Programmable Real-time Unit and Industrial Communication Sub-System (PRU-ICSS) Overview

The hardware modules and descriptions referred to in this document are ***NOT SUPPORTED*** by Texas Instruments (www.ti.com / e2e.ti.com).

These materials are intended for do-it-yourself (DIY) users who want to use the PRU at their own risk without TI support. "Community" support is offered at BeagleBoard.org/discuss.

Agenda

- **Introduction**
- PRU Sub-System Overview
- Getting Started Programming
- Other Resources

Introduction to the PRU SubSystem

- What is PRU SubSystem?
  - **P**rogrammable **R**eal-time **U**nit **S**ub**S**ystem
  - Dual 32bit RISC processors
  - Local instruction and data RAM; access to SoC resources.

- What devices include PRU SubSystem?
  - Legacy PRUSS: OMAPL137/ AM17x, OMAPL138/ AM18x, C674x
  - PRU-ICSS* (PRUSSv2): AM335x

- Why PRU SubSystem?
  - Full programmability allows adding customer differentiation
  - Efficient in performing embedded tasks that require manipulation of packed memory mapped data structures
  - Efficient in handling of system events that have tight real-time constraints.

* PRU-ICSS = **P**rogrammable **R**eal-time **U**nit and **I**ndustrial **C**ommunication **S**ub**S**ystem.

PRU Subsystem Is / Is-Not

IS	IS-Not
Dual 32-bit RISC processor specifically designed for manipulation of packed memory mapped data structures and implementing system features that have tight real time constraints	In not a H/W accelerator to speed up **algorithm computations**.
Simple RISC ISA - Approximately 40 instructions - Logical, arithmetic, and flow control ops all complete in a single cycle	Is **not a general purpose RISC processor** - No multiply hardware/instructions - No cache - No pipeline - No C programming
Could be used to enhance the existing peripheral feature set or implement new peripheral capability with software bit bang	Is not a stand alone configurable peripheral and will need some hardware assist for configurable peripheral implementation
Includes example code to demonstrate various features. Examples can be used as building blocks.	No Operating System or high level application software stack

PRU Value

- Extend connectivity and peripheral capability
  - Implement Industrial Communications protocols (like EtherCAT®,PROFINET, EtherNet/IP™, PROFIBUS, POWERLINK, SERCOS III)
  - Implement special peripherals and bus interfaces (like soft UARTs interfaces)
  - Digital IOs with latency in ns
  - Implement smart data movement schemes (especially useful for audio algorithms like reverb, room correction, etc.)

- Reduce system power consumption
  - Allows switching off both ARM and DSP clocks
  - Implement smart power controller by evaluating events before waking up DSP and/or ARM. Maximized power down time.

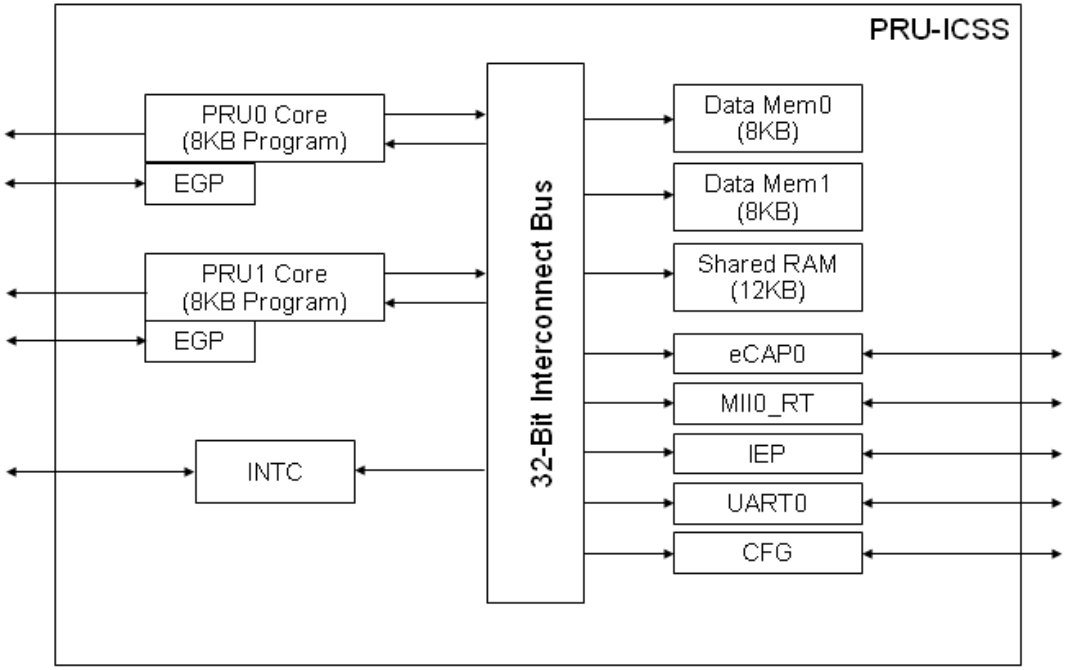
- Accelerate system performance
  - Full programmability allows custom interface implementation
  - Specialized custom data handling to offload CPU

Agenda

- Introduction
- **PRU Sub-System Overview**
  - PRU Overview
  - INTC
  - PRU-ICSS Peripherals
  - Instruction Set
- Getting Started Programming
- Other Resources

PRU-ICSS (PRUSSv2)

- Provides two independent programmable real-time (PRU) cores
  - 32-Bit Load/Store RISC architecture
  - 8K Byte instruction RAM (2K instructions) per core
  - 8K Bytes data RAM per core
  - 12K Bytes shared RAM
- Operating freq: 200 MHz
- PRU operation is little endian similar to ARM processor
- All memories within PRU-ICSS support parity
- Includes Interrupt Controller for system event handling
- Fast I/O interface
  - 30 input pins and 32 output pins per PRU core. (Only 17 GPI and 16 GPO pinned out on AM335x.)
- Integrated peripherals



Enhancements in PRU-ICSS compared to Legacy PRUSS

- Memory
  - Additional data memory (8K Bytes vs 512 Bytes)
  - Additional instruction memory (8K Bytes vs 4K Bytes)
  - 12 KB Shared RAM
  - All memories within PRU-ICSS support parity
- PRU Resources
  - Enhanced GPIO (EGPIO), adding serial, parallel, and MII capture capabilities
- Internal peripheral modules
  - UART
  - eCAP
  - MII_RT
  - MDIO
  - IEP
- Operating frequency
  - Legacy PRUSS: ½ CPU frequency
  - PRU-ICSS: 200 MHz

Local & Global Memory Map

- Local Memory Map
  - Allows PRU to directly access subsystem resources, e.g. DRAM, INTC registers, etc.
  - NOTE: Memory map slightly different from PRU0 and PRU1 point-of-view.

Instruction Space Memory Map

Start	End	PRU0	PRU1
0x0000_0000	0x0000_1FFF	PRU0 Instruction RAM	PRU1 Instruction RAM

Data Space Memory Map

Start	End	PRU0	PRU1
0x0000_0000	0x0000_1FFF	Data 8KB RAM 0*	Data 8KB RAM 1*
0x0000_2000	0x0000_FFFF	Data 8KB RAM 1*	Data 8KB RAM 0*
0x0001_0000	0x0001_FFFF	Data 12KB RAM2	Data 12KB RAM2
0x0002_0000	0x0002_1FFF	INTC	INTC
0x0002_2000	0x0002_23FF	PRU0 Control Registers	PRU0 Control Registers
0x0002_2400	0x0002_3FFF	Reserved	Reserved
0x0002_4000	0x0002_43FF	PRU1 Control	PRU1 Control
0x0002_4400	0x0002_5FFF	Reserved	Reserved
0x0002_6000	0x0002_7FFF	CFG	CFG
0x0002_8000	0x0002_9FFF	UART 0	UART 0
0x0002_A000	0x0002_BFFF	Reserved	Reserved
0x0002_C000	0x0002_DFFF	Reserved	Reserved
0x0002_E000	0x0002_FFFF	IEP	IEP
0x0003_0000	0x0003_1FFF	eCAP 0	eCAP 0
0x0003_2000	0x0003_23FF	MII_RT_CFG	MII_RT_CFG
0x0003_2400	0x0003_3FFF	MII_MDIO	MII_MDIO
0x0003_4000	0x0003_7FFF	Reserved	Reserved
0x0003_8000	0x0003_FFFF	Reserved	Reserved
0x0004_0000	0x0007_FFFF	Reserved	Reserved
0x0008_0000		System OCP_HP0	System OCP_HP1

Global Address Map

Start	End	PRUSS
0x4A30_0000	0x4A30_1FFF	Data 8KB RAM 0
0x4A30_2000	0x4A30_FFFF	Data 8KB RAM 1
0x4A31_0000	0x4A31_FFFF	Data 12KB RAM 2
0x4A32_0000	0x4A32_1FFF	INTC
0x4A32_2000	0x4A32_23FF	PRU0 Control
0x4A32_2400	0x4A32_3FFF	PRU0 Debug
0x4A32_4000	0x4A32_43FF	PRU1 Control
0x4A32_4400	0x4A32_5FFF	PRU1 Debug
0x4A32_6000	0x4A32_7FFF	CFG
0x4A32_8000	0x4A32_9FFF	UART 0
0x4A32_A000	0x4A32_BFFF	Reserved
0x4A32_C000	0x4A32_DFFF	Reserved
0x4A32_E000	0x4A32_FFFF	IEP
0x4A33_0000	0x4A33_1FFF	eCAP 0
0x4A33_2000	0x4A33_23FF	MII_RT_CFG
0x4A33_2400	0x4A33_3FFF	MII_MDIO
0x4A33_4000	0x4A33_7FFF	PRU0 8KB IRAM
0x4A33_8000	0x4A33_FFFF	PRU1 8KB IRAM
0x4A34_0000	0x4A37_FFFF	Reserved

- Global Memory Map
  - Allows external masters to access PRU subsystem resources, e.g. debug and control registers.
  - PRU cores can also use global memory map, but more latency since access routed externally.

Agenda

- Introduction
- PRU Sub-System Overview
  - **PRU Overview**
  - INTC
  - PRU-ICSS Peripherals
  - Instruction Set
- Getting Started Programming
- Other Resources

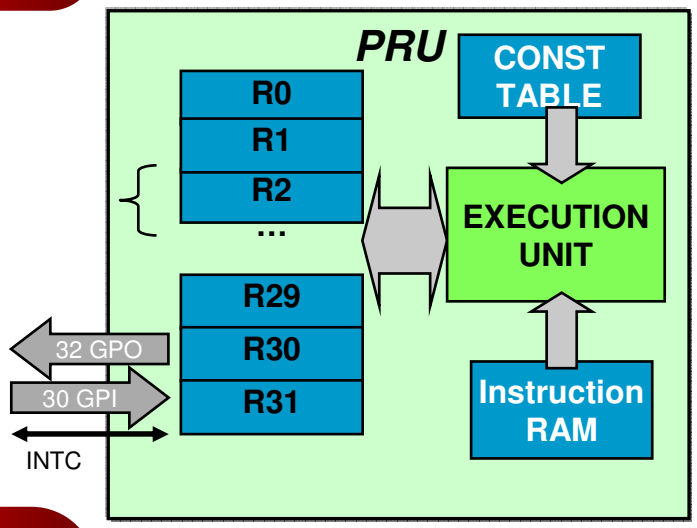
PRU Functional Block Diagram

General Purpose Registers

- All instructions are performed on registers and complete in a single cycle
- Register file appears as linear block for all register to memory operations

Constant Table

- Ease SW development by providing freq used constants
- Peripheral base addresses
- Few entries programmable



Execution Unit

- Logical, arithmetic, and flow control instructions
- Scalar, no Pipeline, Little Endian
- Register-to-register data flow
- Addressing modes: Ld Immediate & Ld/St to Mem

Special Registers (R30 and R31)

- R30
  - Write: 32 GPO (AM335x: 16 GPO pinned out)
- R31
  - Read: 30 GPI + 2 Host Int status (AM335x: 17 GPI pinned out)
  - Write: Generate INTC Event

Instruction RAM

- 8KB in size; 2K Instructions
- Can be updated with PRU reset

PRU Constants Table

- Load and store instructions require that the destination/source base address be loaded in a register.
- Constants table is a list of 32 commonly-used addresses that can be used in memory load and store operations via special instructions.
- Most constant table entries are fixed, but some contain a programmable bit field that is programmable through the PRU control registers.
- Using the constants table saves both the register space as well as the time required to load pointers into registers.

PRU0/1 Constants Table (AM335x)

Entry No.	Region Pointed To	Value [31:0]
0	PRU0/1 Local INTC	0x0002_0000
1	DMTIMER2	0x4804_0000
2	I2C1	0x4802_A000
3	eCAP (local)	0x0003_0000
4	PRU-ICSS CFG (local)	0x0002_6000
5	MMCHS 0	0x4806_0000
6	MCSPI 0	0x4803_0000
7	UART0 (local)	0x0002_8000
8	McASP0 DMA	0x4600_0000
9	GEMAC	0x4A10_0000
10	Reserved	0x4831_8000
11	UART1	0x4802_2000
12	UART2	0x4802_4000
13	Reserved	0x4831_0000
14	DCAN0	0x481C_C000
15	DCAN1	0x481D_0000

Entry No.	Region Pointed To	Value [31:0]
16	MCSPI 1	0x481A_0000
17	I2C2	0x4819_C000
18	eHRPWM1/eCAP1/eQEP1	0x4830_0000
19	eHRPWM2/eCAP2/ePWM2	0x4830_2000
20	eHRPWM3/eCAP3/ePWM3	0x4830_4000
21	MDIO (local)	0x0003_2400
22	Mailbox 0	0x480C_8000
23	Spinlock	0x480C_A000
24	PRU0/1 Local Data	0x0000_0n00, n = c24_blk_index[3:0]
25	PRU1/0 Local Data	0x0000_2n00, n = c25_blk_index[3:0]
26	IEP (local)	0x0002_En00, n = c26_blk_index[3:0]
27	MII_RT (local)	0x0003_2n00, n = c27_blk_index[3:0]
28	Shared PRU RAM (local)	0x00nn_nn00, nnnn = c28_pointer[15:0]
29	TPCC	0x49nn_nn00, nnnn = c29_pointer[15:0]
30	L3 OCMC0	0x40nn_nn00, nnnn = c30_pointer[15:0]
31	EMIF0 DDR Base	0x80nn_nn00, nnnn = c31_pointer[15:0]

NOTES

1. Constants not in this table can be created 'on the fly' by loading two 16-bit values into a PRU register. These constants are just ones that are expected to be commonly used, enough so to be hard-coded in the PRU constants table.

2. Constants table entries 24 through 31 are not fully hard coded, they contain a programmable bit field that is programmable through the PRU control registers. Programmable entries allow you to select different 256-byte pages within an address range.

PRU Event/Status Register (R31)

- Writes: Generate output events to the INTC.
  - Write the event number (0 through 15) to PRU_VEC[3:0] and simultaneously set PRU_VEC_VALID to create a pulse to INTC.
  - Outputs from both PRUs are ORed together to form single output.
  - Output events 0 through 15 are connected to system events 16 through 31 on INTC.
- Reads: Return Host 1 & 0 interrupt status from INTC and general purpose input pin status.

R31 During Writes

Bit	Name	Description
31:6	RSV	Reserved
5	PRU_VEC_VALID	Valid strobe for vector output
4	RSV	Reserved
3:0	PRU_VEC[3:0]	Vector output

R31 During Reads

Bit	Name	Description
31	PRU_INTR_IN[1]	PRU Host 1 interrupt from INTC
30	PRU_INTR_IN[0]	PRU Host 0 interrupt from INTC
29:0	PRU_R31_STATUS[29:0]	Status inputs from PRU*n*_R31[29:0]

Dedicated GPIs and GPOs

- General purpose inputs (GPIs)
  - Each PRU has 30 general purpose input pins: PRU0_R31[29:0] and PRU1_R31[29:0].
  - Reading R31[29:0] in each PRU returns the status of PRU*n*_R31[29:0].
  - On AM335x, only PRU0_R31[16:0] and PRU1_R31[16:0] are pinned out.

- General purpose outputs (GPOs)
  - Each PRU has 32 general purpose output pins: PRU0_R30[31:0] and PRU1_R30[31:0].
  - The value written to R30[31:0] is driven on PRU*n*_R30[31:0].
  - On AM335x, only PRU0_R30[15:0] and PRU1_R30[15:0] are pinned out.

- Notes
  - Unlike the device GPIOs, PRU GPIs and GPOs may be assigned to different pins.
  - You can use the "." operator to read or write a single bit in R30 and R31, e.g. R30.t0.
  - PRU GPOs and GPIs are enabled through the system pin mux registers.

Enhanced GPIO Interface

- Legacy PRUSS only supported direct connect GPIO interface.
- PRU-ICSS (PRUSSv2) supports enhanced GPIO interface.
  - GPI modes:
    - Direct connect (17 GPIs per core)
    - 16-bit parallel capture (1 parallel capture GPI per core)
    - 28-bit shift (1 GPI serializer per core)
  - GPO modes:
    - Direct connect (16 GPOs per core)
    - Shift out (1 GPO serializer per core)
  - GPIO modes are programmable through PRU-ICSS CFG.
  - Only one mode can be active at a time.

PRU-ICSS Enhanced GPIO Signals

GPI Signals

Function	Alias	Internal Signal Name
Direct Mode		
Data input	PRU<n>_DATAIN	pru<n>_r31 [16:0]
Parallel Capture Mode		
Data input	PRU<n>_DATAIN	pru<n>_r31 [15:0]
Clock	PRU<n>_CLOCK	pru<n>_r31 [16]
Shift Mode		
Data input	PRU<n>_DATAIN	pru<n>_r31 [0]
Shift counter	PRU<n>_CNT_16	pru<n>_r31 [28]
Start bit detection	PRU<n>_GPI_SB	pru<n>_r31 [29]

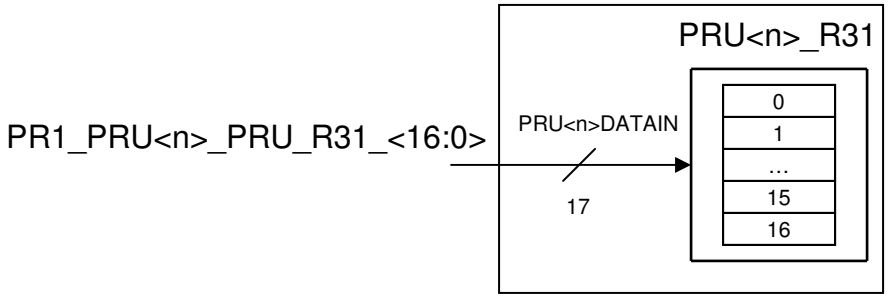
GPO Signals

Function	Alias	Internal Signal Name
Direct Mode		
Data output	PRU<n>_DATAOUT	pru<n>_r31 [15:0]
Shift Mode		
Data output	PRU<n>_DATAOUT	pru<n>_r30 [0]
Clock	PRU<n>_CLOCK	pru<n>_r30 [1]
Load gpo_sh0	PRU<n>_LOAD_GPO_SH0	pru<n>_r30 [29]
Load gpo_sh1	PRU<n>_LOAD_GPO_SH1	pru<n>_r30 [30]
Enable shift	PRU<n>_ENABLE_SHIFT	pru<n>_r30 [31]

Direct Connect Modes

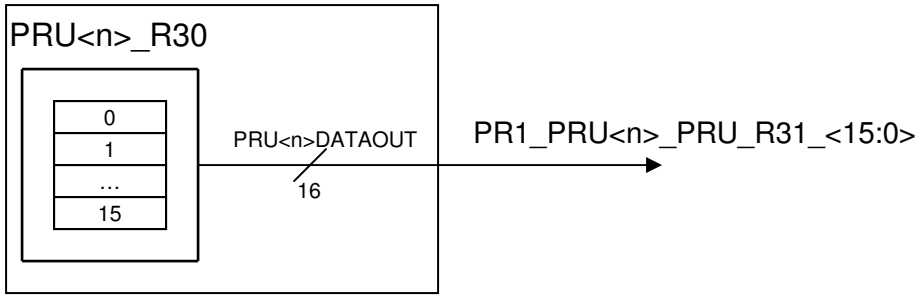
GPI

- PRU<n> R31 [16:0] feed directly into the PRU



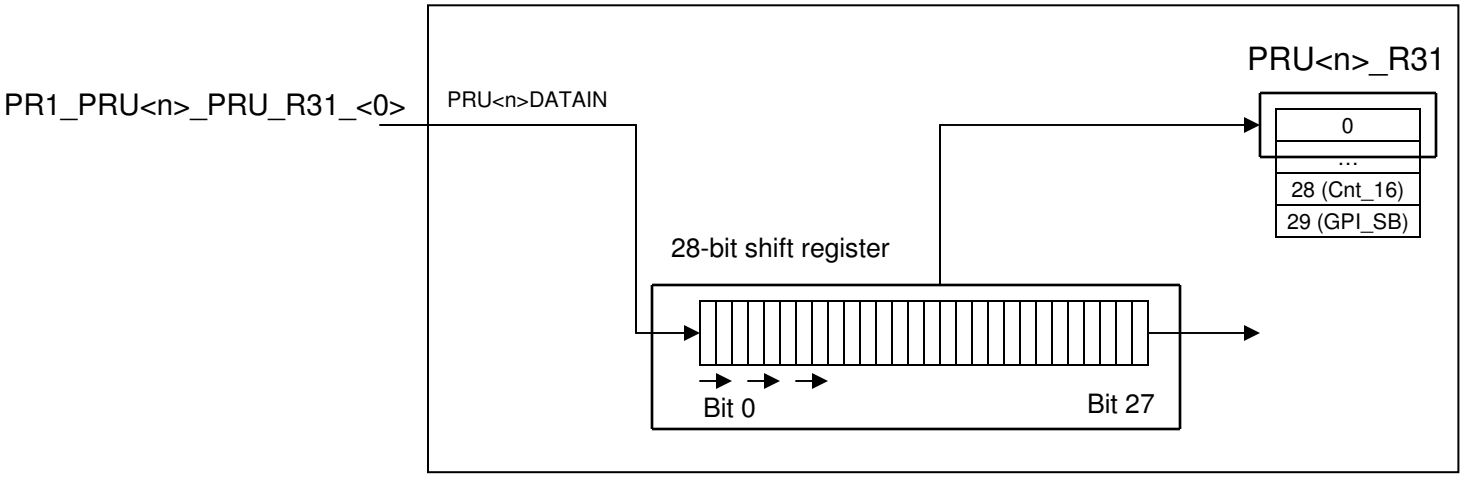
GPO

- PRU<n> R30 [15:0] feed directly out of the PRU



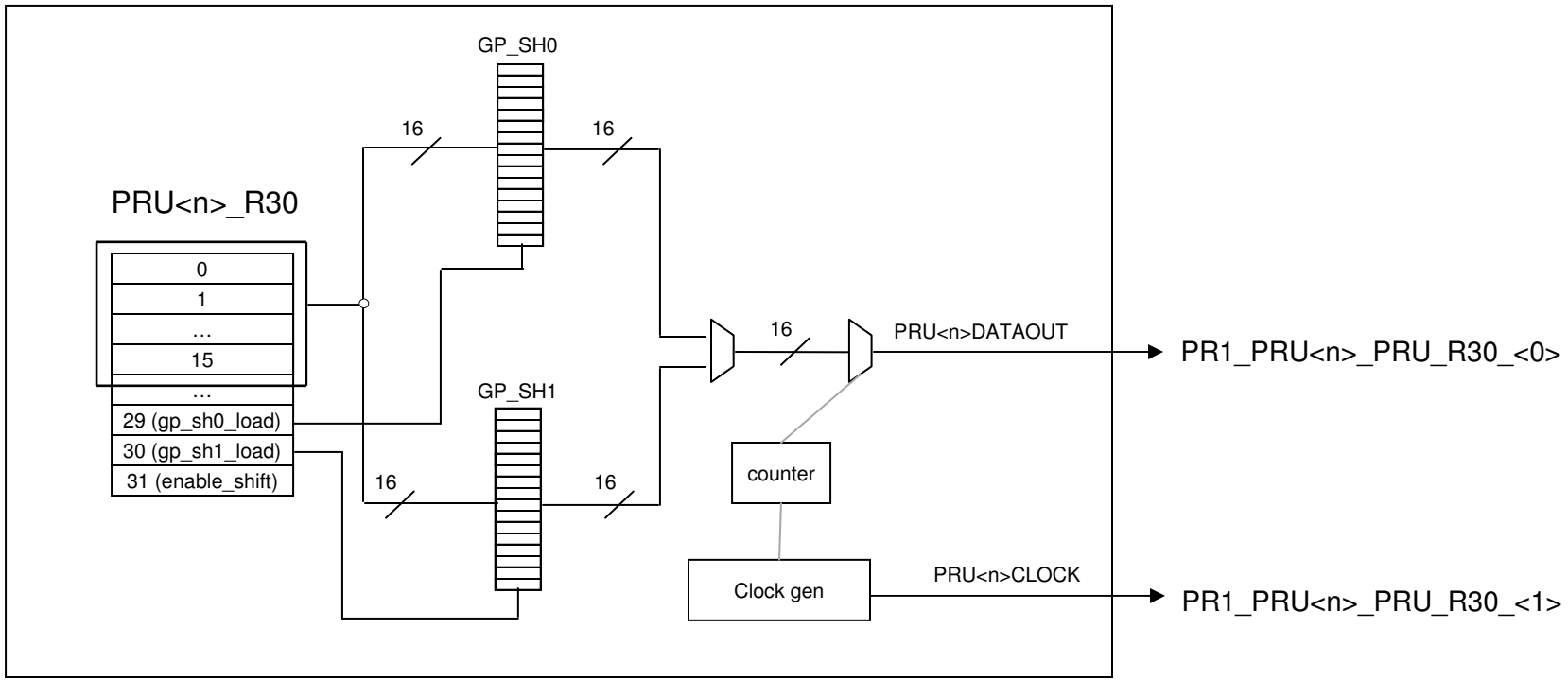
Shift Modes (GPI)

- PRU<n> R31[0] is sampled and shifted into a 28-bit shift register.
  - Shift Counter (Cnt_16) feature uses pru<n>_r31_status [28]
  - Start Bit detection (SB) feature uses pru<n>_r31_status [29]
- Shift rate controlled by effective divisor of two cascaded dividers applied to the 200MHz clock.
  - Each cascaded dividers is configurable through the PRU-ICSS CFG to a value of {1, 1.5, …, 16}.



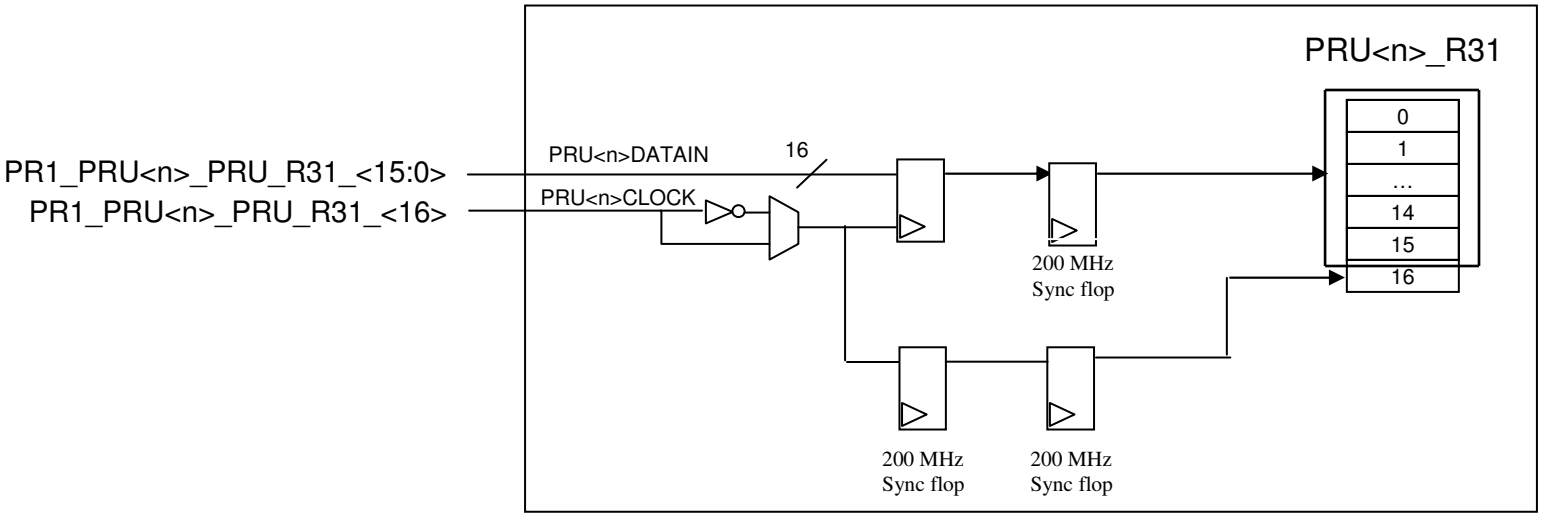
Shift Modes (GP0)

- PRU<n> R30[0] is shifted out on every rising edge of the internal PRU<n>_CLOCK (pru<n>r30 [1]).
- Shift rate is controlled by the effective divisor of two cascaded dividers applied to the 200MHz clock. See Shift Mode (GPI).



Parallel Capture Mode (GPI)

- PRU<n>_R31 [15:0] is captured by posedge or negedge of PRU<n>_CLOCK (pru<n>_r31_status [16]).



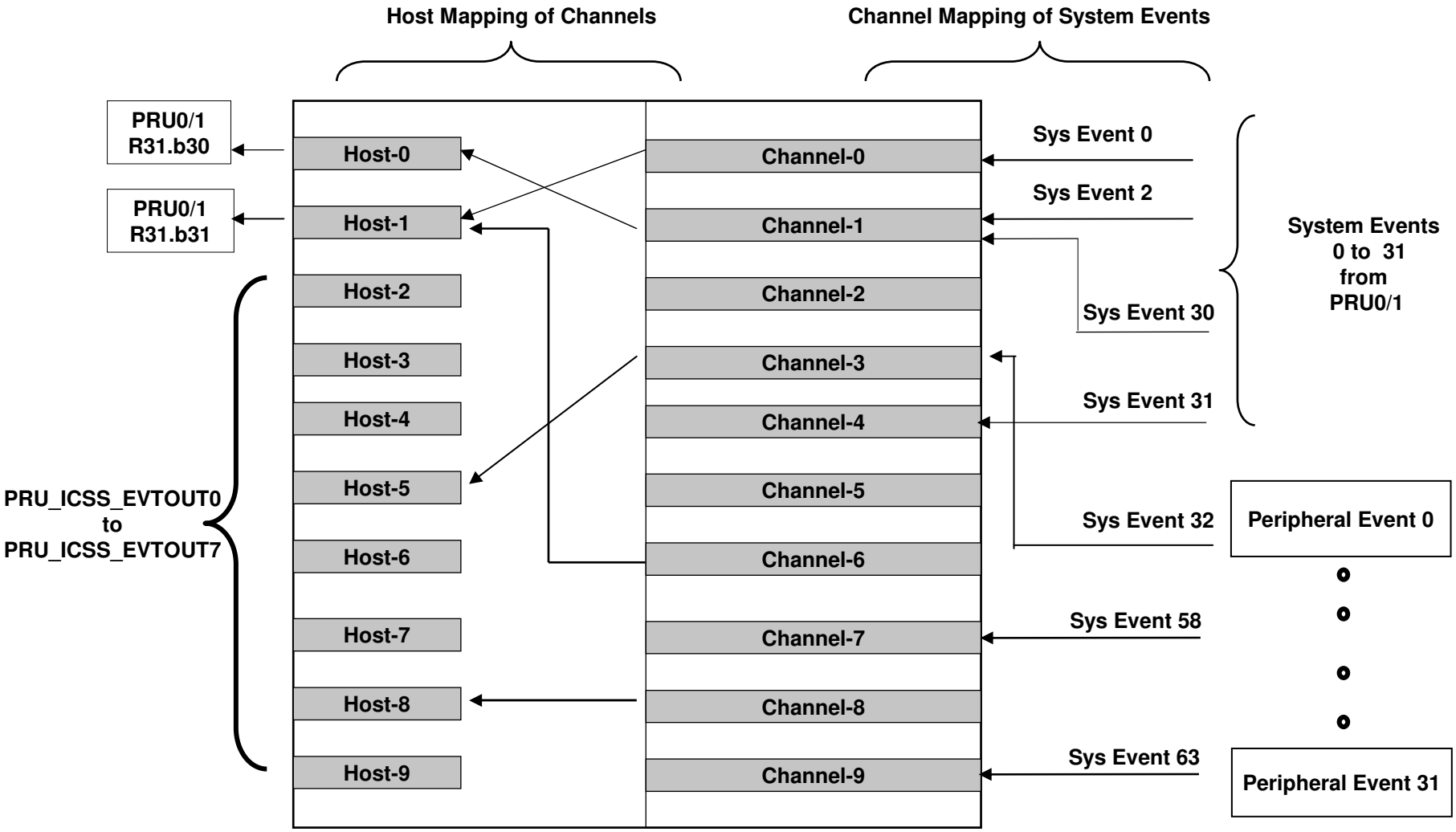
Agenda

- Introduction
- PRU Sub-System Overview
  - PRU Overview
  - **INTC**
  - PRU-ICSS Peripherals
  - Instruction Set
- Getting Started Programming
- Other Resources

Interrupt Controller (INTC) Overview

- Supports 64 system events
  - 32 system events external to the PRU subsystem
  - 32 system events generated directly by the PRU cores
- Supports up to 10 interrupt channels
  - Allows for interrupt nesting.
- Generation of 10 host interrupts
  - Host Interrupt 0 mapped to R31.b30 in both PRUs
  - Host Interrupt 1 mapped to R31.b31 in both PRUs
  - Host Interrupt 2 to 9 routed to ARM and DSP INTCs.
- System events can be individually enabled, disabled, and manually triggered
- Each host event can be enabled and disabled
- Hardware prioritization of system events and channels

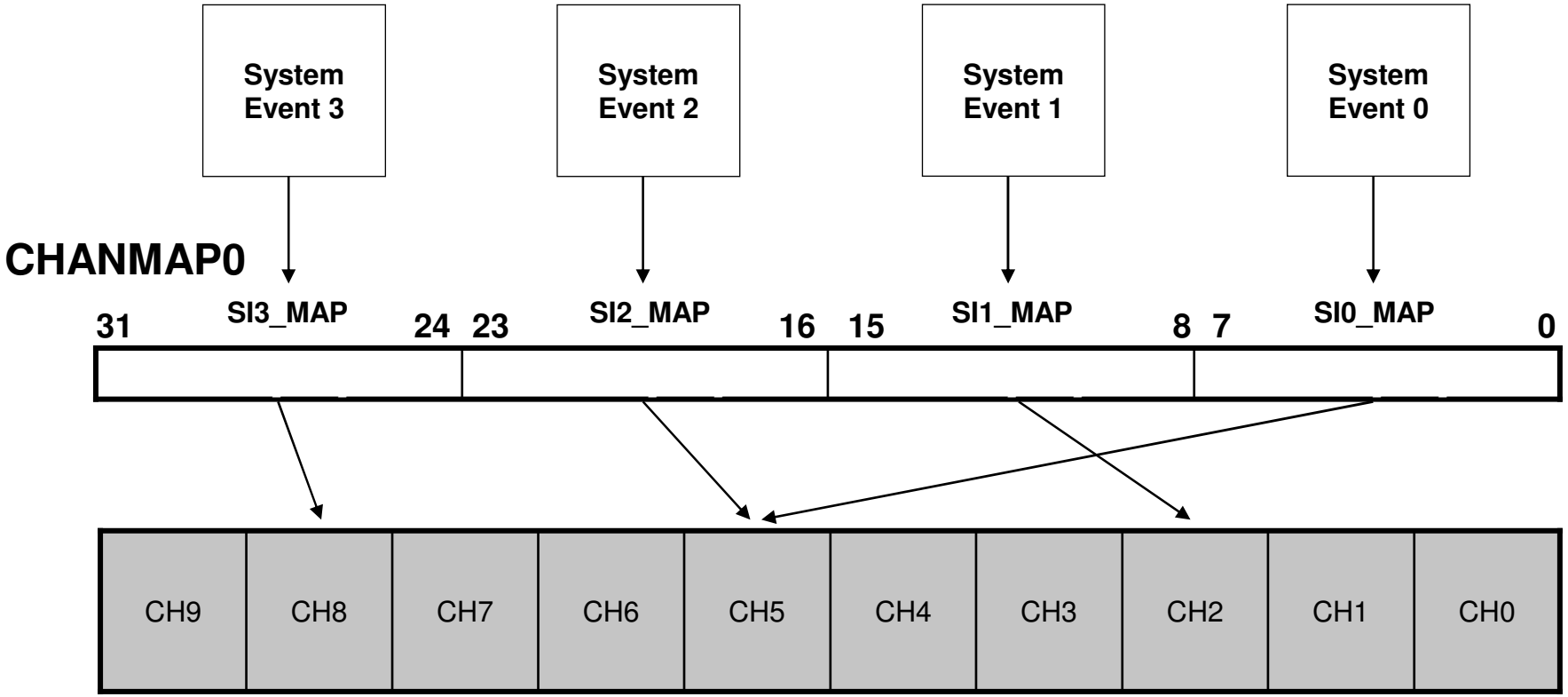
Interrupt Controller Block Diagram



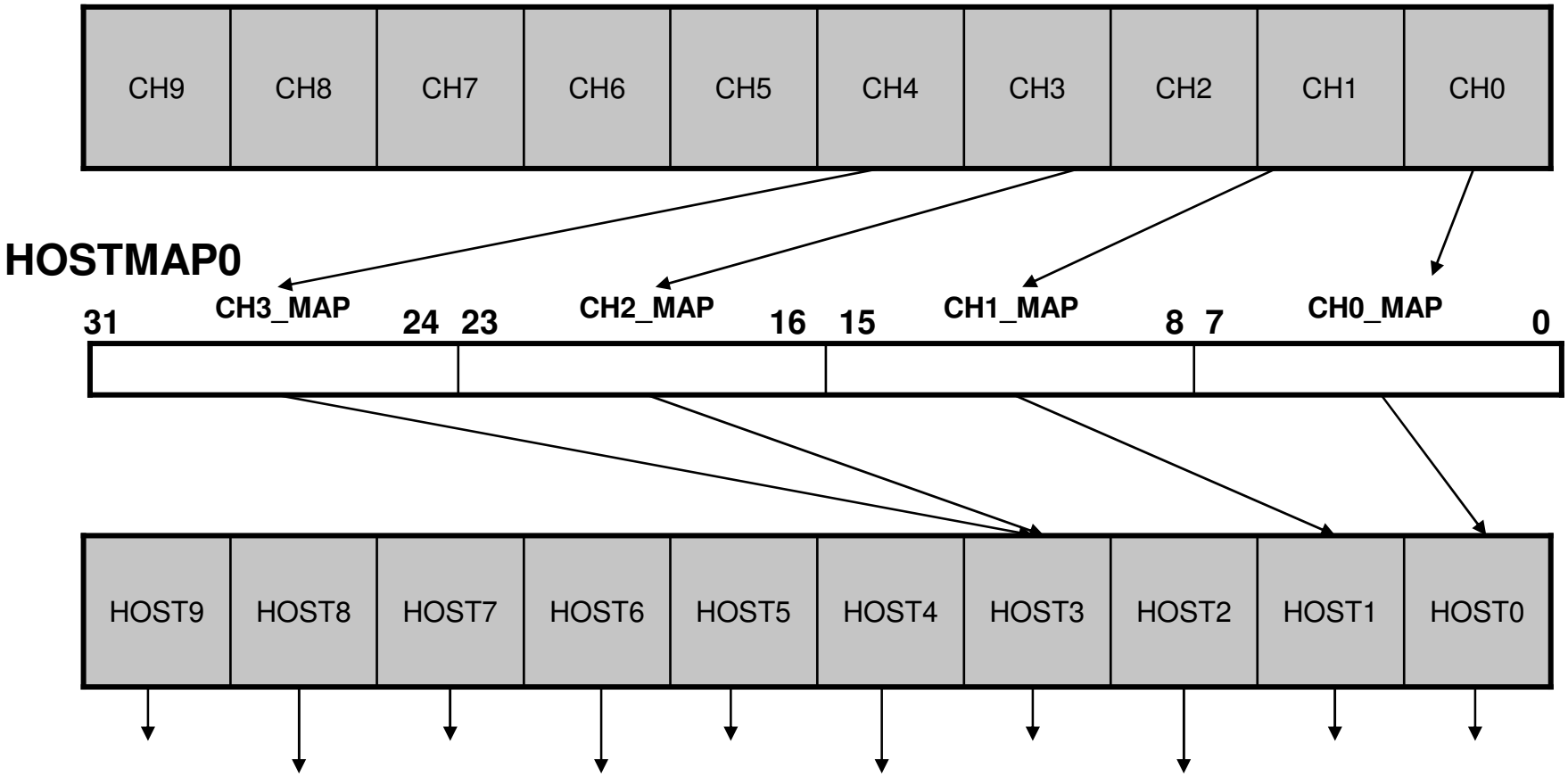
Interrupt Controller Mapping

- System events must be mapped to channels
  - Multiple system events can be mapped to the same channel.
  - Not possible to map system events to more than one channel.
  - System events mapped to same channel ➔ lower-numbered events have higher priority
- Channels must be mapped to host interrupts
  - Multiple channels can be mapped to the same host interrupt.
  - Not possible to map channels to more than one host interrupt.
  - Recommended to map channel “x” to host interrupt “x”, where “x” is from 0 to 9.
  - Channels mapped to the same host interrupt ➔ lower-numbered channels have higher priority

System Event to Channel Mapping



Channel to Host Interrupt Mapping



*** Recommended to map channel "x" to host interrupt "x".**

Agenda

- Introduction
- PRU Sub-System Overview
  - PRU Overview
  - INTC
  - **PRU-ICSS Peripherals**
  - Instruction Set
- Getting Started Programming
- Other Resources

Integrated Peripherals

- PRU-ICSS integrates some peripherals to reduce latency of the PRU accessing these peripherals.
- PRU-ICSS peripherals can be used by the PRU or by the ARM as additional hardware peripherals on the device.
  - ARM has full access of PRU-ICSS peripheral registers.
  - Interrupt mapping through PRU INTC required.
- Integrated peripherals:
  - PRU UART
    - Same as AM1808 UART
    - Supports up to 12M baud
  - PRU eCAP
    - Same as AM335x eCAP module
  - PRU MDIO, MII_RT, IEP
    - EtherCAT-specific modules

Agenda

- Introduction
- PRU Sub-System Overview
  - PRU Overview
  - INTC
  - PRU-ICSS Peripherals
  - **Instruction Set**
- Getting Started Programming
- Other Resources

PRU Instruction Overview

- Four instruction classes
  - Arithmetic
  - Logical
  - Flow Control
  - Register Load/Store
- Instruction Syntax
  - Mnemonic, followed by comma separated parameter list
  - Parameters can be a register, label, immediate value, or constant table entry
  - Example
    - SUB r3, r3, 10
    - Subtracts immediate value 10 (decimal) from the value in r3 and then places the result in r3
- Nearly all instructions (with exception of accessing memory external to PRU) are single-cycle execute
  - 5 ns when running at 200 MHz

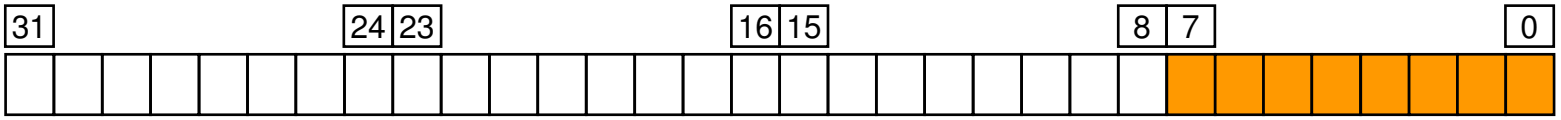
PRU Register Accesses

- PRU is suited to handling packets and structures, parsing them into fields and other smaller data chunks
- Valid registers formats allow individual selection of bits, bytes, and half-words from within individual registers
- The parts of the register can be accessed using the modifier suffixes shown

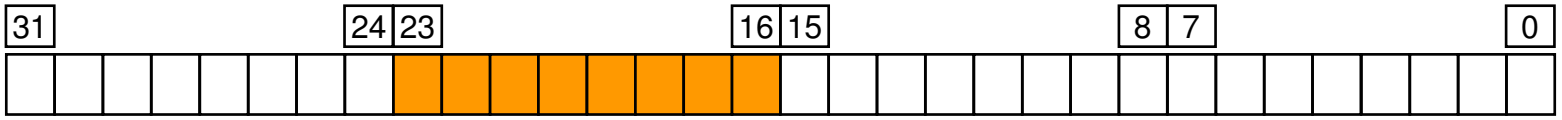
Suffix	Range of n	Meaning
.wn	0 to 2	16 bit field with a byte offset of *n* within the parent field
.bn	0 to 3	8 bit field with a byte offset of *n* within the parent field
.tn	0 to 31	1 bit field with a bit offset of *n* within the parent field

Register Examples

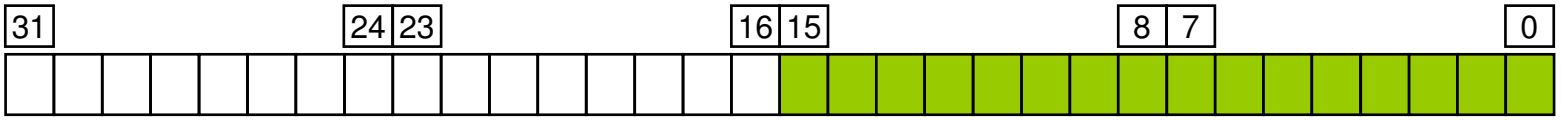
- r0.b0



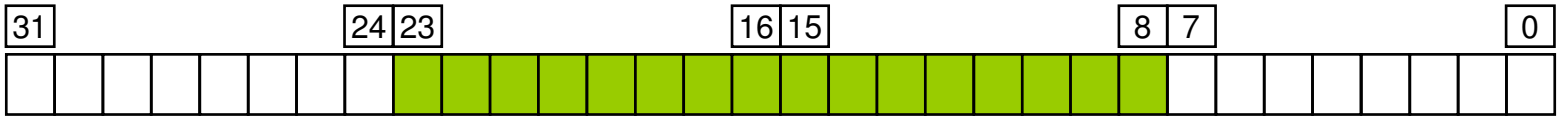
- r0.b2



- r0.w0

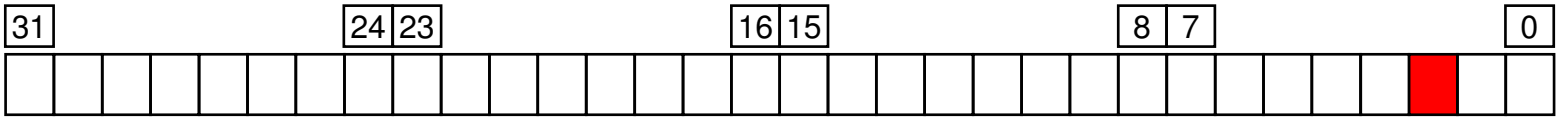


- r0.w1

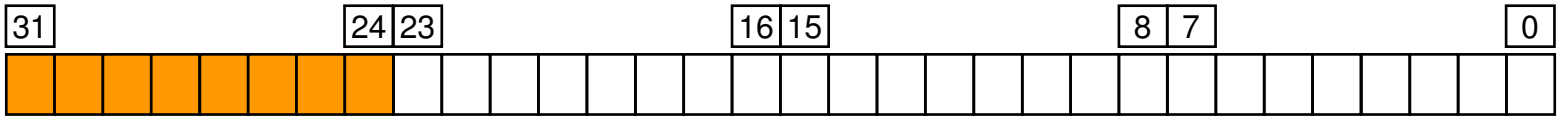


Register Examples, cont'd

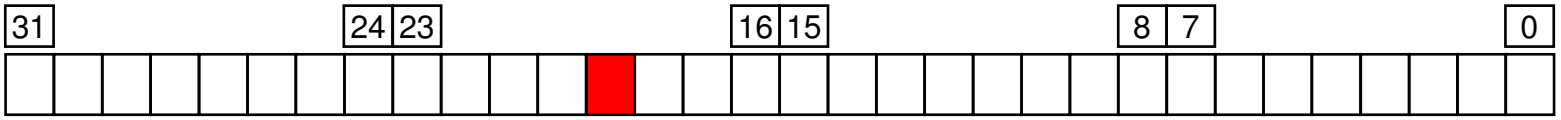
- r0.t2



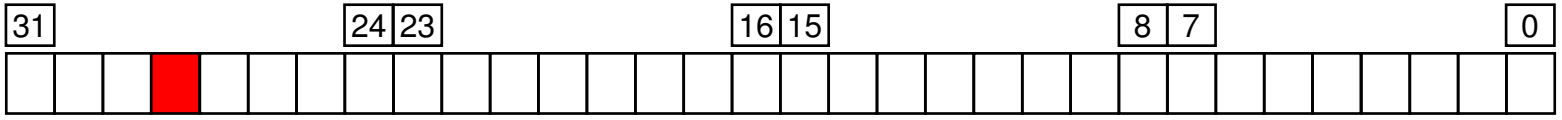
- r0.w2.b1 = r0.b3



- r0.w1.b1.t3 = r0.b2.t3 = r0.t19



- r0.w2.t12 = r0.t28



PRU Instruction Set

Arithmetic Operations (green) **Logic Operations (blue)**

IO Operations (black)

Op-code (Italic)

ADD	ADC	SUB	SUC	RSB
RSC	LSL	LSR	AND	OR
XOR	NOT	MIN	MAX	CLR
SET	SCAN	LMBD	*MOV*	LDI
LBBO	SBBO	LBCO	SBCO	*ZERO*
MVIB	*MVIW*	*MVID*	JAL	JMP
QBGT	QBGE	QBLT	QBLE	QBEQ
QBNE	QBA	QBBS	QBBC	*WBS*
WBC	HALT	SLP	*CALL*	*RET*

Agenda

- Introduction
- PRU Sub-System Overview
- Getting Started Programming
  - **PRU Assembler (PASM)**
  - Linux PRU Application Loader
- Other Resources

PASM Overview

- PASM is a command-line assembler for the PRU cores
  - Converts PRU assembly source files to loadable binary data
  - Output format can be raw binary, C array (default), or hex
    - The C array can be loaded by host processor (ARM or DSP) to kick off PRU
  - Other debug formats also can be output
- Command line syntax:

  pasm [-bcmldxz] SourceFile [-Dname=value] [-CArrayname]
- The PASM tool generates a single monolithic binary
  - No linking, no sections, no memory maps, etc.
  - Code image begins at start of IRAM (offset 0x0000)

Valid Assembly File Inputs

- Four basic assembler statements
  - Hash commands
  - Dot commands (directives)
  - Labels
  - Instructions
    - True instructions (defined previously)
    - Pseudo-instructions
- Assembly comments allowed and ignored
  - Use the double slash single-line format of C/C++
  - Always appear as last field on a line
  - Example:

```
//-------------------------
// This is a comment
//-------------------------
ldi r0, 100 // This is a comment
```

Assembler Hash statements

- Similar to C pre-processor commands
- #include”filename”
  - Specified filename is immediately opened, parsed, and processed
  - Allows splitting large PRU assembly code into separate files
- #define
  - Specify a simple text substitution
  - Can also be used to define empty substitution for use with #ifdef, #ifndef, etc.
- #undef – Used to undefine a substitution previously defined with #define
- Others (#ifdef, #ifndef, #else, #endif, #error) as used in C preprocessor

Assembler Dot Commands

- All dot commands start with a period (the dot)
- Rules for use
  - Must be only assembly statement on line
  - Can be followed by comments
  - Not required to start in column 0

Command	Description
.origin	Set start of next assembly statement
.entrypoint	Only used for debugger, specifies starting address
.setcallreg	Specified 16-bit register field for storing return pointer
.macro, .mparam, .endm	Define assembler macros
.struct, .ends, .u32, .u16, .u8	Define structure types for easier register allocation
.assign	Map defined structure into PRU register file
.enter	Create and enter new variable scope
.leave	Leave a specific variable scope
.using	Use a previously created and left scope

Macro Example

- PASM macros using dot commands expand are like C preprocessor macros using #define
- They save typing and can make code cleaner
- Example macro:

```
//
// mov32 : Move a 32bit value to a register
//
// Usage:
// mov32 dst, src
//
// Sets dst = src. Src must be a 32 bit immediate value.
//
.macro MOV32
.mparam dst, src
  LDI dst.w0, src & 0xFFFF
  LDI dst.w2, src >> 16
.endm
```

- Macro invoked as:

```
MOV32 r0, 0x12345678
```

* **Note:** The latest assembler supports 32-bit immediate values natively, making this mov32 MACRO undesirable for general use.

Struct Example

- Like in C, defined structures can be useful for defining offsets and mapping data into registers/memory
- Declared similar to using typedef in C
  - PASM automatically processes each declared structure template and creates an internal structure type.
  - The named structure type is not yet associated with any registers or storage.

- Example from C:

```
typedef struct _PktDesc_
{
  struct _PktDesc *pNext;
  char *pBuffer;
  unsigned short Offset;
  unsigned short BufLength;
  unsigned short Flags;
  unsigned short PktLength;
} PKTDESC;
```

- Now in PASM assembly:

```
.struct PktDesc
  .u32 pNext
  .u32 pBuffer
  .u16 Offset
  .u16 BufLength
  .u16 Flags
  .u16 PktLength
.ends
```

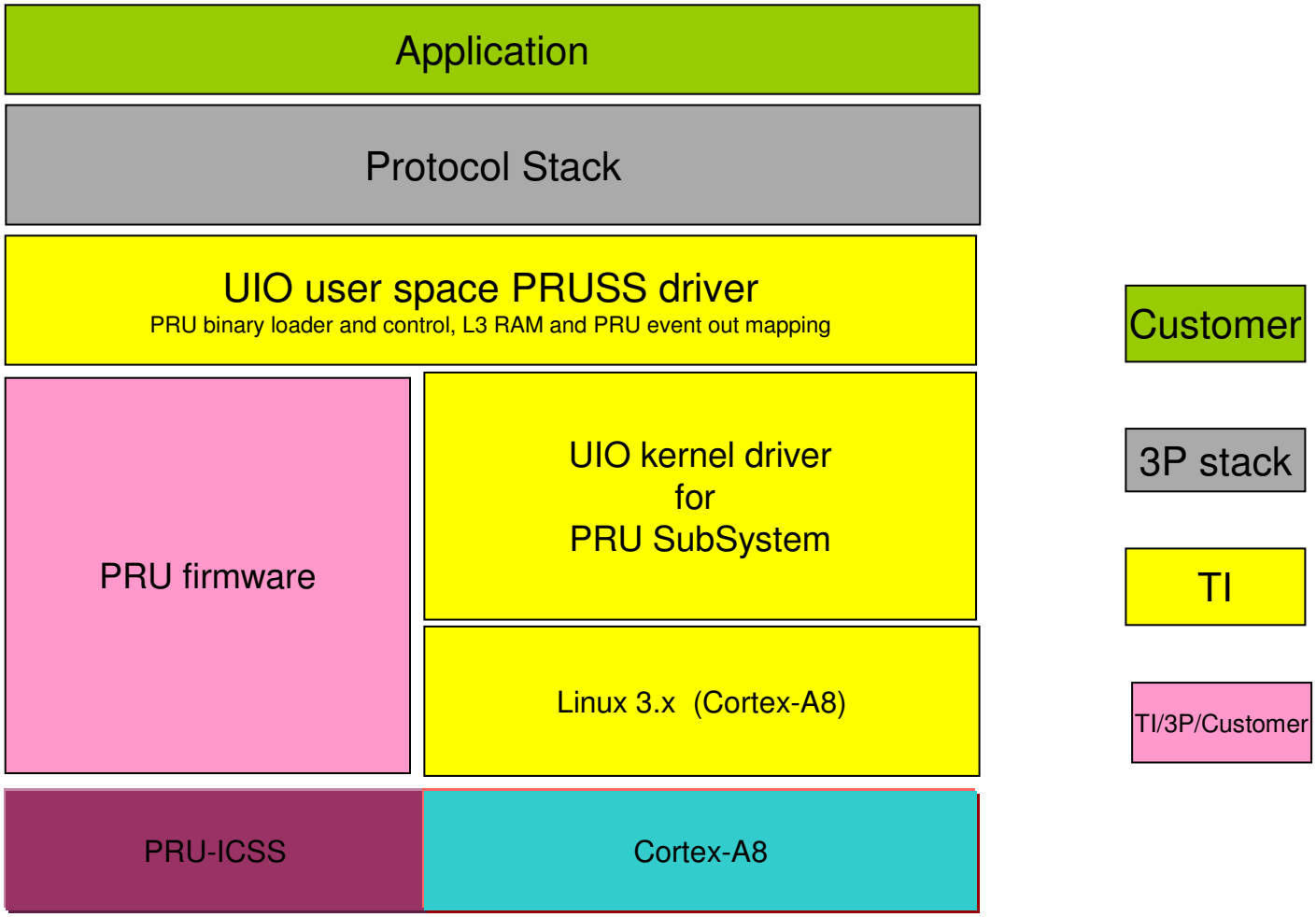
Agenda

- Introduction
- PRU Sub-System Overview
- Getting Started Programming
  - PRU Assembler (PASM)
  - **Linux PRU Application Loader**
- Other Resources

PRU Linux Loader

- Host processor of SoC must load code to a PRU and initiate its execution
- ARM processor can load code to PRU instruction memory and interact with PRU from user space using the application loader
- Application loader is available in open-source
- Application PRU Loader
  - API's allow ARM to interact with PRU in user space
  - Supports BSD licensing
  - Can be used for protocol emulation and user space applications

Application Loader S/W Architecture



Application Loader Examples

AM1808 SDK

- AM335x PRU package includes several basic PRU application example code. These examples use the Linux application loader.
  - Additional PRU examples can be found in the AM1808 SDK.
  - The AM18x PRUSS to AM335x PRU-ICSS Software Migration Guide provides reference of how these examples can be ported to AM335x.
- PRU example code demonstrates:
  - Memory transfers
  - Accessing constant tables *
  - Interrupts
  - Toggling GPIOs *
  - eDMA configuration *
- AM1808 SDK can be downloaded at:
  http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sdk/AM1x/latest/index_FDS.html

* Only included in AM1808 examples.

Steps to use develop code

- To Use,
  - Build UIO kernel driver as module
  - Build User space API's, link to application code
  - Compile application code using API
  - Compile PRU binaries using PASM

  - On file system, install UIO kernel driver, application executables, PRU binaries

Agenda

- Introduction
- PRU Sub-System Overview
- Getting Started Programming
- **Other Resources**

PRU tools/software/documentation

Legacy PRUSS

- Overview - http://tiexpressdsp.com/index.php/Programmable_Realtime_Unit_Subsystem
- Programming Guide – http://tiexpressdsp.com/index.php/Programmable_Realtime_Unit_Software_Development
- Software Development Package including assembler – http://focus.ti.com/docs/toolsw/folders/print/sprc940.html
- AM1808 PRU Linux Loader - http://processors.wiki.ti.com/index.php/PRU_Linux_Loader

- AM1808 SDK with PRU examples - http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/sdk/AM1x/latest/index_FDS.html
- AM1808 PSP - http://processors.wiki.ti.com/index.php/Community_Linux_PSP_for_DA8x/OMAP-L1/AM1x
- Soft-UART code and documentation – http://processors.wiki.ti.com/index.php/Soft-UART_Implementation_on_OMAPL_PRU_-_Software_Users_Guide

PRU tools/software/documentation

PRU-ICSS, or PRUSSv2 (AM335x)

- AM335x PRU package

  http://github.com/beagleboard/am335x_pru_package