Building Blocks for PRU Development

Embedded Processing

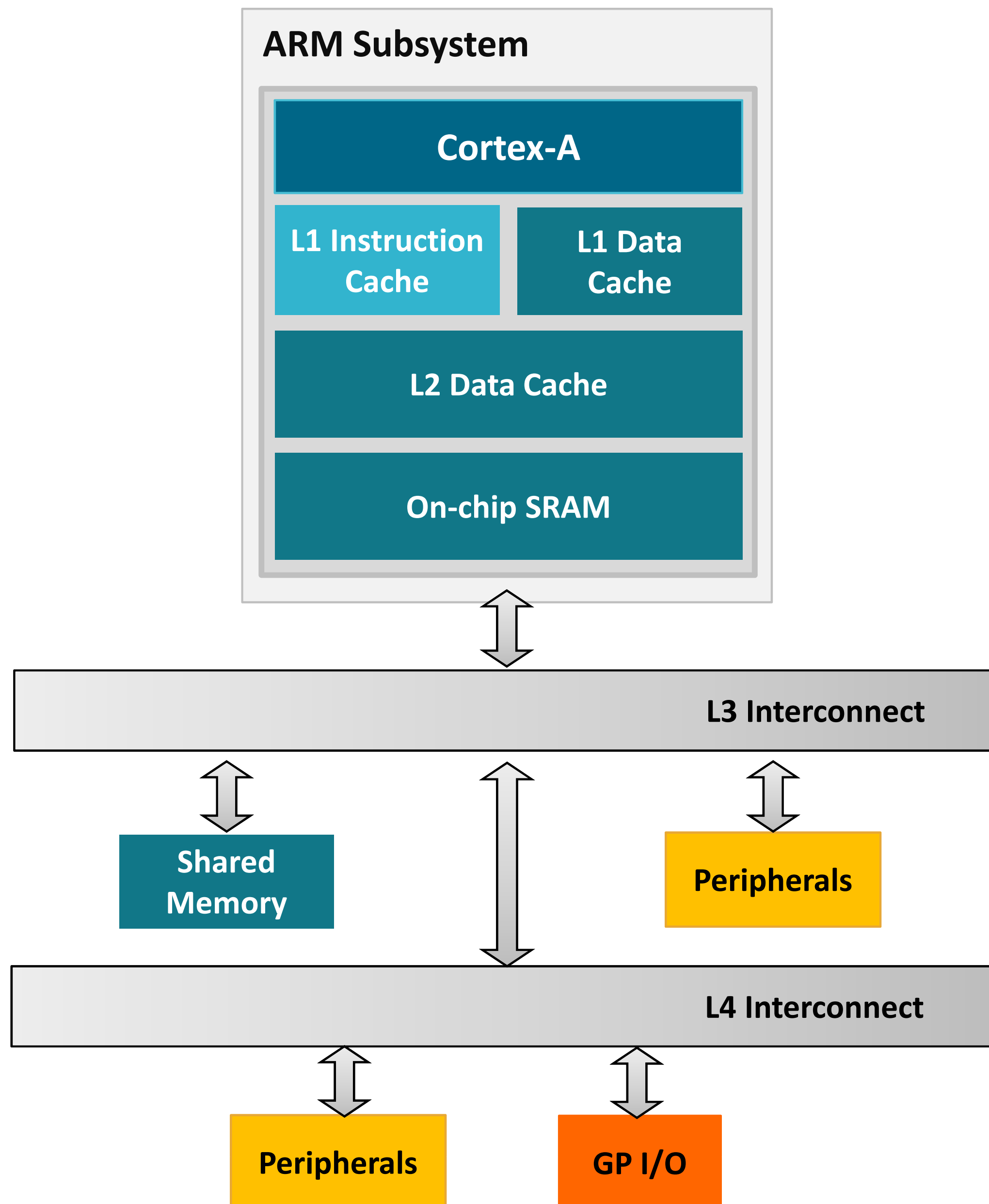
Agenda

- PRU Hardware Overview
- PRU Firmware Development
- Linux Drivers Introduction

PRU Hardware Overview

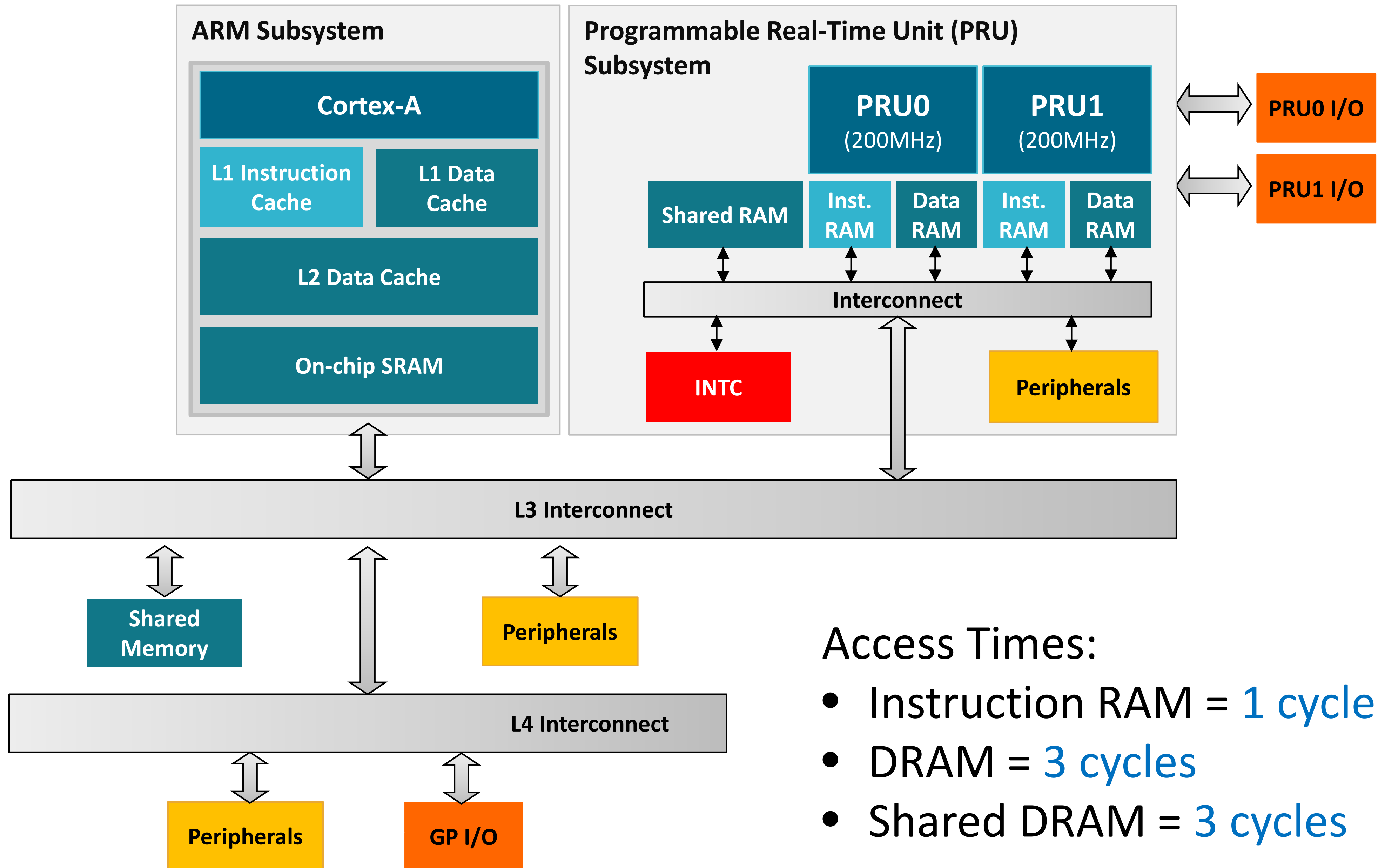
Building Blocks for PRU Development

ARM SoC Architecture



- L1 D/I caches:
  Single-cycle access
- L2 cache:
  Minimum latency of 8 cycles
- Access to on-chip SRAM:
  20 cycles
- Access to shared memory over L3 Interconnect:
  40 cycles

ARM + PRU SoC Architecture

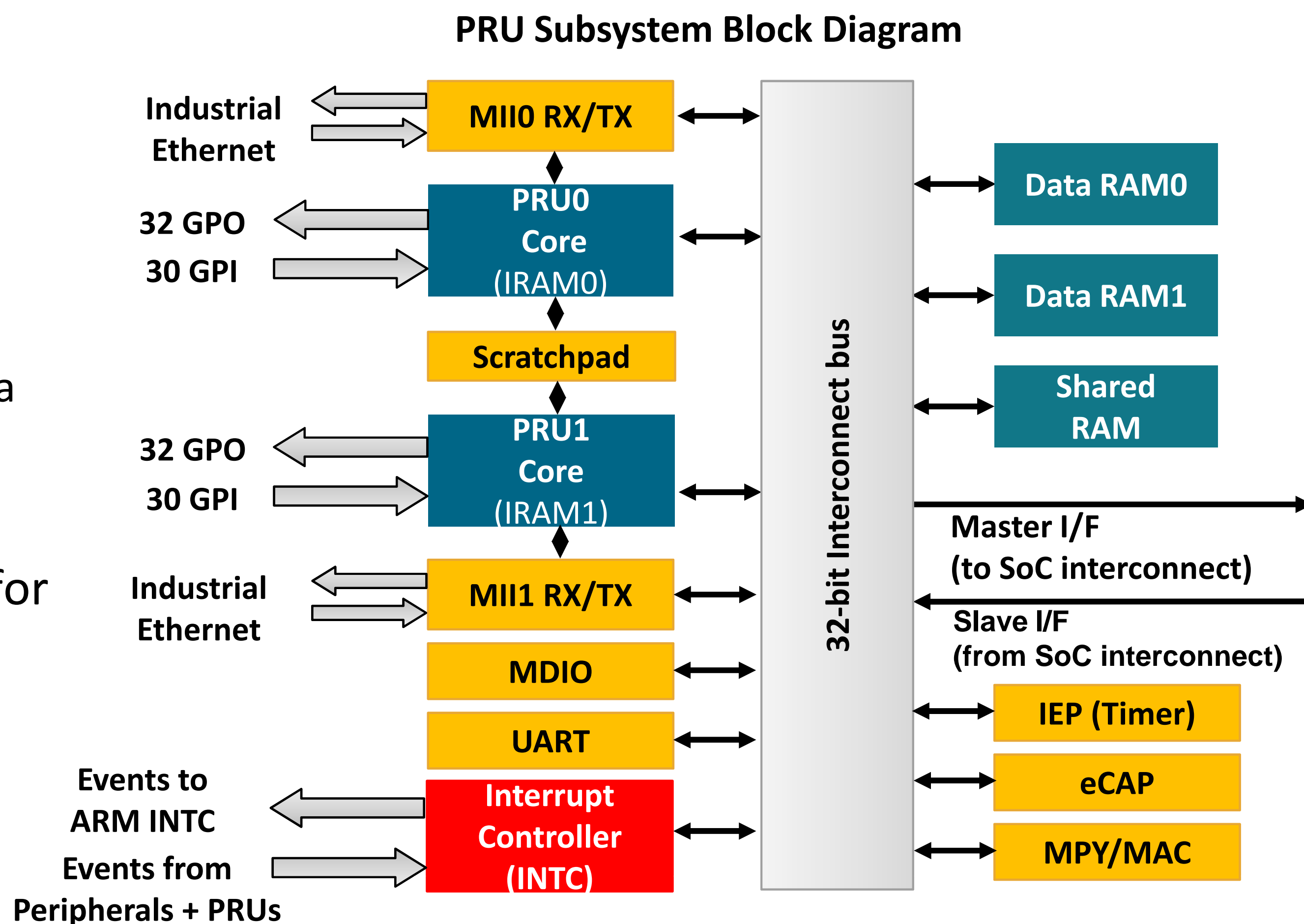


Access Times:

- Instruction RAM = 1 cycle
- DRAM = 3 cycles
- Shared DRAM = 3 cycles

Programmable Real-Time Unit (PRU) Subsystem

- Programmable Real-Time Unit (PRU) is a low-latency microcontroller subsystem.
- Two independent PRU execution units:
  - 32-Bit RISC architecture
  - 200MHz; 5ns per instruction
  - Single cycle execution; No pipeline
  - Dedicated instruction and data RAM per core
  - Shared RAM
- Includes Interrupt Controller for system event handling
- Fast I/O interface: Up to 30 inputs and 32 outputs on external pins per PRU unit.

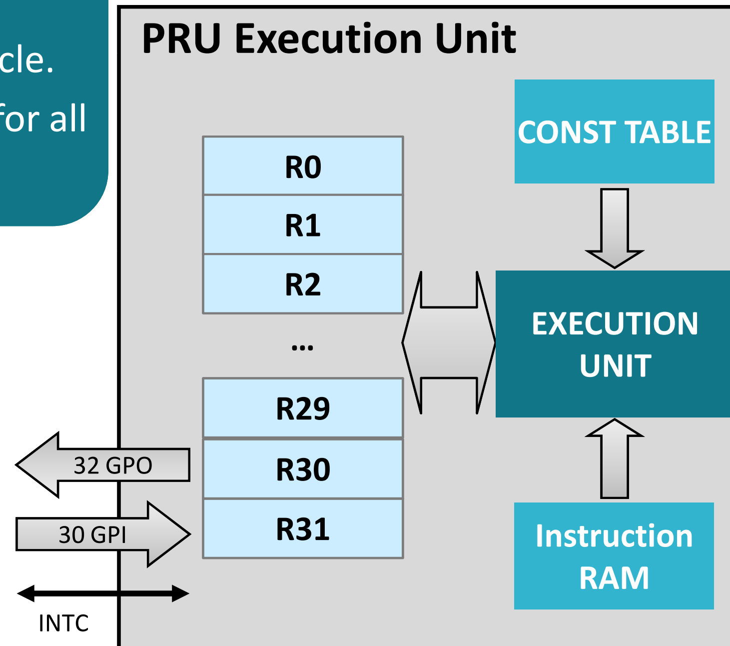


Now let's go a little deeper…

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

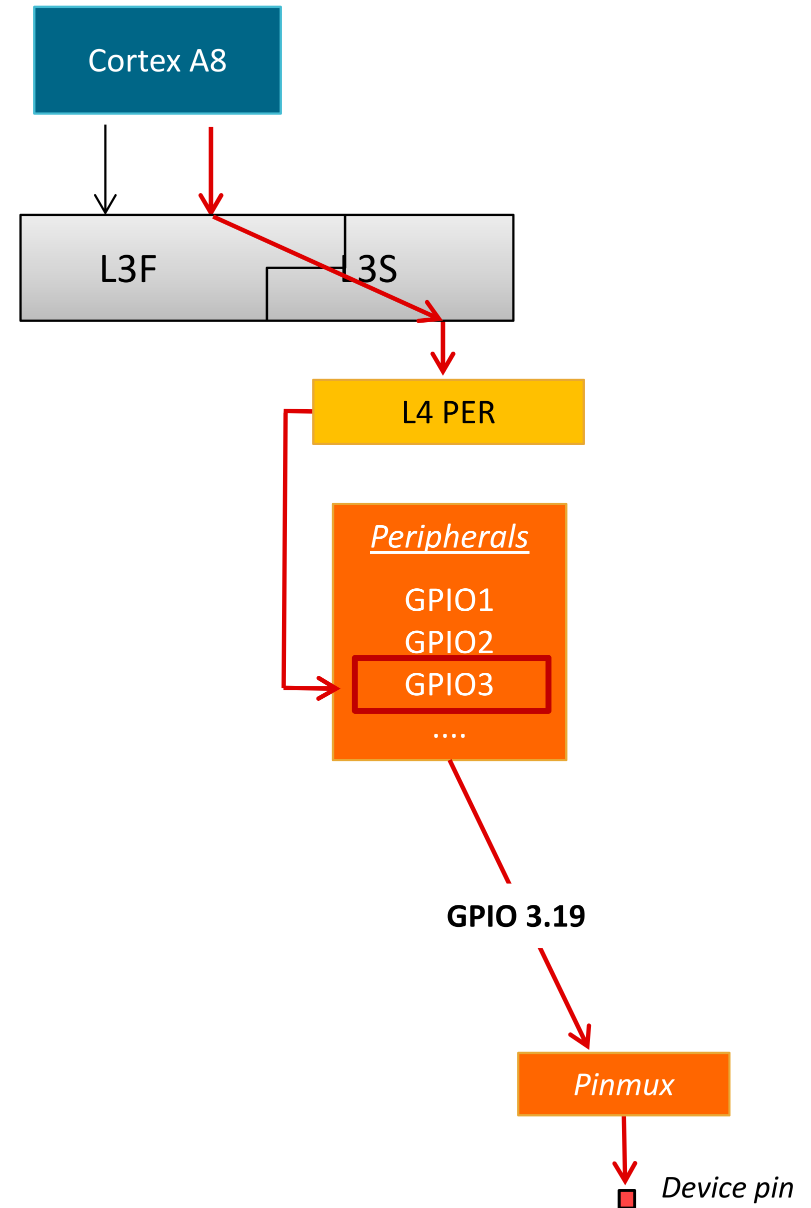
Special Registers (R30 and R31)

- R30
  - Write: 32 GPO
- R31
  - Read: 30 GPI + 2 Host Int status
  - Write: Generate INTC Event

Instruction RAM

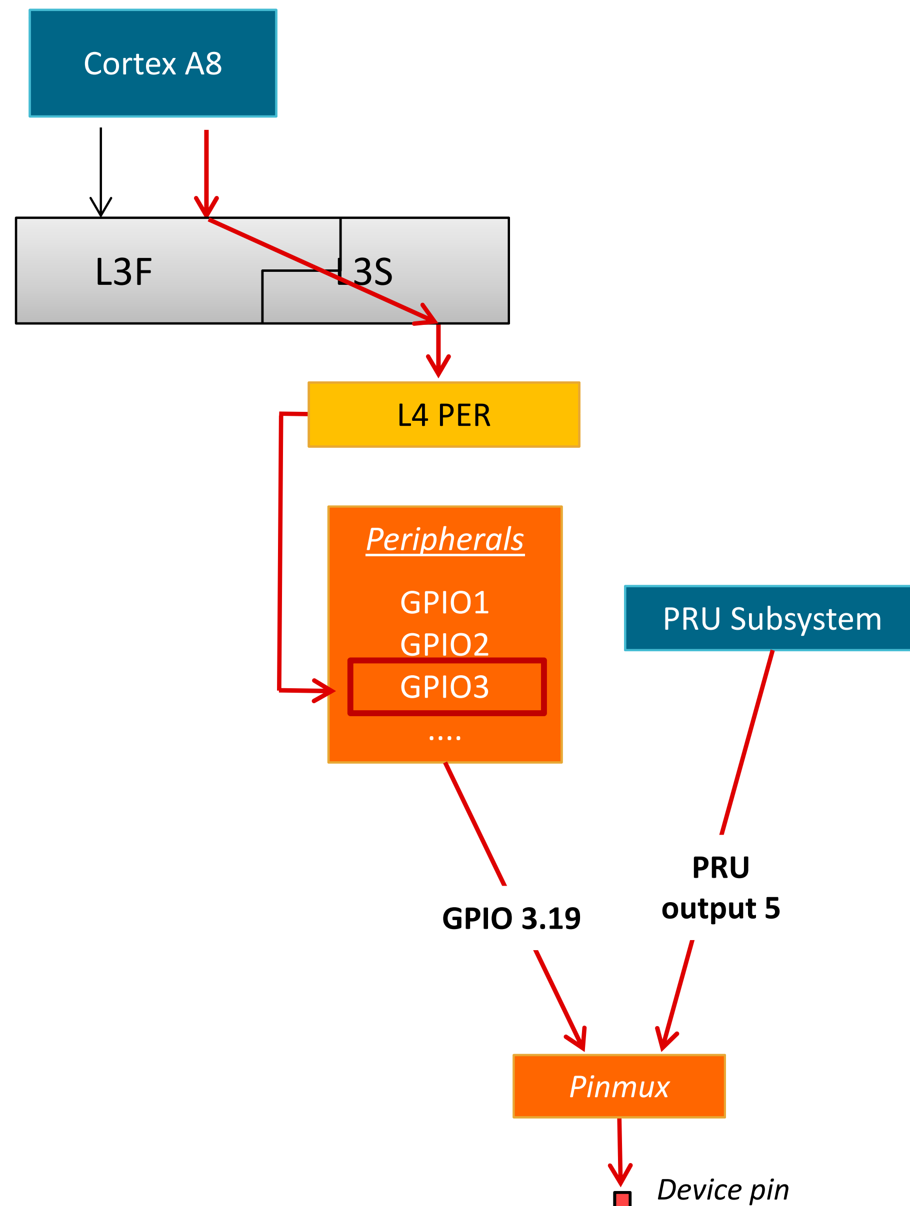
- Typical size is a multiple of 4KB (or 1K Instructions)
- Can be updated with PRU reset

Fast I/O Interface



Fast I/O Interface

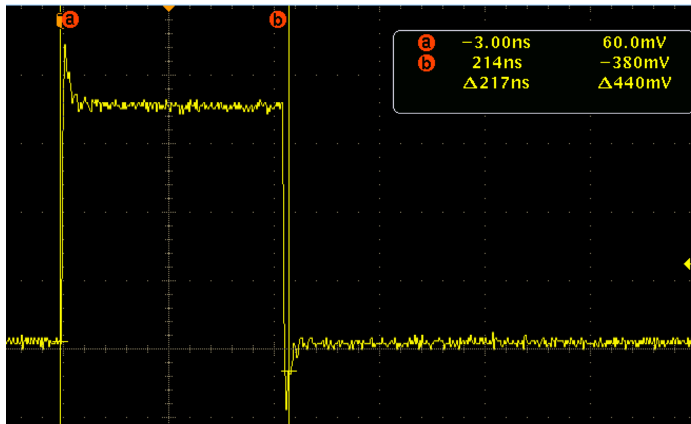
- Reduced latency through direct access to pins:
  - Read or toggle I/O within a single PRU cycle
  - Detect and react to I/O event within two PRU cycles
- Independent general purpose inputs (GPIs) and general purpose outputs (GPOs):
  - PRU R31 directly reads from up to 30 GPI pins.
  - PRU R30 directly writes up to 32 PRU GPOs.
- Configurable I/O modes per PRU core:
  - GP input modes:
    - Direct input
    - 16-bit parallel capture
    - 28-bit shift
  - GP output modes:
    - Direct output
    - Shift out



GPIO Toggle: Bench Measurements

ARM GPIO Toggle:

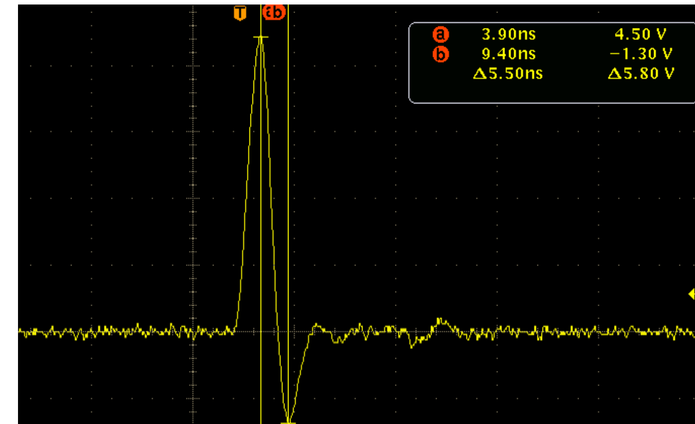
```
int main(){

    // Configure GPIO module, pinmuxing, etc.

    // Toggle system-level GPIO 3.19 from ARM core
    BitToggle(GPIO_INSTANCE_ADDRESS+GPIO_SETDATAOUT,
              GPIO_INSTANCE_ADDRESS+GPIO_CLEARDATAOUT);

    while();
}

unsigned long BitToggle(unsigned long val1, unsigned long val2){
    asm(
        " mov r2, #0x00080000" "\n\t"
        "str     r2,[r0]" "\n\t"        // Set GPIO 3.19
        "str     r2,[r1]" "\n\t"        // Clear GPIO 3.19
    );
    return val1;
}
```



~200ns

PRU IO Toggle:

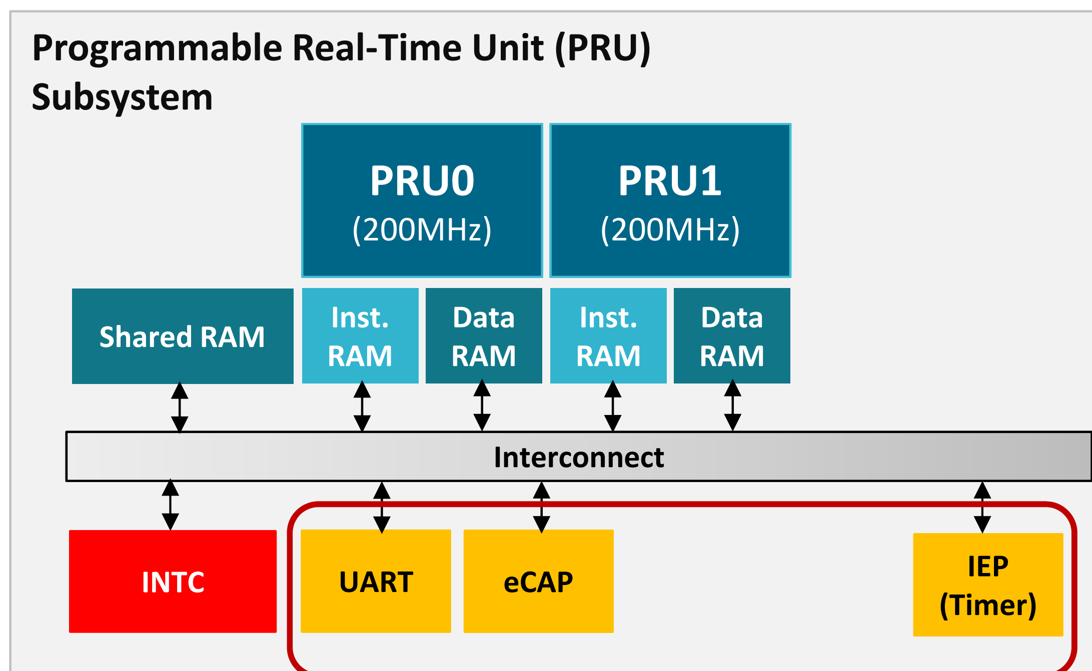
```
.origin 0
.entrypoint PRU_GPIO_TOGGLE

PRU_GPIO_TOGGLE:

    // Set PRU GPO 5
    SET       R30, R30, 5

    // Clear PRU GPO 5
    CLR       R30, R30, 5

    HALT
```



~5ns = ~40x Faster

Integrated Peripherals

- Provide reduced PRU read/write access latency compared to external peripherals
- No need for local peripherals to go through external L3 or L4 interconnects
- Can be used by PRU or by the ARM as additional hardware peripherals on the device
- Integrated peripherals:
  - PRU UART
  - PRU eCAP
  - PRU IEP (Timer)



PRU Read Latencies: Local vs Global Memory Map

The PRU directly accessing internal MMRs (Local MMR Access) is faster than going through the L3 interconnects (Global MMR Access).

	Local MMR Access *(PRU cycles @ 200MHz)*	Global MMR Access *(PRU cycles @ 200MHz)*
PRU R31 (GPI)	1	N/A
PRU CTRL	4	36
PRU CFG	3	35
PRU INTC	3	35
PRU DRAM	3	35
PRU Shared DRAM	3	35
PRU ECAP	4	36
PRU UART	14	46
PRU IEP	12	44

Note: Latency values listed are "best-case" values.

PRU "Interrupts"

- The PRU does not support asynchronous interrupts:
  - However, specialized h/w and instructions facilitate efficient polling of system events.
  - The PRU-ICSS can also generate interrupts for the ARM, other PRU-ICSS, and sync events for EDMA.
- From UofT CSC469 lecture notes, "*Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring.*
  - *Interrupts win if processor has other work to do and event response time is not critical*
  - *Polling can be better if processor has to respond to an event ASAP*"
- Asynchronous interrupts can introduce jitter in execution time and generally reduce determinism. The PRU is optimized for highly deterministic operation.

Sitara Device Comparison

Features	AM18x/ OMAPL138	AM335x	AM437x		AM571x	AM572x (PG1.1)
	PRUSS	PRU-ICSS1	PRU-ICSS1	PRU-ICSS0	2 x PRU-ICSS	2 x PRU-ICSS
PRU core version	1	3	3	3	3	3
Number of PRU cores (per subsystem)	2	2	2	2	2	2
Max frequency	CPU freq / 2	200 MHz	200 MHz	200 MHz	200 MHz	200 MHz
IRAM size (per PRU core)	4 KB	8 KB	12 KB	4 KB	12 KB	12 KB
DRAM size (per PRU core)	512 B	8 KB	8 KB	4 KB	8 KB	8 KB
Shared DRAM size (per subsystem)	--	12 KB	32 KB	--	32KB	32KB
General purpose input (per PRU core)	Direct	Direct; or 16-bit parallel capture; or 28-bit shift	Direct; or 16-bit parallel capture; or 28-bit shift; or 3ch EnDat 2.2; or 9ch Sigma Delta	Direct; or 16-bit parallel capture; or 28-bit shift; or 3ch EnDat 2.2; or 9ch Sigma Delta	Direct; or 16-bit parallel capture; or 28-bit shift; or 3ch EnDat 2.2; or 9ch Sigma Delta	Direct; or 16-bit parallel capture; or 28-bit shift
General purpose output (per PRU core)	Direct	Direct; or Shift out	Direct; or Shift out	Direct; or Shift out	Direct; or Shift out	Direct; or Shift out
GPI Pins (PRU0, PRU1)	30, 30	17, 17	13, 0	20, 20	21*, 21	21, 21
GPO Pins (PRU0, PRU1)	32, 32	16, 16	12, 0	20, 20	21*, 21	21, 21
MPY/MAC	N	Y	Y	Y	Y	Y
Scratchpad	N	Y (3 banks)	Y (3 banks)	N	Y (3 banks)	Y (3 banks)
CRC16/32	0	0	2	2	2	0
INTC	1	1	1	1	1	1
Peripherals	n/a	Y	Y	Y	Y	Y
UART	0	1	1	1	1	1
eCAP	0	1	1	no connect	1	1
IEP	0	1	1	no connect	1	1
MII_RT	0	2	2	no connect	2	2
MDIO	0	1	1	no connect	1	1
Simultaneous protocols	1	1	2**		2	

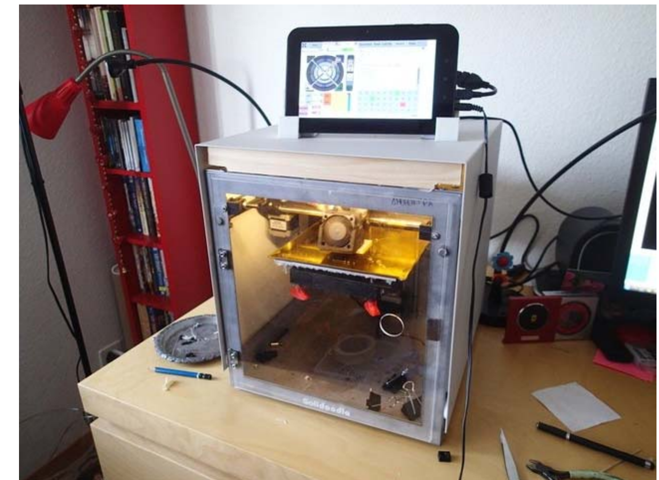
Examples of how people have used the PRU…

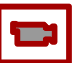
Use Case Examples

- Industrial Protocols
- ASRC
- 10/100 Switch
- Smart Card
- DSP-like functions
- Filtering
- FSK Modulation
- LCD I/F
- Camera I/F
- RS-485
- UART
- SPI
- Monitor Sensors
- I2C
- Bit banging
- Custom/Complex PWM
- Stepper motor control

Not all use cases are feasible on PRU

- Development complexity
- Technical constraints (i.e. running Linux on PRU)





Development Complexity

PRU Firmware Development

Building Blocks for PRU Development

TI PRU Code Generation Tools (CGT): C Compiler

C Compiler

- Developed and maintained by TI CGT team; Remains very similar to other TI compilers
- Full support of C/C++
- Adds PRU-specific functionality:
  - Can take advantage of PRU architectural features automatically
  - Contains several intrinsics: A list can be found in Compiler documentation
- Full instruction-set assembler for hand-tuned routines

For more information, visit http://www.ti.com/lit/ug/spruhv7/spruhv7.pdf.

TI PRU CGT Assembly vs C

- Advantages of coding in Assembly over C:
  - Code can be tweaked to save every last cycle and byte of RAM
  - No need to rely on the compiler to make code deterministic
  - Easily make use of scratchpad

- Advantages of coding in C over Assembly:
  - More code reusability
  - Can directly leverage kernel headers for interaction with kernel drivers
  - Optimizer is extremely intelligent at optimizing routines
    - "Accelerating" math via MAC unit, implementing LOOP instruction, etc.
  - Not mutually exclusive; Inline Assembly can be easily added to a C project

PRU Register Header Files

PRU Register Headers

- Created to make accessing a register easier: Register names match those in documentation
- Code Completion feature in CCS automatically lists all members
- Developed to allow a user to program at the register-level or at a bit-field level
  - Note that bit-field accesses could potentially cause some issues with other C compilers (e.g., gcc), but register-level should not.
- PRU cregister mechanism used to leverage constants table when possible
- Currently provides definitions for the following:
  - PRU INTC
  - PRU Config
  - PRU IEP
  - PRU Control
  - PRU ECAP
  - PRU UART

PRU Register Headers Layout

```
/* PRU_CFG_SYSCFG register bit field */
union {
    volatile uint32_t SYSCFG;

    volatile struct{
        unsigned IDLE_MODE : 2;
        unsigned STANDBY_MODE : 2;
        unsigned STANDBY_INIT : 1;
        unsigned SUB_MWAIT : 1;
        unsigned rsvd6 : 26;
    } SYSCFG_bit;
} ; // 0x4
```

- **Excerpt from pru_cfg.h**
  - **Access register directly CT_CFG.SYSCFG**
  - **Or access specific bitfields CT_CFG.SYSCFG_bit.STANDBY_INIT**

- **Example of how to use in C file**
  - **#include the specific header**
  - **Map the constant table entry to register structures**
  - **Access registers or fields**

```
#include <stdint.h>
#include <pru_cfg.h>
```

```
/* Mapping Constant table register to variable */
volatile pruCfg CT_CFG __attribute__((cregister("PRU_CFG",near) , peripheral));
```

```
/* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;
```

Texas Instruments

Development and Debug Options

Development Within CCS

- In CCS
  - Download and install PRU CGT package via App Center.
  - Open or create new PRU projects just like with any other device.
  - Code completion helps make register accesses easier.

- The Downside
  - It is more difficult to debug while Linux kernel and user application is also running concurrently.

Development Outside of CCS

- Outside of CCS
  - Code in your favorite text editor, build via command line
    - Linux and Windows packages available
  - May be easier to script/automate different processes (build or otherwise)

- The Downside
  - Can be difficult to debug PRU code
  - Lacks code completion

Debug

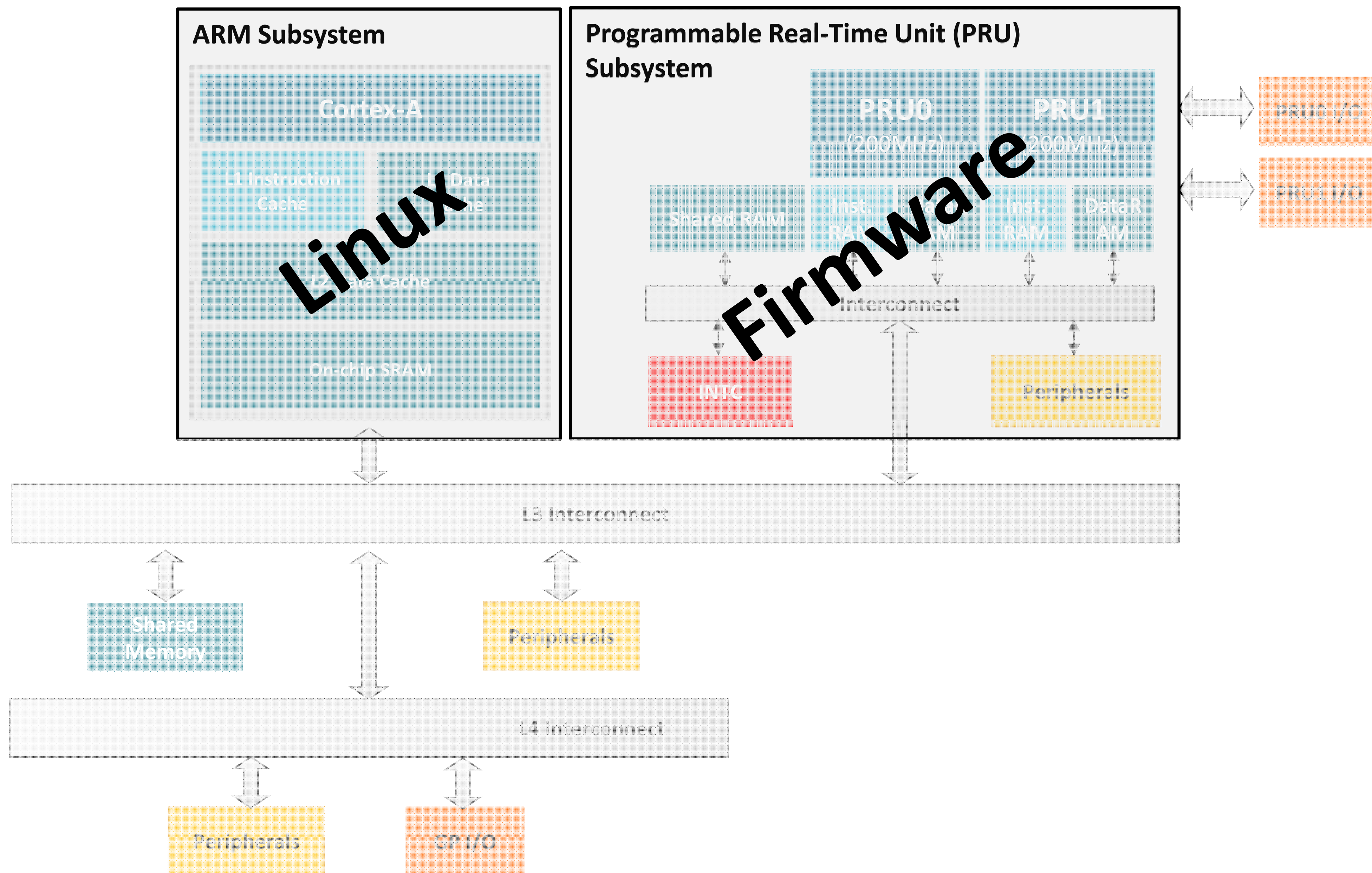
- In CCS
  - Easy to view register and variable contents
  - Access to breakpoints and simply stepping mechanism
- Outside CCS
  - Minimal debug control, but some debugfs control provided through remoteproc
  - Start, halt, single-stepping is all console-based
    - Clunky when done by hand, but can potentially be scripted

Linux Drivers Introduction

Building Blocks for PRU Development

Texas Instruments

ARM + PRU SoC *Software* Architecture



What Do We Need Linux to Do?

- Load the Firmware
- Manage resources (memory, CPU, etc.)
- Control execution (start, stop, etc.)
- Send/receive messages to share data **AND**
- Synchronize through events (interrupts)
- These services are provided through a combination of remoteproc/rpmsg + virtio transport frameworks

For More Information

- Visit the PRU-ICSS Wiki: http://processors.wiki.ti.com/index.php/PRU-ICSS
- Download the PRU tools:
  - PRU Software Package: http://www.ti.com/tool/pru-swpkg
  - PRU CGT (Code Gen Tools): http://processors.wiki.ti.com/index.php/Download_CCS
  - Linux drivers for interfacing with PRU: http://www.ti.com/lsds/ti/tools-software/processor_sw.page
- Order the PRU Cape: http://www.ti.com/tool/PRUCAPE
- For questions about this training, refer to the E2E Sitara Processors Forum: https://e2e.ti.com/support/arm/sitara_arm