PRUSS Software Mitigation Guide for AM335x

The design materials referred to in this document are ***NOT SUPPORTED*** and DO NOT constitute a reference design. Only "community" support is allowed via resources at BeagleBoard.org/discuss [1].

Introduction

This article serves as a Software Migration Guide to assist in porting legacy software developed for the Programmable Real-Time Unit Subsystem (PRUSS) on AM18x to AM335x platforms. This guide will detail the PRU subsystem hardware differences and provide examples of software modifications required to ported PRU firmware and ARM code to AM335x.

The subsystem available on AM335x is the next-generation PRUSS (PRUSSv2). This PRUSS version preserves the same basic features and overall structure as the legacy PRUSS available on AM18x, allowing the PRU code developed on AM18x to be ported to AM335x.

Refer to the PRU wiki [2] for details about the PRUSS on AM18x and the AM335x Technical Reference Manual PRU-ICSS chapter addendum for details about the PRUSS on AM335x.

AM18x and AM335x Hardware Differences

This section provides an overview of the hardware differences between AM18x and AM335x. Both a high-level overview of the SoC-level hardware differences and a detailed overview of the PRU subsystems hardware differences are included.

SoC-level Hardware Differences

AM18x and AM335x devices support different peripherals and features. Table 1 compares the peripherals and features offered on AM18x and AM335x.

The SoC memory map, peripheral register map, pinmuxing, ARM interrupt controller events, and eDMA mapping also differ between the devices. Note other inherit differences may exist within the peripherals and features listed below. For additional details, refer to the AM18x to AM335x hardware migration guide [3] wiki and device-specific data sheets and user guides available at the device product pages:

- AM18x [4]
- AM335x [5]

Device Family	AM18x	AM335x
Device Family	AM1808/6/2 - ARM 9	AM3357/6/2 - CortexA8 AM3359/8/4 - CortexA8 with SGX 530
Package Options		
Packages	361-ball PBGA (ZCE), .65-mm Ball Pitch 361-ball PBGA (ZWT), .80-mm Ball Pitch	284-pin nFBGA (ZCE), .65-mm Ball Pitch with VCA 324-Pin nFBGA (ZCZ), .80-mm Ball Pitch Full Array
Co-processors and Subsystems		

ARM Processor	ARM 9 up to 450 MHz; 16KB Instruction and Data Caches	Cortex-A8 up to 720MHz; 32K-Byte Instruction and Data Caches; 256K-Byte L2 Cache w/ECC
	Supported CVdd: 1.0/1.1/1.2/1.3 V	
Neon Co-processor	not present	Y
SGX530 3D Graphics Engine	not present	Y
eDMA	Y	Y
PRUSS	Y	Y
Memory Interfaces:		
Memory Subsystem	mDDR/DDR2 Controller; EMIFA	EMIF; GPMC; ELM
Security		
Crypto hardware accelerators	not present	Y
Video Interfaces		
LCD Controller	Y	Y
VPIF	Y	not present
Peripherals		
USB	USB 1.1, USB 2.0	USB 2.0 [x2]
eMAC	10/100 Mbps	10/100/1000 Mbps
CAN	not present	2
McASP	1	2
McBSP	2	not present
UART	3 (none with IrDA)	6 (all with IrDA)
McSPI	2	2
I2C	2	3
GPIO	9 banks	4 banks
eCAP	3	3
eHRPWM	2	3
eQPE	not present	3
ADC/TS	not present	8ch 12bit
HPI	Y	not present
uPP	Y	not present
SATA Controller	Y	not present
Removable Media		
MMC/SD/SDIO	2	3
Power, Reset, and Clock Management		
RTC	Y	Y
Test Interfaces		
JTAG	Y	Y

ETM & ETB	Y	Y
IEEE 1500 support	not present	Y
Misc		
GP Timer	3 64b or 6 x32b Timers	7
Watchdog Timer	1	1

PRUSS Hardware Differences between AM18x and AM335x

The AM335x PRUSSv2 is based on the AM18x PRUSS. The basic subsystem hardware features are retained on AM335x, such as two PRU cores, instruction RAM, data RAM, interrupt controller, constant table, global and local accessibility.

Updates to the PRUSSv2 on AM335x include:

- Increased program memory (iRAM) from 4KB to 8KB
- Increased data memory (DRAM) from 512 bytes to 8KB
- Added 12KB shared RAM
- Expanded memory mapping
- Updated constant table entries
- Updated interrupt table events

New features added to the PRUSSv2 on AM335x are described in the AM335x Technical Reference Manual PRU-ICSS chapter addendum. Note this migration guide does not discuss these new features since there is no associated impact when porting legacy code.

Below shows a comparison block diagram of the subsystems:

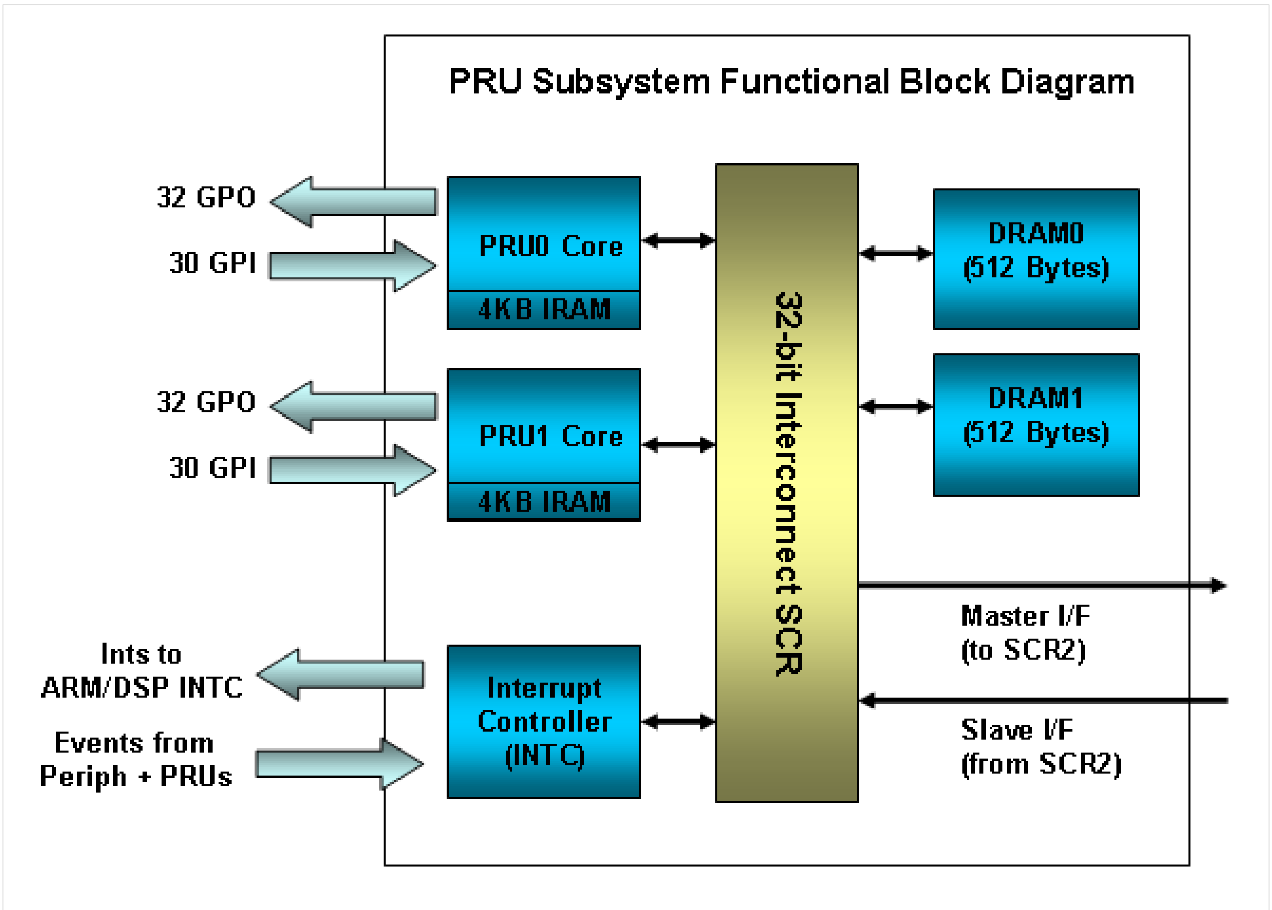

Figure 1. AM18x PRUSS block diagram

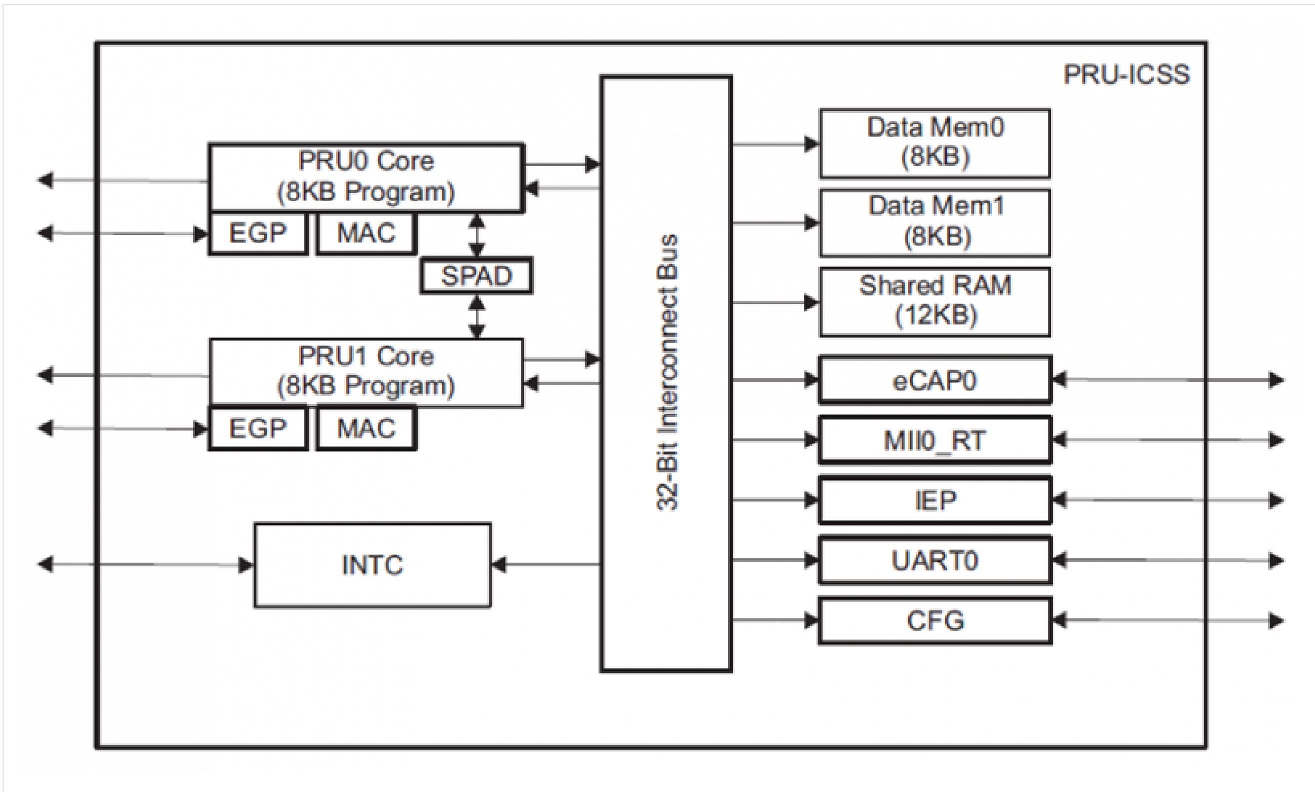

Figure 2. AM335x PRUSS block diagram

PRUSS Memory Map Comparison

The local and global memory maps are different on AM335x to accommodate a larger data RAM and instruction RAM, as well as new features.

Local Memory Map Differences

Table 2 shows the differences between the local memory map in AM18x and AM335x.

Table 2. Local Memory Map Comparison

Start Address	AM18x		AM335x	
	PRU0	PRU1	PRU0	PRU1
0x0000_0000	Data RAM 0	Data RAM 1	Data 8KB RAM 0	Data 8KB RAM 1
0x0000_0200	Reserved	Reserved		
0x0000_2000	Data RAM 1	Data RAM 0	Data 8KB RAM 1	Data 8KB RAM 0
0x0000_2200	Reserved	Reserved		
0x0000_4000	INTC Registers	INTC Registers		
0x0000_7000	PRU0 Registers	PRU0 Registers		
0x0000_7800	PRU1 Registers	PRU1 Registers		
0x0000_8000	Reserved	Reserved		

0x0001_0000	Reserved	Reserved	Shared Data 12KB RAM2	Shared Data 12KB RAM2
0x0002_0000			INTC	INTC
0x0002_2000			PRU0 Control Registers	PRU0 Control Registers
0x0002_2400			Reserved	Reserved
0x0002_4000			PRU1 Control Registers	PRU1 Control Registers
0x0002_4400			Reserved	Reserved
0x0002_6000			New Features	New Features
0x0008_0000				

Global Memory Map Differences

The global view of the PRUSS internal memories and control ports are documented in Table 3 and 4. This global memory map is a system-level mapping that allows other SoC resources to access the PRUSS memories.

The PRU0 and PRU1 cores can use either the local or global addresses to access their internal memories, but using the local addresses will provide access time several cycles faster than using the global addresses. This is because when accessing via the global address the access needs to be routed through the switch fabric outside PRUSS and back in through the PRUSS slave port.

AM18x and AM335x have different global memory maps. Table 3 compares the global start address for the PRU subsystem on AM18x and AM335x. The base addresses (listed as an offset of the global PRUSS start address) for each memory block in the PRUSS global memory map are compared in Table 4.

Table 3. Start Address Comparison

	AM18x	AM335x
Start Address	0x01C3_0000	0x4A30_0000

Table 4. Global Base Address Offset Comparison

Base Address Offset	AM18x	AM335x
0x0000_0000	Data RAM 0	Data 8KB RAM 0
0x0000_0200	Reserved	
0x0000_2000	Data RAM 1	Data 8KB RAM 1
0x0000_2200	Reserved	
0x0000_4000	INTC Registers	
0x0000_7000	PRU0 Registers	
0x0000_7800	PRU1 Registers	
0x0000_8000	PRU0 Instruction RAM	
0x0000_9000	Reserved	
0x0000_C000	PRU1 Instruction RAM	
0x0000_D000	Reserved	

0x0001_0000		Data 12KB RAM 2
0x0002_0000		INTC
0x0002_2000		PRU0 Control
0x0002_2400		PRU0 Debug
0x0002_4000		PRU1 Control
0x0002_4400		PRU1 Debug
0x0002_6000		New Features
0x0003_4000		PRU0 8KB IRAM
0x0003_8000		PRU1 8KB IRAM
0x0004_0000		Reserved

** The Base Address Offset values in Table 4 are offsets from the Start Address shown in Table 3.*

PRU<n> Register Content and Offsets

The PRU<n> Registers (Control and Debug) and offsets of each control and debug register within this area of memory are identical on both devices.

INTC Register Content and Offsets

The PRU INTC registers and offsets of each INTC register within this area of memory are identical on both devices.

Constants Table Differences

The PRUSS constant table entries are partially backwards compatible. However, as shown in Table 5, some entries have been replaced by new or more pertinent peripherals supported on AM335x.

Table 5. Constant Table Comparison

	AM18x		AM335x	
Entry #	**Region Pointed To**	**Value [31:0]**	**Region Pointed To**	**Value [31:0]**
0	PRU0/1 Local INTC	0x0000_4000	PRU0/1 Local INTC	0x0002_0000
1	Timer64P0	0x01C2_0000	DMTIMER2	0x4804_0000
2	I2C0	0x01C2_2000	I2C1	0x4802_A000
3	PRU0/1 Local Data	0x0000_0000	eCAP (local)	0x0003_0000
4	PRU1/0 Local Data	0x0000_2000	PRUSS CFG(local)	0x0002_6000
5	MMC/SD	0x01C4_0000	MMCHS 0	0x4806_0000
6	SPI0	0x01C4_1000	MCSPI 0	0x4803_0000
7	UART 0	0x01C4_2000	UART0 (local)	0x0002_8000
8	McASP0 DMA	0x01D0_2000	McASP0 DMA	0x4600_0000
9	*Reserved*	0x01D0_6000	GEMAC	0x4A10_0000
10	*Reserved*	0x01D0_A000	*Reserved*	0x4831_8000
11	UART1	0x01D0_C000	UART1	0x4802_2000
12	UART2	0x01D0_D000	UART2	0x4802_4000
13	USB0	0x01E0_0000	*Reserved*	0x4831_0000

14	USB1	0x01E2_5000	DCAN0	0x481C_C000
15	UHPI Config	0x01E1_0000	DCAN1	0x481D_0000
16	*Reserved*	0x01E1_2000	MCSPI 1	0x481A_0000
17	I2C1	0x01E2_8000	I2C2	0x4819_C000
18	EPWM0	0x01F0_0000	eHRPWM1/ eCAP1/ eQEP1	0x4830_0000
19	EPWM1	0x01F0_2000	eHRPWM2/ eCAP2/ePWM2	0x4830_2000
20	*Reserved*	0x01F0_4000	eHRPWM3/ eCAP3/ePWM3	0x4830_4000
21	ECAP0	0x01F0_6000	MDIO (local)	0x0003_2400
22	ECAP1	0x01F0_7000	Mailbox 0	0x480C_8000
23	ECAP2	0x01F0_8000	Spinlock	0x480C_A000
24	PRU0/1 Local Data	0x0000_0n00, n=c24_blk_index[3:0]	PRU0/1 Local Data	0x0000_0n00, n=c24_blk_index[3:0]
25	McASP0 Control	0x01D0_0n00, n=c25_blk_index[3:0]	PRU1/0 Local Data	0x0000_2n00, n=c25_blk_index[3:0]
26	*Reserved*	0x01D0_4000	IEP (local)	0x0002_En00, n=c26_blk_index[3:0]
27	*Reserved*	0x01D0_8000	MII_RT (local)	0x0003_2n00, n=c27_blk_index[3:0]
28	DSP Megamodule RAM/ROM	0x11nn_nn00, nnnn=c28_pointer[15:0]	Shared PRU RAM (local)	0x00nn_nn00, nnnn=c28_pointer[15:0]
29	EMIFA SDRAM	0x40nn_nn00, nnnn=c29_pointer[15:0]	TPCC	0x49nn_nn00, nnnn=c29_pointer[15:0]
30	Shared RAM	0x80nn_nn00, nnnn=c30_pointer[15:0]	L3 OCMC0	0x40nn_nn00, nnnn=c30_pointer[15:0]
31	mDDR/DDR2 Data	0xC0nn_nn00, nnnn=c31_pointer[15:0]	EMIF0 DDR Base	0x80nn_nn00, nnnn=c31_pointer[15:0]

PRU Module Interface Added Features

The functionality and structure of R30 and R31 is preserved on AM335x. AM335x supports several new GPI / GPO modes using R30 and R31. These modes are configured through the AM335x PRU-ICSS CFG register space. The direct connect GPI and GPO (default) mode on AM335x is equivalent to that on AM18x.

Note the R30 and R31 registers should be initialized before releasing the PRU from HALT. Also, to avoid driving uninitialized values on GP pins configured as OUTPUTs, the register write of this initialization must complete soon after reset, but before setting the SoC level Pin Mux to OUTPUT values from PRU.

Instruction Set and Format Compatibility

The instruction set and format on AM335x is backwards compatible with AM18x.

Note the SCAN instruction is not supported on AM335x.

Interrupt Controller Differences

The basic structure of the interrupt controller is the same in both devices. However, the events assigned to the PRUSS system event are different. The PRUSS-generated interrupts are also assigned to different event numbers in the ARM interrupt controller.

The INTC mapping of system events to channels to hosts is still the same. Both devices support the same number of total system events (64), channels (16), and hosts (10). On both AM18x and AM335x, Host0 and Host1 are connected to the PRU cores and Host2-9 are exported for signaling the ARM and eDMA.

Table 6 shows the differences between the system event numbers designated for external events generated by peripherals and events generated by writing to R31. Note on AM335x some system events are allocated for events from new modules in the PRUSS. Therefore, the number of available events generated by writing to R31 on AM335x is less than on AM18x.

Table 6. Event Mapping Structure

	AM18x Event Numbers	AM335x Event Numbers
Externally generated events	0 – 31	32 – 63
R31-generated events	32 - 63	16 – 31
PRUSS module generated events	N/A	0 - 15

Table 7.1 and Table 7.2 compare the system events in both devices. The events of peripherals that do not exist on AM335x are grayed out. The events of peripherals supported by AM335x but not include in the PRUSS INTC are listed as “not included.” Refer to the PRUSS wiki [2] and AM335x TRM PRU-ICSS chapter addendum for a complete list of each device's system events.

Table 7.1 INTC Event Comparison (Mode 0 on AM18x)

	AM18x		AM335x	
Function	**Event**	**Mode**	**Event**	**Mode**
Emulation Suspend Signal (Software Use Only)	0	0	58	0
ECAP0 Interrupt	1	0	42	0
ECAP1 Interrupt	2	0	35	0
Timer64P0 Event Out 12	3	0	not included	
ECAP2 Interrupt	4	0	36	0
McASP0 TX DMA Request	5	0	not included	
McASP0 RX DMA Request	6	0	not included	
McBSP0 TX DMA Request	7	0		
McBSP0 RX DMA Request	8	0		
McBSP1 TX DMA Request	9	0		
McBSP1 RX DMA Request	10	0		
SPI0 Interrupt 0	11	0	44	0
SPI1 Interrupt 0	12	0	not included	
UART0 Interrupt	13	0	51	0
UART1 Interrupt	14	0	32	0
I2C0 Interrupt	15	0	41	0
I2C1 Interrupt	16	0	not included	
UART2 Interrupt	17	0	52	0
MMCSD0 Interrupt 0	18	0	not included	
MMCSD0 Interrupt 1	19	0	not included	
USB0 (USB2.0 HS OTG) Subsystem Interrupt Request (aggregated from subsystem’s INTD)	20	0	not included	

USB1 (USB1.1 FS OHCI) Subsystem IRQ Interrupt	21	0	not included	
Timer64P0 Event Out 34	22	0	not included	
ECAP0 input (output from mux)	23	0	not included	
EPWM0 Interrupt	24	0	43	0
EPWM1 Interrupt	25	0	46	0
SATA Interrupt	26	0		
EDMA3_0_CC0_INT2 (region 2) ***	27	0	63	0
EDMA3_0_CC0_INT3 (region 3) ***	28	0	63	0
UHPI CPU_INT	29	0		
EPWM0TZ Interrupt or EPWM1TZ Interrupt	30	0	56	0
McASP0 TX Interrupt	31	1	55	0
McASP0 RX Interrupt			54	0

Table 7.2 INTC Event Comparison (Mode 1 on AM18x)

	AM18x		AM335x	
Function	**Event**	**Mode**	**Event**	**Mode**
Emulation Suspend Signal (Software Use Only)	0	1	58	0
Timer64P2_T12CMPEVT0	1	1	not included	
Timer64P2_T12CMPEVT1	2	1	not included	
Timer64P2_T12CMPEVT2	3	1	not included	
Timer64P2_T12CMPEVT3	4	1	not included	
Timer64P2_T12CMPEVT4	5	1	not included	
Timer64P2_T12CMPEVT5	6	1	not included	
Timer64P2_T12CMPEVT6	7	1	not included	
Timer64P2_T12CMPEVT7	8	1	not included	
Timer64P3_T12CMPEVT0	9	1	not included	
Timer64P3_T12CMPEVT1	10	1	not included	
Timer64P3_T12CMPEVT2	11	1	not included	
Timer64P3_T12CMPEVT3	12	1	not included	
Timer64P3_T12CMPEVT4	13	1	not included	
Timer64P3_T12CMPEVT5	14	1	not included	
Timer64P3_T12CMPEVT6	15	1	not included	
Timer64P3_T12CMPEVT7	16	1	not included	
Timer64P0_T12CMPEVT0 or Timer64P0_T12CMPEVT1 or Timer64P0_T12CMPEVT2 or Timer64P0_T12CMPEVT3 or Timer64P0_T12CMPEVT4 or Timer64P0_T12CMPEVT5 or Timer64P0_T12CMPEVT6 or Timer64P0_T12CMPEVT7	17	1	not included	
Timer64P2 Event Out 12	18	1	not included	
Timer64P3 Event Out 12	19	1	not included	
Timer64P1 Event Out 12	20	1	not included	
UART1 Interrupt	21	1	32	0

UART2 Interrupt	22	1	52	0
SPI0 Interrupt 0	23	1	44	0
EPWM0 Interrupt	24	1	43	0
EPWM1 Interrupt	25	1	46	0
SPI1 Interrupt 0	26	1	not included	
GPIO Bank 0 Interrupt	27	1	57	0
GPIO Bank 1 Interrupt	28	1	not included	
McBSP0 TX DMA Request	29	1		
McBSP0 RX DMA Request	30	1		
McASP0 TX Interrupt	31	1	55	0
McASP0 RX Interrupt			54	0

*** *The eDMA shadow regions accessible by the PRUSS has changed from region 2 & 3 on AM18x to region 1 on AM335x.*

The AINTC event numbers mapped to the PRUSS source interrupts have also been updated. Table 8 shows these changes.

Table 8. AINTC Mapping of Source Interrupt to Event Number Comparison

Source	AM18x Event Number	AM335x Event Number
PRUSS1_EVTOUT0	3	20
PRUSS1_EVTOUT1	4	21
PRUSS1_EVTOUT2	5	22
PRUSS1_EVTOUT3	6	23
PRUSS1_EVTOUT4	7	24
PRUSS1_EVTOUT5	8	25
PRUSS1_EVTOUT6	9	26
PRUSS1_EVTOUT7	10	27

On AM335x the pr1_host[7] is mapped to EDMA event 0 and pr1_host[6] is mapped to EDMA event 1.

Example Software Modifications

The software changes required to port legacy code from AM18x to AM335x are based on the hardware differences between the two devices. This section details the key differences in software and describes how legacy code can be modified for AM335x. Note additional modifications may be required relating to other SoC differences external to the PRUSS. Some of these modifications are discussed in the modifying software for SoC related differences section.

A checklist of changes required for both legacy PRU firmware and ARM code is provided below.

PRU Firmware Checklist

1	PRU constant table values
2	PRU addresses within local memory map
3	PRU addresses within global memory map
4	PRUSS interrupt system event numbers
5	SoC related changes (ie. peripheral addressing or registers, pinmux configuration, etc.)

ARM Code Checklist

1	PRU addresses within global memory map
2	PRUSS interrupt system event numbers
3	SoC related changes (ie. peripheral addressing or registers, pinmux configuration, AINTC, etc.)

Samples of code are shown throughout this section to demonstrate the changes described. Many of these code snippets are part of the PRU example code for the ARM. This source code can be obtained from the AM1808 SDK [6].

Updating PRU Constant Table References in Firmware

Differences in the PRU constant table will require changes to PRU firmware code. The PRU constant table entries are partially backwards compatible, as some peripherals and features maintain the same entry numbers. However, other peripherals and features have been removed, added, or remapped to different entry numbers in the AM335x table. Refer to Table 5 in the Constant Table section for a comparison between the constant tables on both devices.

Remapped Constant Table Entries

Some of the peripherals and features in the AM18x constant table are still present in the AM335x table but are mapped to a different entry number. Any reference of these peripherals by an LBCO or SBCO instruction in the firmware needs to be updated.

Below is an example of how to update the PRU_memAccessPRUDataRam example code for this change. The PRU Data RAM constant table entry changed from C3 to C24 between AM18x and AM335x, respectably. Note the constant table entries C24 – C31 are partially programmable. If the changes impact entries C24 – C31, confirm that the correct memory location is accessed. The Constant Table Block Index register, Constant Table Programmable Pointer registers (CTPPR_0, CTPPR_1), and LBCO and SBCO offsets may also need to be updated to point to the intended memory location.

AM18x: PRU_memAccessPRUDataRam	AM335x: PRU_memAccessPRUDataRam

#define CONST_PRUSSINTC C0
#define CONST_PRUDRAM C3
#define CONST_L3RAM C30
#define CONST_DDR C31
// Address for the Constant table Programmable Pointer Register 0(CTPPR_0)
#define CTPPR_0 0x7028
// Address for the Constant table Programmable Pointer Register 1(CTPPR_1)
#define CTPPR_1 0x702C
//Load 4 bytes from memory location c3(PRU0/1 Local Data)+4 into r4 using constant table
LBCO r4, CONST_PRUDRAM, 4, 4
// Add r3 and r4
ADD r3, r3, r4
//Store result in into memory location c3(PRU0/1 Local Data)+8 using constant table
SBCO r3, CONST_PRUDRAM, 8, 4
	#define CONST_PRUSSINTC C0
	#define CONST_PRUDRAM **C24**
	#define CONST_L3RAM C30
	#define CONST_DDR C31
	// Address for the Constant table Programmable Pointer Register 0(CTPPR_0)
	#define CTPPR_0 **0x00022028**
	// Address for the Constant table Programmable Pointer Register 1(CTPPR_1)
	#define CTPPR_1 **0x0002202C**
	// Address for the Constant table Block Index Register 0(CTBIR_0)
	#define CTBIR_0 0x00022020
	// Address for the Constant table Block Index Register 1(CTBIR_1)
	#define CTBIR_1 0x00022024
	// Configure the programmable pointer register for PRU0 by setting
	// c24_pointer[15:0] field to 0x00. This will make C24 point to
	// 0x00000000 (PRU DRAM).
	MOV r0, 0x00000000
	MOV r1, CTBIR_1
	ST32 r0, r1
	//Load 4 bytes from memory location c3(PRU0/1 Local Data)+4 into r4 using constant table
	LBCO r4, CONST_PRUDRAM, 4, 4
	// Add r3 and r4
	ADD r3, r3, r4
	//Store result in into memory location c3(PRU0/1 Local Data)+8 using constant table
	SBCO r3, CONST_PRUDRAM, 8, 4

Removed Constant Table Entries

Some entries from the AM18x constant table have been removed and replaced by more pertinent peripherals supported on AM335x. If a previously used entry no longer exists in the constant table, the PRU firmware will need to be updated to replace the corresponding byte burst with constant table offset instruction (SBCO or LBCO) to a standard byte burst instruction (SBBO or LBBO). This update requires three steps:

1. Prior to accessing this memory region, the base address needs to be first loaded into a PRU register.
2. The standard byte burst instruction (ie. SBBO or LBBO) will replace the existing byte burst instruction with constant table offset (ie. SBCO or LBCO).
3. Replace the constant register with the PRU register from step 1.

Below shows an example of how to update AM18x code that accesses USB0 memory. Note USB is not included in the AM335x constant table.

AM18x: Removed constant table entry	AM335x: Removed constant table entry

#define CONST_USB0 C13
// Load value into register
LDI r0.w0, 0x0001
// Read value from USB0
LBCO r1, CONST_USB0, 0, 4
// Write value to USB0
SBCO r0, CONST_USB0, 0, 4
	// Address for USB0
	#define USB0 0x47400000
	// Load value into register
	LDI r0.w0, 0x0001
	MOV r10, USB0
	// Read value from USB
	LBBO r1, **r10**, 0, 4
	// Write value to USB
	SBBO r0, **r10**, 0, 4

Updating Local Memory Map References in Firmware

Differences in the local PRUSS memory map require modifications to the PRU firmware. The local PRUSS memory map for AM335x contains the same modules, or blocks of memory, as AM18x. However, the module base addresses for data RAM, INTC, and PRU0/1 registers are different between devices, as shown in Table 2 of the Local Memory Map section. Note that within each module, the defined registers and offsets are the same.

The legacy PRU firmware will access the registers in local memory map by either loading the address (or defined variable name set to the address) into a register or by using the constant table entries for the PRU Data RAM and INTC registers. No change related to the local memory map of the INTC registers is required if the constant table is used. However, any references to the PRU Data RAM constant table entry need to be updated (refer to the Remapped Constant Table Entries section for any required constant table changes). If the constant table is not used, the base address or specific memory address may need to be updated, depending on how the legacy firmware is coded.

Common firmware code updates related to the local memory map module addresses include:

- Data RAM1/0

  * If byte burst instructions (LBBO, SBBO) are used, no changes are required for Data RAM0/1.

- Interrupt Controller
- PRU Control Register

  * The primary registers that the PRU firmware will access are related to the programmable constant table entries (ie. PRU Constant Table Block Index Register, PRU Constant Table Programmable Pointer Register 0, PRU Constant Table Programmable Pointer Register 1).

In the example code below, the base address of the PRU INTC is updated. The firmware divides the address into a base address + offset.

AM18x: Updating INTC base address	AM335x: Updating INTC base address

#define GER_OFFSET 0x10
#define INTC_REGS_BASE 0x00004000
// Global enable of all host interrupts
LDI regVal.w0, 0x0001
SBBO regVal, INTC_REGS_BASE, GER_OFFSET, 2
	#define GER_OFFSET 0x10
	#define INTC_REGS_BASE **0x00020000**
	// Global enable of all host interrupts
	LDI regVal.w0, 0x0001
	SBBO regVal, INTC_REGS_BASE, GER_OFFSET, 2

The example below shows modifying the CTPPR_1 of the PRU Control Register. In this case, the firmware stores the entire address to the CTPPR_1 constant, rather than dividing it into base address + offset.

AM18x: Updating PRU Control Register address	Am335x: Updating PRU Control Register address

// Address for the Constant table
// Programmable Pointer Register 1(CTPPR_1)
#define CTPPR_1 0x702C
// To access the DDR memory, the
// programmable pointer register is
// configured by setting C31[15:0] field. Set R0
// to zero and store that value into in CTPR_1
// to configure c31_pointer[15:0]
MOV r0, 0x00000000
MOV r1, CTPPR_1
SBBO r0, r1, 0, 4
	// Address for the Constant table
	// Programmable Pointer Register 1(CTPPR_1)
	#define CTPPR_1 **0x0002402C**
	// To access the DDR memory, the
	// programmable pointer register is
	// configured by setting C31[15:0] field. Set R0
	// to zero and store that value into in CTPR_1
	// to configure c31_pointer[15:0]
	MOV r0, 0x00000000
	MOV r1, CTPPR_1
	SBBO r0, r1, 0, 4

Updating Global Memory Map References

The changes to the global PRUSS memory map require any ARM code interacting with the PRU and any PRU assembly code that accesses the global (rather than local) memory map to be updated.

There are two changes relating to the global PRUSS memory map between devices. First, the start address for the PRUSS memory block differs between the devices' global memory maps, as shown in Table 2. Second, within the PRUSS memory block, the base addresses of each module (Data RAM, INTC, PRU0/1 registers, etc.) differ, as shown in Table 3. The registers and offsets within each module remain the same.

Examples for modifying both PRU firmware and ARM code are provided in the following sections.

Enable AM335x PRUSS to access global memory addresses

The AM335x PRUSS requires a configuration step to access global memory addresses that was not required by the legacy PRUSS on AM18x.

By default, the AM335x OCP master port is in standby and needs to be enabled in the PRUSS CFG register space, SYSCFG[STANDBY_INIT]. This can be done either by the PRU firmware or by the ARM before loading and enabling the PRU.

AM335x: PRU firmware enabling global memory access	AM335x: ARM code enabling global memory access

#define CONST_PRUCFG C4
// Enable OCP master port:
// clear SYSCFG[STANDBY_INIT] to enable OCP master port
LBCO r0, CONST_PRUCFG, 4, 4
CLR r0, r0, 4
SBCO r0, CONST_PRUCFG, 4, 4
	#define PRUSS0_CFG 5
	static void *cfgMem;
	static unsigned int *cfgMem_int;
	// within main(), before prussdrv_exec_program
	prussdrv_map_peripheral_io (PRUSS0_CFG, &cfgMem);
	cfgMem_int = (unsigned int*) cfgMem;
	cfgMem_int[1] &= 0xFFFFFFEF;

PRU firmware modifications

Most PRU firmware code should use the local memory map to reduce latencies. However, the PRU also has access to the global memory map. If the firmware code does access the global memory map, these addresses are required to be updated.

Below is a simple example of modifying the PRU firmware with the updated PRU global addresses.

AM18x: Firmware accessing global memory map	AM335x: Firmware accessing global memory map

#define PRU0_BASE_REG 0x01C37000
#define CTPPR_1_OFFSET 0x2C
MOV r0, 0x00000000
MOV r1, PRU0_CTR_REG_BASE
SBBO r0, r1, CTPPR_1_OFFSET, 4
	#define PRU0_BASE_REG **0x4A300000**
	#define CTPPR_1_OFFSET 0x2C
	MOV r0, 0x00000000
	MOV r1, PRU0_CTR_REG_BASE
	SBBO r0, r1, CTPPR_1_OFFSET, 4

ARM code modifications

ARM code interacting with the PRU uses the PRUSS global memory map and will require updates to the PRU defined addresses.

The example code below shows updates to a header file used both by kernel and application code.

AM18x: ARM code accessing global memory map	AM335x: ARM code accessing global memory map
// PRU Memory Macros #define PRU0_DATA_RAM_START (0x01C30000) #define PRU0_PROG_RAM_START (0x01C38000) #define PRU1_DATA_RAM_START (0x01C32000) #define PRU1_PROG_RAM_START (0x01C3C000) #define PRU_DATA_RAM_SIZE (0x200) #define PRU_PROG_RAM_SIZE (0x1000) #define PRU_PRU0_BASE_ADDRESS 0 #define PRU_INTC_BASE_ADDRESS (PRU_PRU0_BASE_ADDRESS + 0x4000) #define PRU_INTC_REVID (PRU_INTC_BASE_ADDRESS + 0) #define PRU_INTC_CONTROL (PRU_INTC_BASE_ADDRESS + 0x4) #define PRU_INTC_GLBLEN (PRU_INTC_BASE_ADDRESS + 0x10)	// PRU Memory Macros #define PRU0_DATA_RAM_START (**0x4A300000**) #define PRU0_PROG_RAM_START (**0x4A334000**) #define PRU1_DATA_RAM_START (**0x4A302000**) #define PRU1_PROG_RAM_START (**0x4A338000**) #define PRU_DATA_RAM_SIZE (0x1F40) #define PRU_PROG_RAM_SIZE (0x1F40) #define PRU_PRU0_BASE_ADDRESS 0 #define PRU_INTC_BASE_ADDRESS (PRU_PRU0_BASE_ADDRESS + 0x20000) #define PRU_INTC_REVID (PRU_INTC_BASE_ADDRESS + 0) #define PRU_INTC_CONTROL (PRU_INTC_BASE_ADDRESS + 0x4) #define PRU_INTC_GLBLEN (PRU_INTC_BASE_ADDRESS + 0x10)

For application code, if the PRUSS application loader API call prussdrv_map_prumem( ) is used to map the PRU data RAM to a pointer, then no changes are required for this section of code.

Application code manually mapping PRUSS addresses to a pointer will need to update the global PRU address. The example below shows updating the address for a register in the PRU debug registers.

AM18x: PRU_gpioToggle.c	AM335x: PRU_gpioToggle.c

/* map the memory */
mem_pru0reg = mmap(0, 0x00000FFF, PROT_WRITE | PROT_READ, MAP_SHARED, mem_fd, 0x01C37000);
mem_pruReg30 = mem_pru0reg + 0x00000478;
	/* map the memory */
	mem_pru0reg = mmap(0, 0x00000FFF, PROT_WRITE | PROT_READ, MAP_SHARED, mem_fd, **0x04A22000**);
	mem_pruReg30 = mem_pru0reg + 0x00000478;

Updating PRU system events and PRUSS INTC mapping

The AM18x and AM335x INTC have different system events. The system events will need to be updated both in the PRU firmware code and in the interrupt controller mapping in the ARM code (ie. using the prussdrv_pruintc_init API call from the PRU application loader, refer to section 4.6). Note the INTC mapping will not change, only the system event numbers.

The ARM interrupt map event numbers corresponding to PRUSS interrupts have also changed. In the ARM code, the user needs to confirm that IRQs are updated for the new PRUSS event numbers.

External events generated by peripherals

Updating event numbers

In both the PRU firmware and PRU INTC mapping in ARM code, the event numbers for peripheral-generated events should be updated according to Table 7.

If an interrupt used on AM18x is no longer supported on AM335x, one option is to poll or periodically read the peripheral’s status register, if it exists. Refer to the AM335x TRM for peripheral details.

The AM18x INTC event table maps both McASP tx and rx interrupts to one system event. However, AM335x splits McASP interrupts into separate tx and rx system events. To minimize changes to the code structure, any reference to the McASP system event could be split into two duplicate instructions—one processes the tx event and the other processes the rx event.

Note the PRU INTC has two modes or PRUSSEVTSEL options. A mode change between the AM18x and AM335x code may require an additional modification—- either removing code that sets the mode or if the new mode is not the default mode on AM335x, setting the mode through the PRUSS CFG register space on AM335x.

External and internal event numbering swapped

On AM18x external events generated by peripherals are mapped to system events 0-31, while AM335x maps these events to 32-63.

Some of the INTC functions are split into two registers, one for events 0-31 and another for events 32-63. This change in system event numbers means this registers must be switched in the PRU/ARM code. The effected registers are listed below:

SRSR1	(0x200)	->	SRSR2	(0x204)
SECR1	(0x280)	->	SECR2	(0x284)
ESR1	(0x300)	->	ESR2	(0x304)
ECR1	(0x380)	->	ECR2	(0x384)
SIPR1	(0xD00)	->	SIPR2	(0xD04)
SITR1	(0xD80)	->	SITR2	(0xD84)

AM18x: Processing External Interrupt	AM335x: Processing External Interrupt

#define hostEventStatus r31
#define HOST_0_BIT 30
#define MCASP_TXRX_EVENT 31
#define SICR_OFFSET 0x24
#define SRSR1_OFFSET 0x200
#define SRSR2_OFFSET 0x204
WBS hostEventStatus, HOST_0_BIT
// Read the PRUINTC register to know if the
// event is from McASP. If yes, then branch
MOV r2, SRSR1_OFFSET
LBCO r1, CONST_PRUSSINTC, r2, 4
QBBS MCASP_EVENT, r1, MCASP_TXRX_EVENT
MCASP_EVENT:
	#define hostEventStatus r31
	#define HOST_0_BIT 30
	#define MCASP_TX_EVENT **33**
	#define MCASP_RX_EVENT **34**
	#define SICR_OFFSET 0x24
	#define SRSR1_OFFSET 0x200
	#define SRSR2_OFFSET 0x204
	WBS hostEventStatus, HOST_0_BIT
	// Read the PRUINTC register to know if the
	// event is from McASP. If yes, then branch
	MOV r2, **SRSR2_OFFSET**
	LBCO r1, CONST_PRUSSINTC, r2, 4
	SUB r10, **MCASP_TX_EVENT**, #32
	QBBS MCASP_EVENT, r1, r10
	SUB r10, **MCASP_RX_EVENT**, #32
	QBBS MCASP_EVENT, r1, r10
	MCASP_EVENT:

Events generated by writing to PRU R31

Updating event numbers

In both the PRU firmware and PRU INTC mapping in ARM code, the event numbers for peripheral-generated events should be updated according to Table 7.

Note the event numbers of the R31-generated events are different between devices. AM335x also has fewer of these events than AM18x (16 vs. 32).

Internal and external event numbering swapped

On AM18x external events generated by peripherals are mapped to system events 0-31, while AM335x maps these events to 32-63.

The simplest modification for this difference is to update the internal system event number by subtracting 32 from the original event number (ie. 32 -> 0, 33 -> 1, ect.).

Some of the INTC functions are split into two registers, one for events 0-31 and another for events 32-63. This change in system event numbers means this registers must be switched in the PRU/ARM code. If direct swap of interrupt numbers done (ie. 32 -> 0, 33 -> 1, ect.), then the only change is to swap which register is being read. The effected registers are listed below:

SRSR2	(0x204)	->	SRSR1	(0x200)
SECR2	(0x284)	->	SECR1	(0x280)
ESR2	(0x304)	->	ESR1	(0x300)
ECR2	(0x384)	->	ECR1	(0x380)
SIPR2	(0xD04)	->	SIPR1	(0xD00)
SITR2	(0xD84)	->	SITR1	(0xD80)

AM18x: Processing R31-generated Interrupt	AM335x: Processing R31-generated Interrupt

#define hostEventStatus r31
#define HOST_0_BIT 30
#define ARM_TO_PRU0_EVENT 34
#define SICR_OFFSET 0x24
#define SRSR1_OFFSET 0x200
#define SRSR2_OFFSET 0x204
WBS hostEventStatus, HOST_0_BIT
// Read the PRUINTC register to know if the
// event is from the ARM. If yes, then clear
MOV r2, SRSR2_OFFSET
LBCO r1, CONST_PRUSSINTC, r2, 4
QBBS CLEAR, r1, 3
CLEAR:
MOV r1, ARM_TO_PRU0_EVENT
SBCO r1, CONST_PRUSSINTC, SICR_OFFSET, 4
	#define hostEventStatus r31
	#define HOST_0_BIT 30
	#define ARM_TO_PRU0_EVENT **18**
	#define SICR_OFFSET 0x24
	#define SRSR1_OFFSET 0x200
	#define SRSR2_OFFSET 0x204
	WBS hostEventStatus, HOST_0_BIT
	// Read the PRUINTC register to know if the
	// event is from the ARM. If yes, then clear
	MOV r2, **SRSR1_OFFSET**
	LBCO r1, CONST_PRUSSINTC, r2, 4
	QBBS CLEAR, r1, 3
	CLEAR:
	MOV r1, ARM_TO_PRU0_EVENT
	SBCO r1, CONST_PRUSSINTC, SICR_OFFSET, 4

Modifying software for SoC related differences

AM18x and AM335x devices have additional differences that also require changes in both PRU firmware and ARM code. Below is a list of some key differences that require code updates. However, this is not an exhaustive list, and the AM18x to AM335x software migration guide [3] should be referenced for more details.

Key differences between AM18x and AM335x devices require PRU legacy code updates include:

1. Global device memory map
   a. Start addresses of peripherals and features
   b. Base addresses of modules
   c. Register addresses and offsets
2. Peripherals
   a. Refer to Table 1 in section 2 for peripherals supported on each device
   b. Peripherals may have new memory or register maps. The functionality of registers may also change.
      * Note in PRU firmware, register map changes may affect offsets in LBBO, SBBO, LBCO, SBCO when accessing peripheral registers.
3. Pinmuxing

Frequently Asked Questions

1. Why does my AM335x PRU firmware hangs when reading or writing to an address external to the PRU Subsystem?

    The OCP master port is in standby and needs to be enabled in the PRUSS CFG register space, SYSCFG[STANDBY_INIT].

References

[1] http://BeagleBoard.org/discuss
[2] http://processors.wiki.ti.com/index.php/Programmable_Realtime_Unit_Subsystem
[3] http://processors.wiki.ti.com/index.php/AM18x_To_AM335x_Hardware_Migration_Guide
[4] http://focus.ti.com/docs/prod/folders/print/am1808.html
[5] http://focus.ti.com/docs/prod/folders/print/am3359.html
[6] http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sdk/AM1x/latest/index_FDS.html

Article Sources and Contributors

PRUSS Software Mitigation Guide for AM335x *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=134217 *Contributors*: M-watkins

Image Sources, Licenses and Contributors

Image:Legacy_PRUSS.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:Legacy_PRUSS.jpg *License*: unknown *Contributors*: M-watkins
Image:PRUSSv2.jpg *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:PRUSSv2.jpg *License*: unknown *Contributors*: M-watkins