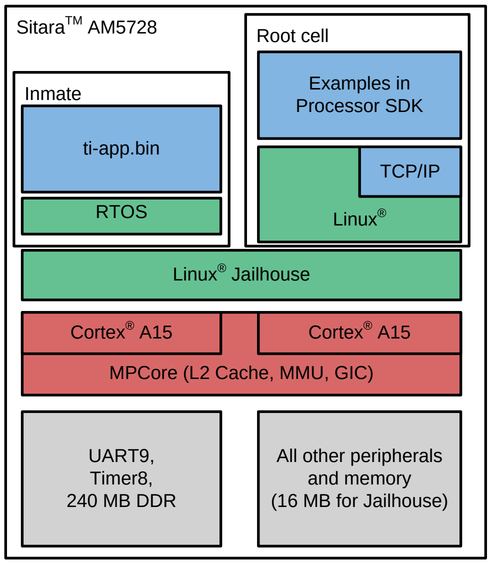

Sitara™ AM5728
Root cell
Examples in Processor SDK
Inmate
ti-app.bin
TCP/IP
RTOS
Linux®
Linux® Jailhouse
Cortex® A15
Cortex® A15
MPCore (L2 Cache, MMU, GIC)
UART9, Timer8, 240 MB DDR
All other peripherals and memory (16 MB for Jailhouse)

図 3. Bare Metal Application and Linux®

3.1.1.1 Hardware

This example runs on AM5728 EVM and AM5728 IDK and a host Windows or Linux machine with Ethernet and USB. Connect a USB to serial port cable to the host and connect both host and the EVMto the same LAN. There are a couple slight differences between the AM5728 EVM and AM5728 IDK with or without a display are highlighted in parenthesis.

3.1.1.2 Software

All the required software is in Processor SDK Linux 4.0.0 and can be run using the prebuilt SD card image. A Telnet and serial port console program, such as PuTTY, is required on the host. Source code for the application is located in */ti-processor-sdk-linux-am57xx-evm-04.00.00.04/board-support/extra-drivers/jailhouse-0.7/inmates/ti_app/*.

1. Connect IDK and host to the same Ethernet LAN.
2. Connect serial to USB cable to the host. Check which COM port becomes visible when USB cable is connected from the host (typically COM3). Open a console with 115200 baud rate 8-1-none
3. Boot the EVM with the Processor SDK Linux 4.0.0 prebuilt SD card image.
4. Halt the boot with a key press to the console get to u-boot shell.

5. Modify the boot arguments to use the correct device tree am572x-evm-jailhouse.dtb for the EVM (am572x-idk-jailhouse.dtb for the IDK without display, am572x-idk-jailhouse-lcd-osd101t2045.dtb, or am572x-idk-jailhouse-lcd-osd101t2587.dtb with display), and increase the contiguous virtual memory to for example 512M from the default 240M with environment variable vmalloc=512M. The device tree addition shown in 2.4.1 statically defines the memory used by jailhouse hypervisor and the inmate, and the interrupts and peripherals will be managed by the inmate. The contiguous memory is required to statically allocatargs_mmce the memory (16MB for hypervisor, 16MB for inmate) for the inmate and hypervisor, and in 32-bit plafroms the typical default value is not sufficient. For the SD card binary with Processor SDK 4.0.0 these modifications can be done modifying args_mmc by appending it with *setenv args_mmc ${args_mmc} vmalloc=512M*, and *setenv findfdt 'setenv fdtfile am572x-evm-jailhouse.dtb'*. Store the modifications with saveenv, and continue booting with the command boot.
6. Open a Telnet window to Linux on the EVM.
7. Next steps are to insert the kernel module, enable the hypervisor, create a cell for the inmate, load the bare metal binary, and start the binary. This is achieved with:

```
root@am57xx-evm:~# modprobe jailhouse
root@am57xx-evm:~# jailhouse enable /usr/share/jailhouse/examples/am57xx-evm.cell
root@am57xx-evm:~# jailhouse cell create /usr/share/jailhouse/examples/am57xx-evm-ti-app.cell
root@am57xx-evm:~# jailhouse cell load 1 /usr/share/jailhouse/examples/ti-app.bin
root@am57xx-evm:~# jailhouse cell start 1
```

The bare-metal example application is now running on core 1 and owns the UART, a timer, and related interrupts. Note the hypervisor sdebug messages will also print out on the serial port console in addition to the inmate. Linux continues to run utilizing cpu0 and has control over the rest of AM5728. This can be verified by launching any of the examples from the Matrix user interface, which will run in parallel to the inmate.

For example select *Video Analytics* and then *OpenCV+OpenCL+OpenGL* in Matrix to run an example application that utilizes the camera module, DSP offload, and the GPU. This and the other examples utilizing the Sitara AM5728 will continue to work with the ARM Cortex A15 core 1, a timer, and UART, which is carved out for the bare-metal inmate.

3.1.2 RTOS-Based Application Example

This section contains details of setting up jailhouse on Sitara AM5728 with an RTOS-based inmate built external to Processor SDK Linux on Cortex A15 core 1 and with Linux running on core 0. Further information on setting up *jailhouse* can be found in *Processor SDK Jailhouse Hypervisor* . The demonstration application is a TI RTOS-based application *pruss* loading the test application from Processor SDK RTOS to the programmable real-time units (PRUs). The PRU is a programmable building block inside the industrial communications subsystem (ICSS) that can be used to implement Ethernet or serial communication based fieldbusses (such as EtherCAT, PROFINET IRT, or PROFIBUS) or other functions requiring the lowest possible latency. This test is intended to show how a binary built outside Linux and utilizing an RTOS can be used. The test application loads simple firmware to the two PRUs, uses a timer, exchanges interrupts with the PRUs, and prints this out using the UART.

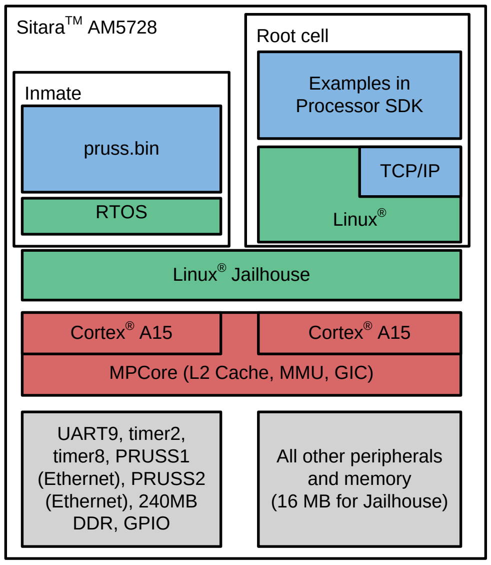

Sitara™ AM5728
Root cell
Inmate
Examples in Processor SDK
pruss.bin
TCP/IP
RTOS
Linux®
Linux® Jailhouse
Cortex® A15
Cortex® A15
MPCore (L2 Cache, MMU, GIC)
UART9, timer2, timer8, PRUSS1 (Ethernet), PRUSS2 (Ethernet), 240MB DDR, GPIO
All other peripherals and memory (16 MB for Jailhouse)

図 4. RTOS and Linux With Jailhouse Hypervisor

3.1.2.1 Hardware

This example runs on AM5728 IDK and a host Windows or Linux machine with Ethernet and USB. Connect a USB to serial port cable to the host and connect both host and the EVM to the same LAN.

3.1.2.2 Software

All the required software is in Processor SDK Linux 4.0.0 and can be run using the prebuilt SD card image. A Telnet and serial port console program, such as PuTTY, is required on the host.Processor SDK RTOS 4.0.0 is used as the RTOS, and the test application for PRUs is used as the test application. The example application source code is in */ti/pdk_am57xx_1_0_7/packages/ti/drv/pruss/test*. There are couple modifications needed to run this application as an inmate are limited to removing initialization related to UART, GIC, and the IO delays which are covered by Linux in the Jailhouse case. The source code for these modifications and makefile is in */ti/processor_sdk_rtos_am57xx_4_00_00_04/demos/jailhouse-inmate/rtos/pru-icss*.

注: Update the path name will change if using a newer Processor SDK release.

The TI RTOS configuration file, bios/pruss_arm_wSoCLib.cfg related, and board initialization file, src/idkAM572x_jh.c, have been modified to remove the initialization for the board already covered by Linux. For details compare to the stand alone RTOS version located in */ti/pdk_am57xx_1_0_7/packages/ti/drv/pruss/test/am572x/armv7/bios/*.

Build the pruss application to be used as inmate (pruss.bin is not included in the Processor SDK Linux):

1. Go to */processor_sdk_rtos_am57xx_4_00_00_04*.

   注: Update the path name will change if using a newer Processor SDK release.

2. Type: *source setupenv.sh* to set up environment variables related to compiling.
3. Go to */processor_sdk_rtos_am57xx_4_00_00_04/demos/jailhouse-inmate*.

   注: Update the path name will change if using a newer Processor SDK release.

4. Type: *source setenv.sh* to set up environment variables related to the *pruss* application.
5. Run: *make pruss_test*.
6. Copy *pruss.bin* to */usr/share/jailhouse/examples/* directory on the IDK

Run the pruss application:

1. Connect IDK and host to the same Ethernet LAN.
2. Connect serial to USB cable to the host. Check which COM port becomes visible when USB cable is connected from the host (typically COM3), and open a console with 115200 baud rate 8-1-none.
3. Boot the EVM with the Processor SDK 4.0.0 prebuilt SD card image.
4. Halt the boot with a key press to the console get to u-boot shell.
5. Modify the boot arguments to use the correct device tree am572x-idk-jailhouse.dtb for IDK without a display (am572x-idk-jailhouse-lcd-osd101t2045.dtb or am572x-idk-jailhouse-lcd-osd101t2587.dtb with display) and increase the contiguous virtual memory to for example 512M from the default 240M with environment variable vmalloc=512M. The device tree addition statically defines the memory used by jailhouse hypervisor and the inmate and which interrupts and peripherals will be managed by the inmate. The contiguous memory is required to statically allocate the memory (16MB for hypervisor, 16MB for inmate) for the inmate and hypervisor, and in 32-bit platforms, the typical default value is not sufficient. For the SD card binary with Processor SDK 4.0.0 these modifications can be done modifying args_mmc by appending it with *setenv args_mmc ${args_mmc} vmalloc=512M* and *setenv findfdt ‘setenv fdtfile am572x-idk-jailhouse.dtb’*. Store the modifications with saveenv, and continue booting with the command boot.
6. Open a Telnet window to Linux on the EVM.
7. Next steps are to insert the kernel module, enable the hypervisor, create a cell for the inmate, load the bare metal binary, and start the binary. The inmate will see intermediate physical addresses (IPA) as opposed to physical addresses. Depending on the RTOS used this might require some startup code to get running with Jailhouse. TI RTOS based binary expects to be loaded to address 0x80000000, which is the beginning of the DDR on AM572x while Jailhouse loads the program to address 0x0 in IPA address space. linux-loader.bin is a helper program that itself is loaded to address 0x0 and contains a branch to the entry point of the TI RTOS based image (0x80005128, passed to linux-loader.bin with a string at address 0x100). This is achieved with:

```
root@am57xx-evm:~# modprobe jailhouse
root@am57xx-evm:~# jailhouse enable /usr/share/jailhouse/examples/am57xx-evm.cell
root@am57xx-evm:~# jailhouse cell create /usr/share/jailhouse/examples/am572x-rtos-pruss.cell
root@am57xx-evm:~# jailhouse cell load 1 linux-loader.bin -a 0 -s "kernel=0x80005128" -
a 0x100 pruss.bin -a 0x80000000
root@am57xx-evm:~# jailhouse cell start 1
```

The RTOS example application pruss is now running on core 1 and will print out status messages on the UART. Linux continues to run using core 0 and the rest of AM5728. This can be verified by running Linux applications from the Telnet shell or with a display, by launching them from the Matrix GUI.

3.2 Testing and Results

3.2.1 Test Setup

To verify the real-time performance of Jailhouse Sitara AM5728 was setup to run Linux on one of the ARM Cortex A15 cores and a TI-RTOS inmate on the other A15 core. A test was run to measure interrupt latency. Poll mode driver based application performance of an inmate should be identical to a system without virtualization in a static partitioning system like Jailhouse. Anything interrupt-based is required to share the interrupt controller (GIC), which will introduce some interference from Linux to the real-time application. The measurements shown in 3.2.2 a million interrupts clearly shows the interference and captures the upper bound at 8.8 µs. For the first run of interrupt latency test an unloaded Linux running on core 0 is in the first column. In the second column Linux on core 0 is running STREAM. STREAM is an external memory access benchmark that fully uses the number of outstanding reads and writes to memory. The benchmark is scalable from individual processors to clusters supercomputers and is used at the processor level. STREAM was chosen as representative of a worst case memory access behavior of a Linux-based application on a Cortex A15, essentially with a memory access profile like an optimized memoryto-memory copy. In AM5728 the two Cortex-A15 cores share L2 cache and access to the rest of the SoC, which the STREAM benchmark running on core 0 stresses while core 1 access GIC registers to respond to the interrupt.

3.2.2 Test Results

3.2.2.1 Jailhouse Performance on Sitara™ AM5728

表 1. Interrupt Latency Of A Bare Metal Inmate (Core 1)

	UNLOADED LINUX ON CORE 0	LINUX RUNNING STREAM BENCHMARK ON CORE 0
Interrupt count Bucket 1.6 µs to 3.2 µs	99.3756%	33.9323%
Interrupt count Bucket 3.2 µs to 6.4 µs	0.6244%	66.0632%
Interrupt count Bucket 6.4 µs to 12.8 µs	none	0.0045%
Minimum interrupt latency	2.2 ms	1.8 ms
Maximum interrupt latency	5 ms	8.8 ms

4 Software Files

- Processor SDK Linux 4.00.00.04
  - Prebuilt SD card image can be used.
- Processor SDK RTOS 4.00.00.04
  - *pruss* example is used as an example inmate built outside Linux using TI RTOS

5 Related Documentation

1. *STREAM Benchmark*

5.1 商標

Sitara is a trademark of Texas Instruments.
ARM, Cortex are registered trademarks of ARM Limited.
EtherCAT is a registered trademark of Beckhoff Automation GmbH.
GE is a registered trademark of General Electric Compay.
Linux is a registered trademark of Linux Foundation.
PROFINET, PROFIBUS are trademarks of PROFIBUS and PROFINET International (PI).
Siemens, Mindsphere are registered trademarks of Siemens AG.
すべての商標および登録商標はそれぞれの所有者に帰属します。

6 Terminology

- GIC—generic interrupt controller
- GUI—graphical user interface
- MMU—memory management unit
- MPCore—multi processor core, a cluster of ARM Cortex A cores with a shared cache
- SoC—System on Chip

7 About the Author

PEKKA VARIS is the Chief Technologist for Catalog Processors Business in Texas Instruments.

VITALY ANDRIANOV is a Software Engineer in the the Processor SDK Linux team who has implemented Jailhouse support for the Sitara processors.

リビジョンAの改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

2017年**10**月発行のものから更新 **Page**