06-3 Real-Time BeagleBone Interfacing

Chapter 13 – Exploring the BeagleBone

RealTime BeagleBone

- Real-time systems guarantee a response within a specified time
- *Hard real-time* systems are when systems fail if a deadline is missed (think: braking systems)
- *Soft real-time* systems are when a missed deadline may be reduced quality (think: streaming video)
- Mainline Linux Kernels typically meet soft real-time
- Other processes may be running when the real-time event occurs

Real-time Kernels

- Running a GPIO shell script can result in lots of *jitter*

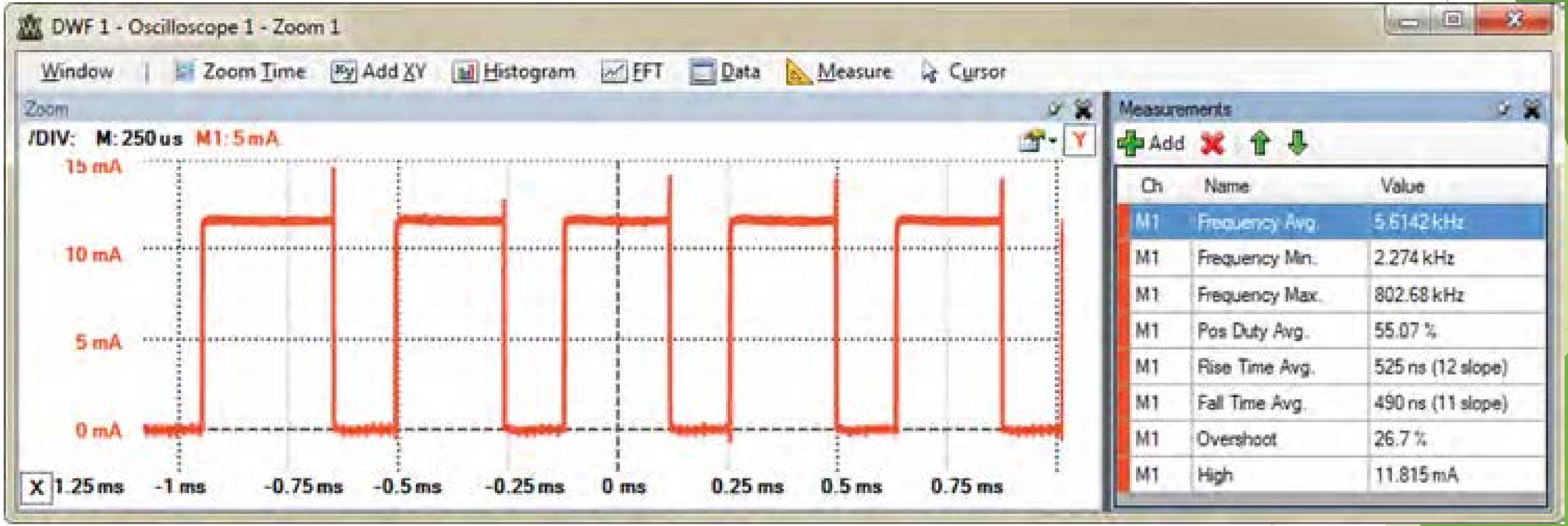
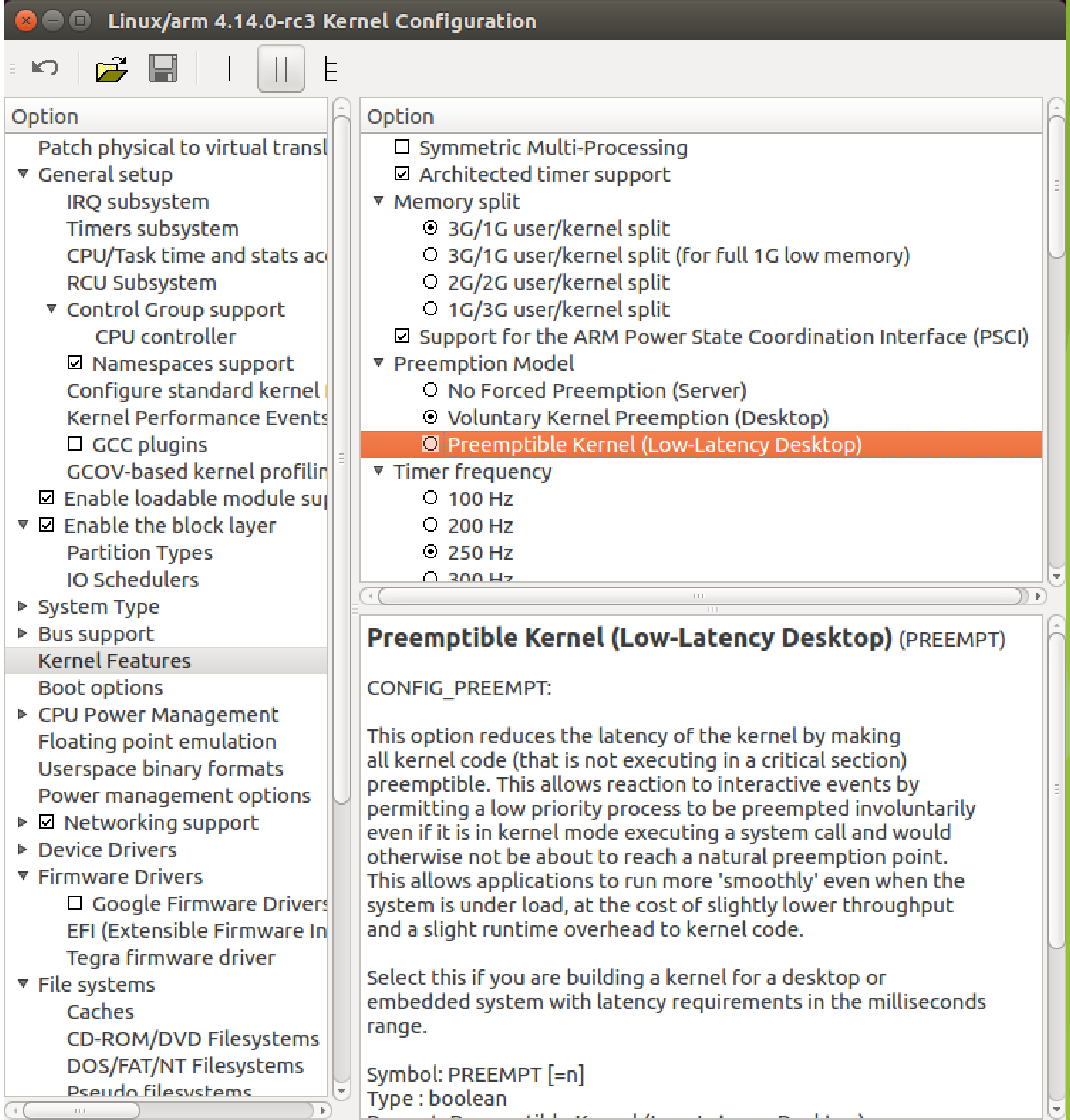


Figure 6-3 EBB

Real-time Kernels

- Using mmap() can help, but
- Linux is a *nonpreemptive* OS
- The kernel can be compiled using **`CONFIG_PREEMPT=y`**
- You can load precompiled RT kernels...



Precompiled Kernels

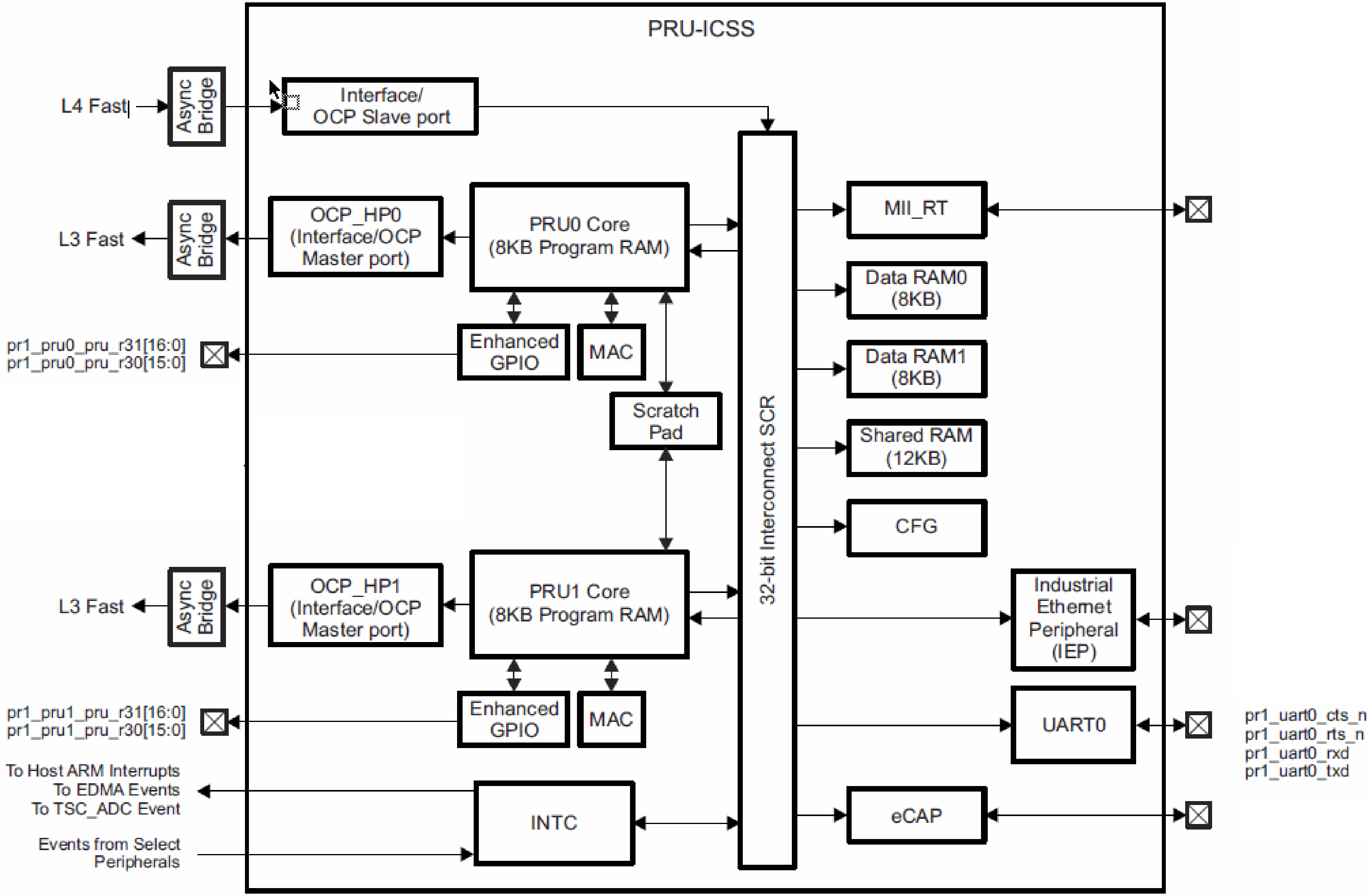
```
bone$ apt-cache pkgnames | grep linux-image-4.4
linux-image-4.4.89-armv7-lpae-x8
linux-image-4.4.89-armv7-rt-x14
linux-image-4.4.89-armv7-x14
linux-image-4.4.89-bone19
linux-image-4.4.89-bone-rt-r19
linux-image-4.4.89-ti-r130
linux-image-4.4.89-ti-rt-r130
```

Real-time Hardware

- You can get a cape with an FPGA on it such as the Valent F(x) LOGi-Bone

Programmable Real-Time Unit (PRU)

- The Bone comes with two 32-bit 200 MHz RISC cores, PRUs
- Enhanced GPIO: fast GPIO on P8/P9 headers
- 32x32 Multiplier/Accumulator (MAC)
- Interrupt controller (INTC)
- UART0: 192 MHz on P8/P9



TECH DAYS
Texas Instruments

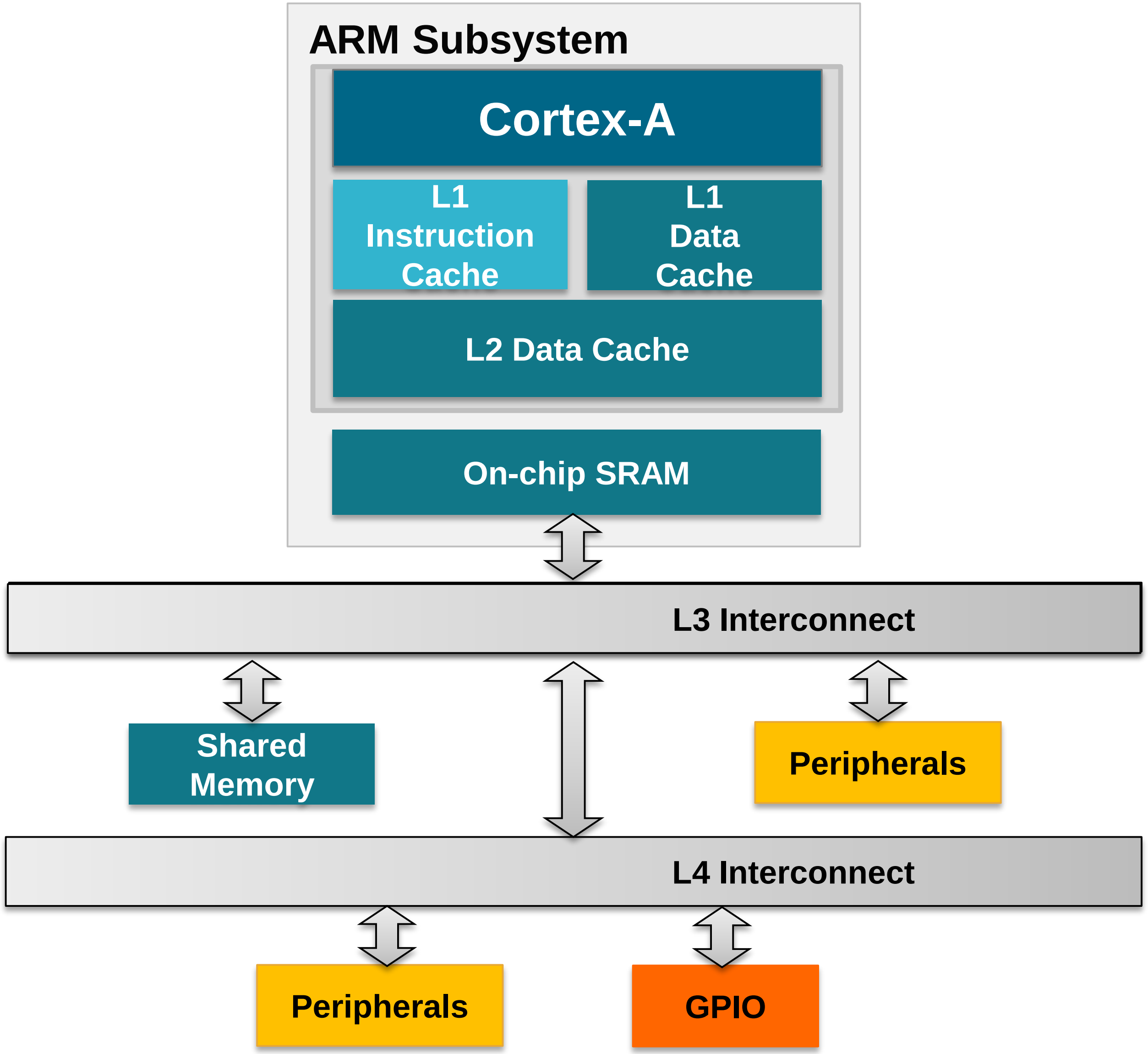
Building Blocks for Programmable Real-Time Unit (PRU) Programming & Development

Jason Reeder & Melissa Watkins

Agenda

- PRU Hardware Overview
- PRU Firmware Development
- ~~Linux Drivers Introduction~~
- ~~PRU Application Examples~~
- ~~Getting Started with the PRU~~

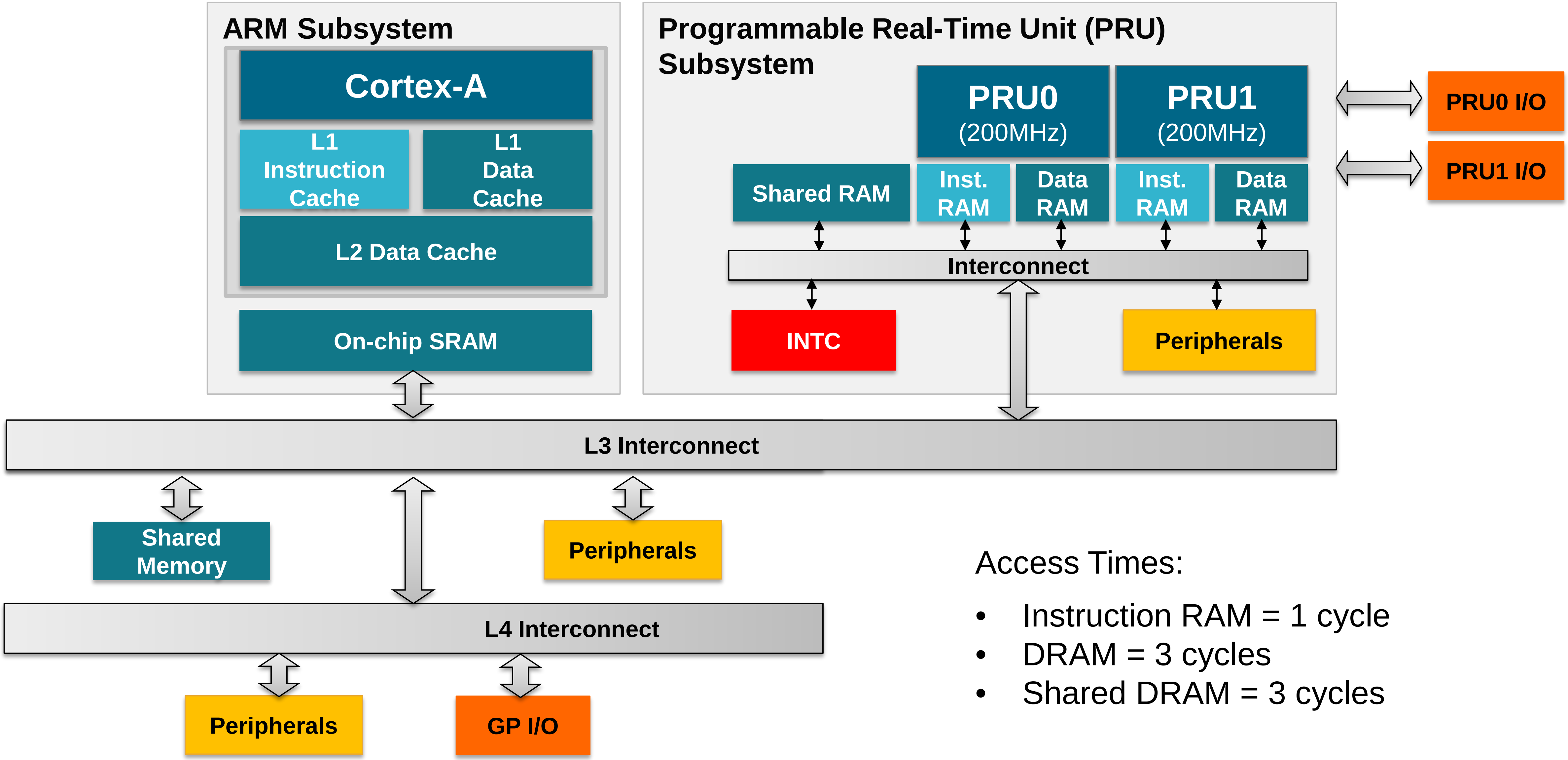
ARM SoC Architecture



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

ARM + PRU SoC Architecture

TECH DAYS
Texas Instruments



Access Times:

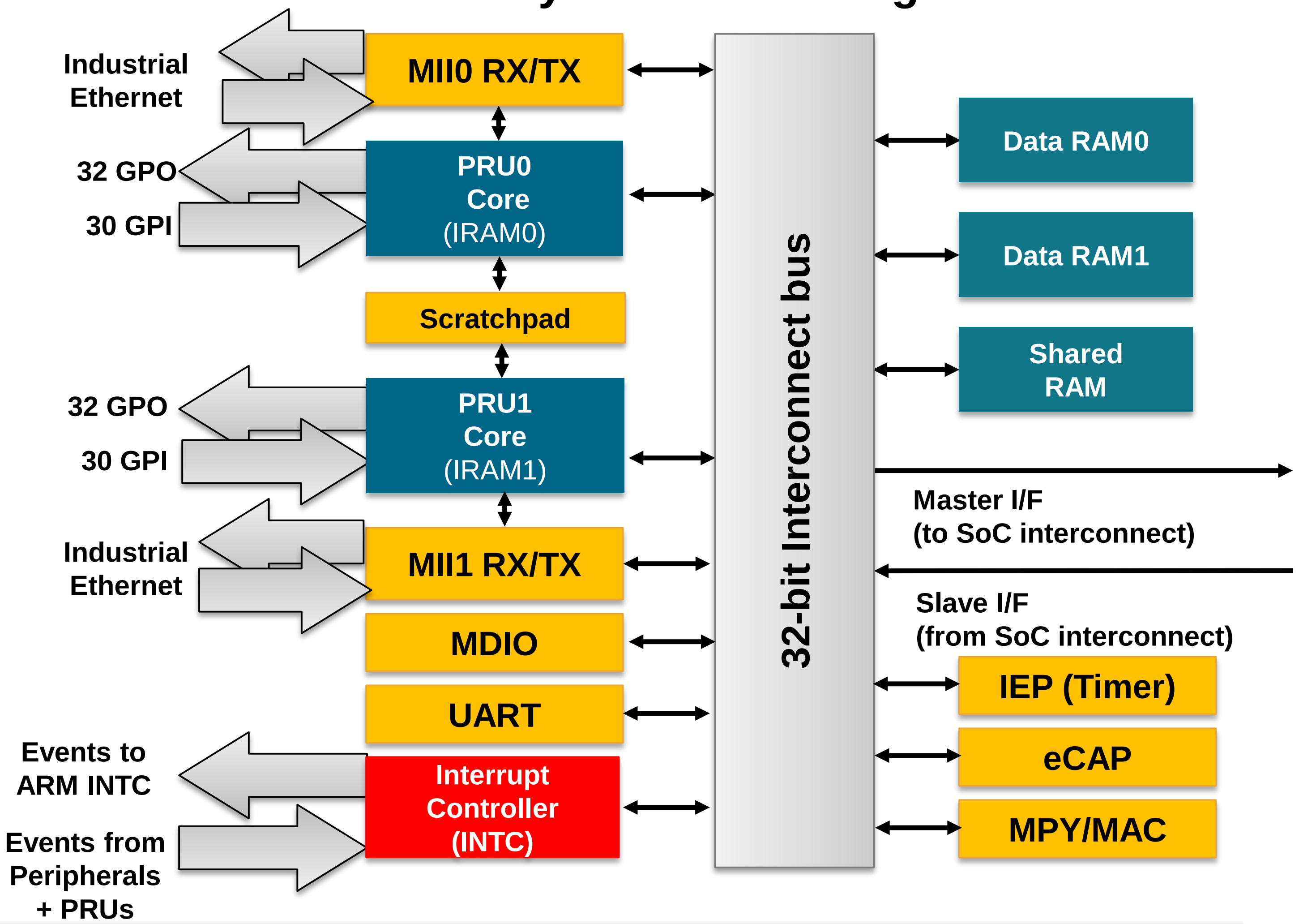
- Instruction RAM = 1 cycle
- DRAM = 3 cycles
- Shared DRAM = 3 cycles

Now let’s go a little deeper…

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

PRU Subsystem Block Diagram

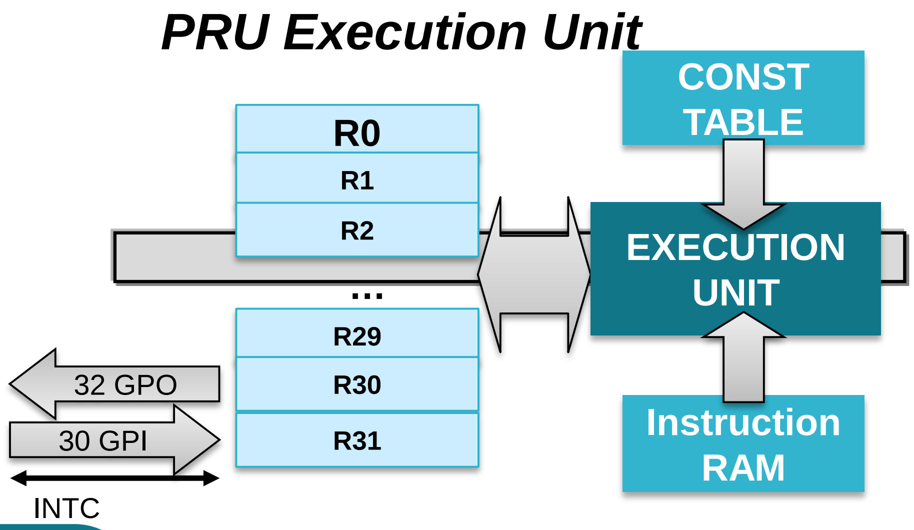


PRU Functional Block Diagram

TECH DAYS
Texas Instruments

Constant Table

- Saves PRU cycles by providing frequently used peripheral base addresses



Execution Unit

- Logical, arithmetic, and flow control instructions
- Scalar, no Pipeline, Little Endian
- Single-cycle execution

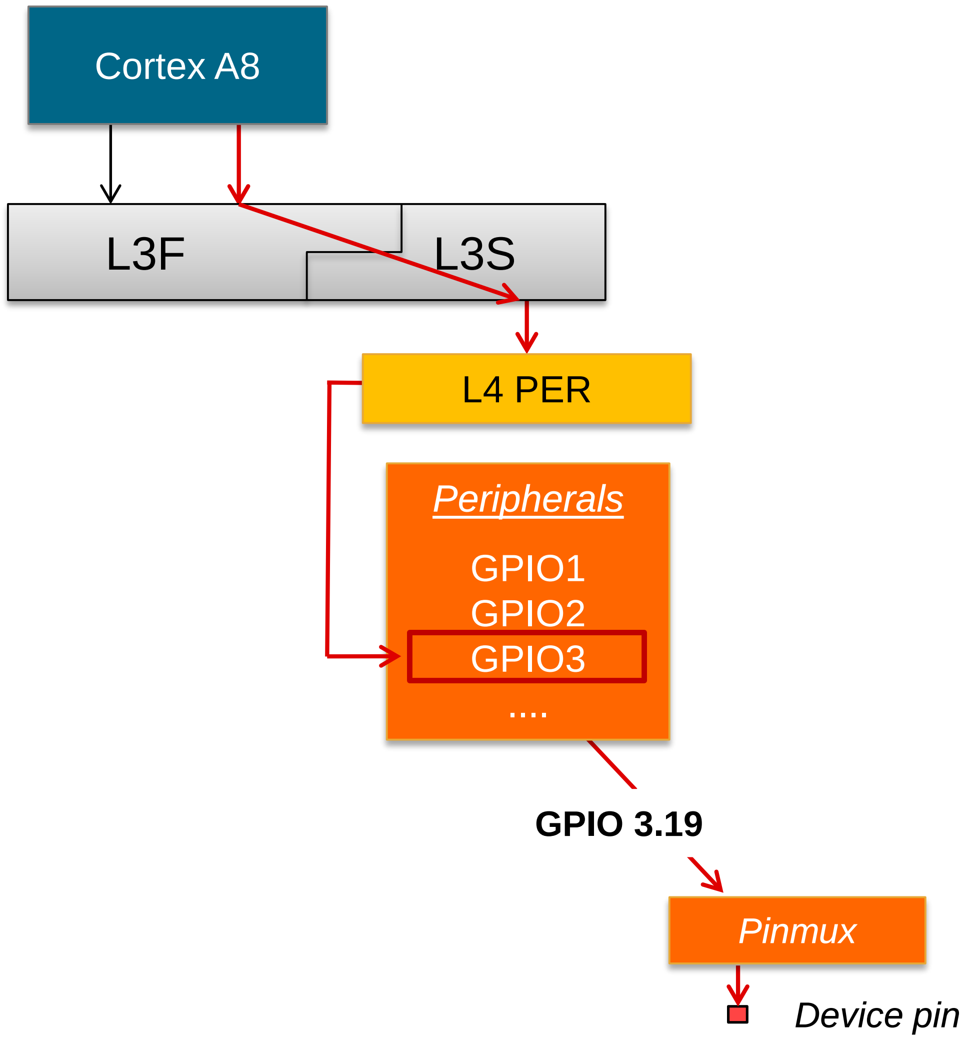
Special Registers (R30 and R31)

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

Instruction RAM

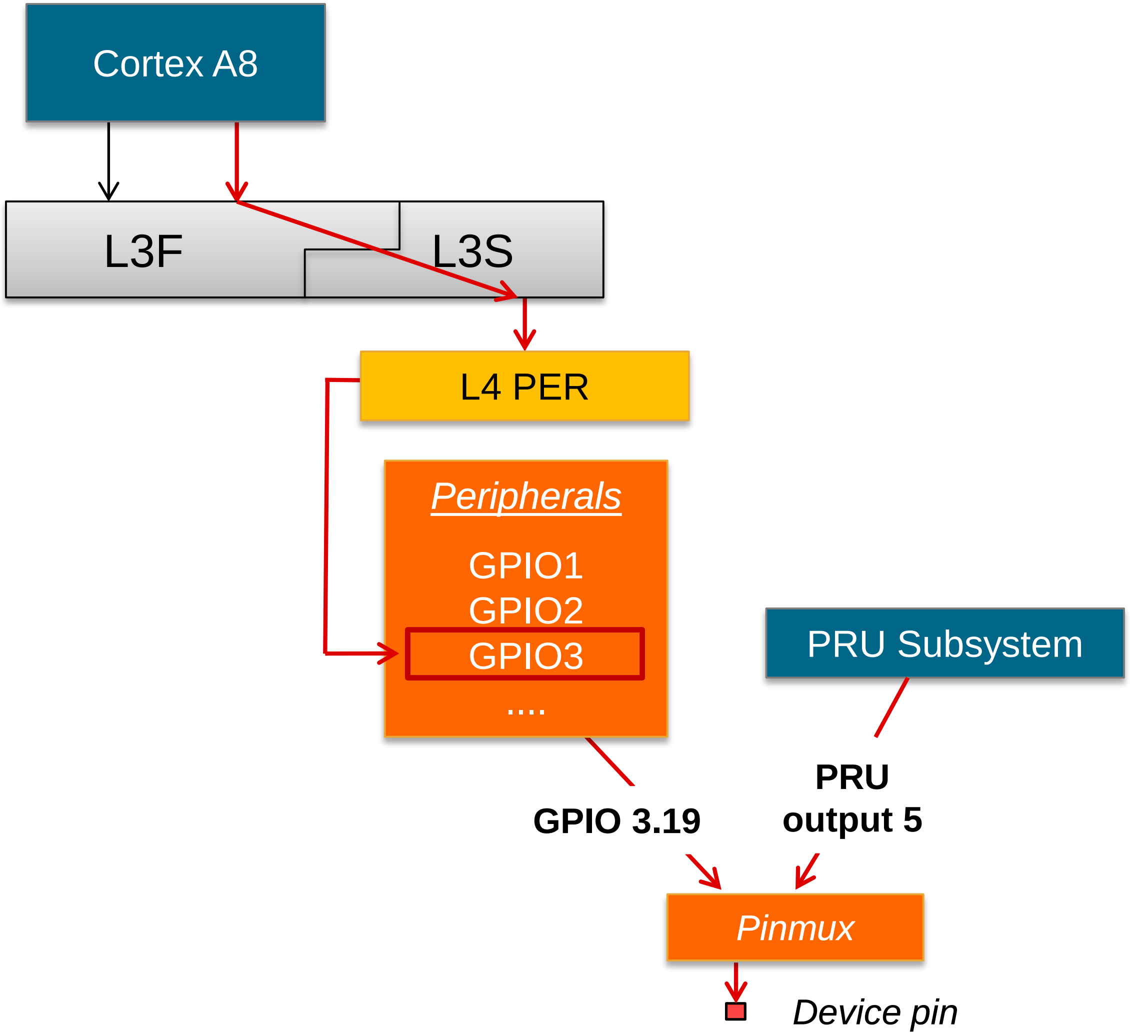
- Typical size is a multiple of 4KB (or 1K Instructions)
- Can be updated with PRU reset

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 connect
    - 16-bit parallel capture
    - 28-bit shift
  - GP output modes
    - Direct connect
    - Shift out



GPIO Toggle: Bench measurements

ARM GPIO Toggle:

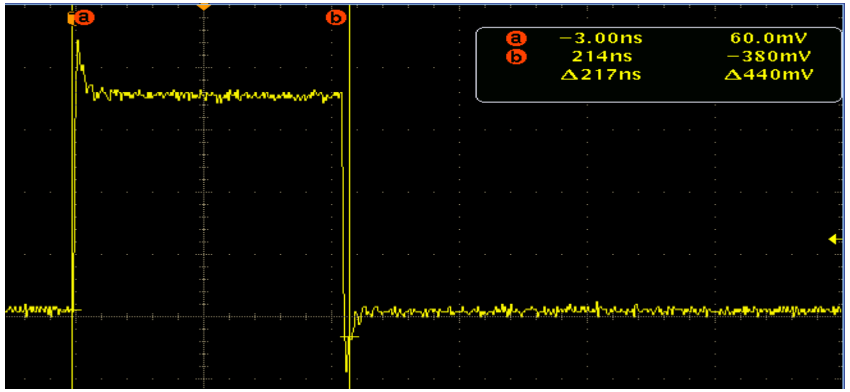
```
int main(){

    // 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:

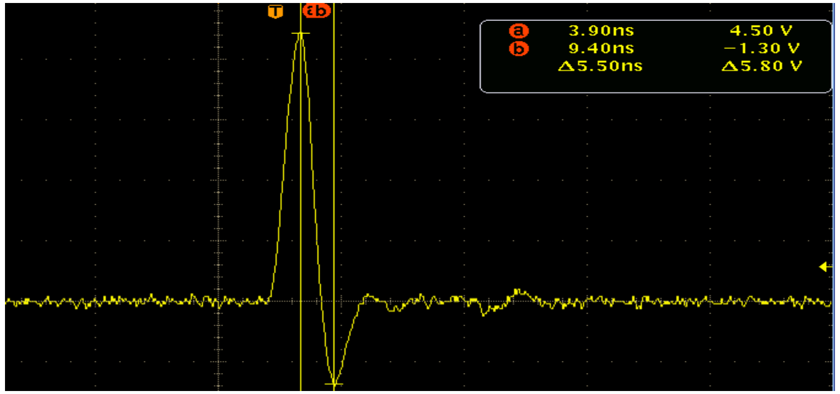
```
.origin 0
.entrypoint PRU_GPIO_TOGGLE


PRU_GPIO_TOGGLE:

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

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

    HALT
```



~5ns = ~40x Faster

Examples from PRU Cookbook

- Getting started example is from https://markayoder.github.io/PRUCookbook/02start/start.html#_blinking_an_led
- Clone the repo

```
bone$ git clone https://github.com/MarkAYoder/PRUCookbook.git
```

- Find the example

```
bone$ cd PRUCookbook/docs/02start/code
```

- Set environmental variables

```
bone$ source setup.sh
PRUN=0
TARGET=hello
```

Which PRU to compile for

Which file to compile

Make sure remoteproc is running

```
bone$ lsmod | grep pru
pruss_soc_bus          16384  0
pru_rproc              28672  1
pruss                  16384  1 pru_rproc
pruss_intc             16384  1 pru_rproc
```

If you get the above response, you are good, move on

Make and run

```
bone$ make
-    Stopping PRU 0
[sudo] password for debian:
stop
CC   hello.c
LD   /tmp/pru0-gen/hello.obj
-   copying firmware file /tmp/pru0-gen/hello.out to
/lib/firmware/am335x-pru0-fw
-    Starting PRU 0
start
```

This is where the kernel looks for PRU code

- The USR3 LED should blink 5 times

P8 Header **pr1_pru1_pru_r31_11**

P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]	pr1_pru1_pru_r31_11	pr1_pru1_pru_r30_11
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn	
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn	
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn	
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn	
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn	
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn	
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn	
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn	
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1

P9 Header `pr1_pru1_pru_r31_11`

P9_25	107	0x9ac/1ac	117	GPIO3_21	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7
P9_26	96	0x980/180	14	UART1_RXD	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd
P9_27	105	0x9a4/1a4	115	GPIO3_19	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5
P9_28	103	0x99c/19c	113	SPI1_CS0	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3
P9_29	101	0x994/194	111	SPI1_D0	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1
P9_30	102	0x998/198	112	SPI1_D1	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2
P9_31	100	0x990/190	110	SPI1_SCLK	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0
P9_32				VADC			
P9_33				AIN4			
P9_34				AGND			
P9_35				AIN6			
P9_36				AIN5			
P9_37				AIN2			
P9_38				AIN3			
P9_39				AIN0			
P9_40				AIN1			
P9_41A	109	0x9b4/1b4	20	CLKOUT2	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16
P9_41B		0x9a8/1a8	116	GPIO3_20	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6
P9_42A	89	0x964/164	7	GPIO0_7	gpio0[7]	xdma_event_intr2	mmc0_sdwp
P9_42B		0x9a0/1a0	114	GPIO3_18	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4
P9_43				GND			
P9_44				GND			
P9_45				GND			
P9_46	cat		(Mode 7)	GND			
P9	$PINS	ADDR +	GPIO NO.	Name	Mode 7		

dmesg -Hw

```
[Oct 5 10:15] remoteproc remoteproc1: stopped remote processor 4a334000.pru
[  +0.176812] remoteproc remoteproc1: powering up 4a334000.pru
[  +0.000347] remoteproc remoteproc1: Booting fw image am335x-pru0-fw, size 32724
[  +0.000037] remoteproc remoteproc1: remote processor 4a334000.pru is now up
```

Programming the PRU - c

- You can use either C or assembly, Let’s do both

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

#define GPIO1 0x4804C000
#define GPIO_CLEARDATAOUT 0x190
#define GPIO_SETDATAOUT 0x194
#define USR0 (1<<21)
#define USR1 (1<<22)
#define USR2 (1<<23)
#define USR3 (1<<24)
unsigned int volatile * const GPIO1_CLEAR = (unsigned int *) (GPIO1 + GPIO_CLEARDATAOUT);
unsigned int volatile * const GPIO1_SET   = (unsigned int *) (GPIO1 + GPIO_SETDATAOUT);
```

Programming the PRU - c

- You can use either C or assembly, Let’s do both

```
void main(void) {
    int i;
    /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
    CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

    for(i=0; i<5; i++) {
        *GPIO1_SET = USR3;            // The USR3 LED on
        __delay_cycles(500,000,000/5);  // Wait 1/2 second

        *GPIO1_CLEAR = USR3;
         __delay_cycles(500,000,000/5);
    }
    __halt();
}
```

The PRU Cookbook

- https://github.com/MarkAYoder/PRUCookbook

MarkAYoder/PRUCookbook: This

GitHub, Inc. [US] | https://github.com/MarkAYoder/PRUCookbook

Apps | Rose | Travel | Courses | mp3 | ECE205 | Linux | Mandi | Advanced Databases | Schedule Lookup | mp3 | Linux | Mandi | Other bookmarks

339 commits | 1 branch | 0 releases | 2 contributors

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

MarkAYoder Used for testing matrices — Latest commit a55ea81 on Aug 28

docs	Used for testing matrices	a month ago
README.md	Updated	2 months ago
_config.yml	Added title	5 months ago
install.sh	Install instructions of asciidoctor-pdf	4 months ago

README.md

PRU Cookbook

Look in the *docs* folder so see the source for the examples.

Go to https://markayoder.github.io/PRUCookbook/ to see the text.

1. Outline

A cookbook for programming the PRUs in C using remoteproc and compiling on the Beagle

- Notes

1. Case Studies
   - ✓ Robotics Control Library
   - ✓ BeagleLogic
   - ✓ LEDscape
   - ✓ Falcon Player
   - ❏ MachineKit
   - ❏ ArduPilot
2. Getting started
   - ✓ Selecting a Beagle
   - ✓ Installing the Latest OS on Your Bone
   - ✓ Flashing a Micro SD Card
   - ✓ Cloud9 IDE
   - ✓ Getting Example Code
   - ✓ Blinking an LED
3. Running a Program; Configuring Pins
   - ✓ Getting Code Example Files

✓ dmesg –Hw

✓ prudebug - A Simple Debugger for the PRU

✓ UART

5. [Building Blocks – Applications]

   ✓ Memory Allocation

   ✓ Auto Initialization of Built in LED Triggers

   ✓ PWM Generator

   ✓ Controlling the PWM Frequency

   ✓ Loop Unrolling for Better Performance

   ✓ Making All the Pulses Start at the Same Time

   ✓ Adding More Channels via PRU 1

   ✓ Sychronziing Two PRUs

   ✓ Reading an Input at Regular Intervals

   ✓ Analog Wave Generator

   ✓ WS2812 (NeoPixel) driver

   ✓ Setting NeoPixels to Different Colors

   ✓ Controlling Arbitrary LEDs

   ✓ Controlling NeoPixels Through a Kernel Driver

   ✓ RGB LED Matrix - No Integrated Drivers

   ✓ Compiling and Inserting rpmsg_pru

6. Accessing More I/O [2]

   ✓ /boot/uEnv.txt to access P8 I/O

1.3. PWM Generator

One of the simplest things a PRU can to is generate a simple signals starting with a single channel PWM that has a fixed frequency and duty cycle and ending with a multi channel PWM that the ARM can change the frequency and duty cycle on the fly.

Problem

I want to generate a PWM signal that has a fixed frequency and duty cycle.

Solution

The solution is fairly easy, but be sure to check the **Discussion** section for details on making it work.

pwm1.c shows the code.

pwm1.c

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

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
    uint32_t gpio;

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

    gpio = 0x0001;  // Select which pin to toggle.

    while (1) {
```

Programming the PRU - c

```c
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"
volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void) {
  uint32_t gpio; /* Clear SYSCFG[STANDBY_INIT] */
  CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

  gpio = 0x0001; // Select which pin to toggle.
    while (1) {
      __R30 |= gpio; // Set the GPIO pin to 1
      __delay_cycles(100000000);
      __R30 &= ~gpio; // Clearn the GPIO pin
      __delay_cycles(100000000);
     }
 }
```