Programmable Real-Time Unit

(PRU)

Topics

1) How can we do hard real-time on the BBG?
2) How can we get our code into the PRU?
3) How can we use GPIO with the PRU?

Hard Real-Time with BBG:
About the PRU

Our Definitions of Realtime

- Hard Realtime
  - User requires..
  - "Guaranteed Services"
    Mathematical/logical proof or exhaustive simulation required
  - Hard real-time is about..
- Soft Realtime
  - User only requires..

    (statistical analysis)
  - "Best effort Services"
    Ex: Average # missed deadline < 2 per minute.
  - Soft real-time is about..

Motivation

- Linux can do fine with soft real-time tasks
  - ~10s of ms accuracy/latency
- Hard real-time
  - Cannot use Linux: cannot
    ..
    but, Linux gives us great software power!
  - Analyze system and
    ..
    from soft real-time and non-real-time
- Hard real-time task solution
  - Run hard real-time tasks on a processor without Linux
    - Use external processor (Arduino)
    - Use internal microcontroller (PRU)

PRU

- Programmable Real-time Unit (PRU)
  - BBG's AM335x System on Chip (SoC) has
    2 integrated programmable microcontrollers
  - Run bare-metal programs

- Useful for hard real-time interactions, such as:
  - deterministic latency to respond to GPIO event
  - .. GPIO and ..
    (Ex: timing ultrasonic distance sensor readings)
  - .. (manual) protocol implementation:
    UART, I2C, SPI, Neo-pixel 1-wire protocol, ...
- It is called the
  PRU and Industrial Communication Subsystem (PRU-ICSS)

PRU Architecture

- 2 PRUs: PRU0, PRU1
  - 32-bit RISC processors, 200MHz
- Each PRU has:
  - own registers
  - PRU RAM: 8KB dedicated; 12KB shared
- Resources
  - Share RAM with Linux
  - "Enhanced" GPIO pins
    (fast in/out on some P8/P9 pins)
  - Peripheral access (UART, etc)
  - And more!
    (interrupts, OCP Port to access general hardware, ...)

Interface to PRU

- Linux's Remote Processor Framework
  - ..
  - called remoteproc
- (bbg)$ **ls /sys/class/remoteproc/**
  - 0: power management (Cortex M3 co-processor: Wakeup M3)
  - 1: PRU0 (32-bit RISC, 200MHz)
  - 2: PRU1 (32-bit RISC, 200MHz)
- remoteproc can load firmware from /lib/firmware into a remote processor:

```
(bbg)$ cd /sys/class/remoteproc/remoteproc1/
(bbg)$ echo 'stop'  | sudo tee ./state
(bbg)$ echo 'start' | sudo tee ./state
(bbg)$ cat ./state
```

Coding the PRU

Sample Program

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

// Delay 250ms (# cycles 200Mhz / 4)
#define DELAY_250_MS 50000000

volatile register uint32_t __R30;   // output GPIO register
volatile register uint32_t __R31;   // input GPIO register

// GPIO Output: P8_12 = pru0_pru_r30_14
//    = LEDDP2 (Turn on/off right 14-seg digit)
#define DIGIT_ON_OFF_MASK (1 << 14)

// GPIO Input: P8_15 = pru0_pru_r31_15
//    = JSRT (Joystick Right) on Zen Cape
#define JOYSTICK_RIGHT_MASK (1 << 15)
```

```c
void main(void)
{
    // Toggle digit on/off; slow down when RIGHT press
    while (true) {
        __R30 ^= DIGIT_ON_OFF_MASK;
        __delay_cycles(DELAY_250_MS);

        // Longer delay if pressed
        if (!(__R31 & JOYSTICK_RIGHT_MASK)) {
            __delay_cycles(DELAY_250_MS);
        }
    }
}
```


= 14SegFun.c

Build Process

- ..
  - Place code in some folder on host (use Git!)
  - Code with VS Code as usual!
  - Custom makefile **copies** files to shared folder
    (host)$ make
- ..
  - In shared folder, build gen/ledFun.out **natively** on BBG
    (bbg)$ cd /mnt/remote/pru/ledFun
    (bbg)$ make
  - Install into PRU (via /lib/firmware/am335x-pru0-fw):
    (bbg)$ make install_PRU0

Tools

- PRU Code Generation Tools (PRU CGT)
  - `clpru` compiler for C code (C89/C99)
  - Pre-installed on BBG
  - Can code in ASM for more control, but we'll do C
- Debugging (FYI)
  - Install PRU Debugger from *sourceforge.net/projects/prudebug/*
  - Run:
    (bbg)$ prudebug

Debugger Commands

r	register info
pru n	switch to PRU n
DD	Debug dump PRU memory
RESET	Reset this PRU
SS	Single step
BR	Set breakpoint
Q	Quit

GPIO with PRU

GPIO Output

- PRU's Enhanced GPIO can access some pins on P8 & P9, such as

Head_pin	$PINS	ADDR/OFFSET	GPIO NO.	Name	Mode7	Mode6	Mode5
P9_24	97	0x984/184	UART1_TXD	15	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd
P9_25	107	0x9ac/1ac	GPIO3_21	117	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7
P9_26	96	0x980/180	UART1_RXD	14	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd
P9_27	105	0x9a4/1a4	GPIO3_19	115	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5
P9_28	103	0x99c/19c	SPI1_CS0	113	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3
P9_29	101	0x994/194	SPI1_D0	111	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1
P9_30	102	0x998/198	SPI1_D1	112	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2
P9_31	100	0x990/190	SPI1_SCLK	110	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0

- PRU Pin Naming

  pr1_pru**<N>**_pru_r3**<D>**_**<B>**

  - N: 0 or 1, for PRU0 or PRU1
  - D: 0 for output, 1 for input (Direction)
  - B: 0-31 for Bit number

  – Ex: **pr1_pru0_pru_r31_3** =
  PRU____, Direction ________, Pin #___; maps to P9_28

GPIO Output Example - Flash 14 Seg

- Drive LED with P8_12

Head_pin	$PINS	ADDR/OFFSET	GPIO NO.	Name	Mode7	Mode6
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14

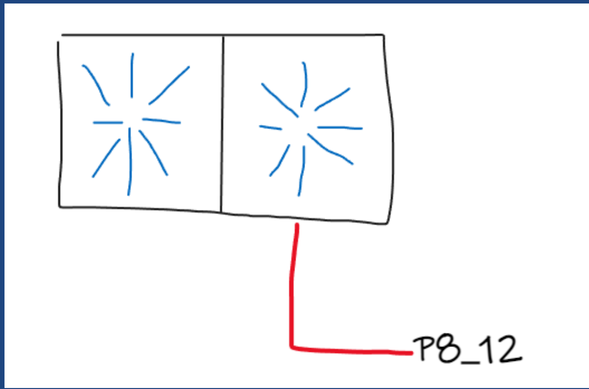
  - P8_12 = pr1_pru0_pru_r30_14 (r30=output)

- Configure Pin

```
(bbg)$ config-pin -l P8_12
Available modes for P8_12 are:
      default gpio gpio_pu gpio_pd eqep pruout

(bbg)$ config-pin P8_12 pruout
Current mode for P8_12 is:     pruout

(bbg)$ config-pin -q P8_12
Current mode for P8_12 is:     pruout
```



Getting 14SegFun.c Working

Setup (all commands on Target)

1. Set both GPIO pins to be controlled by PRU0

```
config-pin p8_12 pruout
config-pin p8_15 pruin
```

Display something on 14-seg display:

2a. Enable I2C:

```
config-pin P9_18 i2c
config-pin P9_17 i2c
```

2b. Enable I2C chip & set pattern; Pick correct board

ZEN CAPE GREEN:

```
i2cset -y 1 0x20 0x00 0x00
i2cset -y 1 0x20 0x01 0x00
i2cset -y 1 0x20 0x14 0x1E
i2cset -y 1 0x20 0x15 0x78
```

ZEN CAPE RED

```
i2cset -y 1 0x20 0x02 0x00
i2cset -y 1 0x20 0x03 0x00
i2cset -y 1 0x20 0x00 0x0f
i2cset -y 1 0x20 0x01 0x5e
```

3. On target, compile PRU code (after copying to target) & load:

```
make
make install_PRU0
```

Interactive Demo: 14SegFun changes

- Change 14SegFun code to:
  - Create a
    `void flash(int onDelayMs, int offDelayMs);`
    (note: must have a constant for `__delay_cycles`)
  - Make it flash slow once, then fast twice
  - Make it flash faster and faster, repeat
- Build Reminder
  - On host, copy files to shared folder
    ```
    (host)$ make
    ```
  - On target, build gen/14SegFun.out (based on folder name)
    ```
    (bbg)$  make
    ```
  - On target, install to /lib/firmware/am335x-pru0-fw
    ```
    (bbg)$  make install_PRU0
    ```

How To Start/Stop PRU Code

- Control PRU Via filesystem:

```
(bbg)$ cd /sys/class/remoteproc/remoteproc1/
(bbg)$ cat ./state
(bbg)$ echo 'stop' | sudo tee ./state
(bbg)$ echo 'start' | sudo tee ./state
(bbg)$ cat state
```

- Note: Starting / Stopping twice generates error
- View gen/ledFlashC.out

```
(bbg)$ readelf -h gen/ledFlashC.out
```

GPIO Output Example to Custom LED

- Drive LED with P9_27

P9_27	105	0x9a4/1a4	GPIO3_19	115	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5

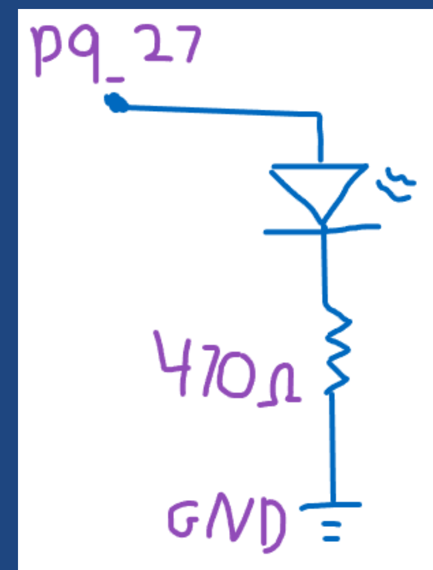
  - P9_27 = pr1_pru0_pru_r30_5 (0=output)

- Configure Pin

```
(bbg)$ config-pin -l P9_27
Available modes for P9_27 are:
            default gpio gpio_pu gpio_pd eqep pruout pruin

(bbg)$ config-pin P9_27 pruout
Current mode for P9_27 is:     pruout

(bbg)$ config-pin -q P9_27
Current mode for P9_27 is:     pruout
```



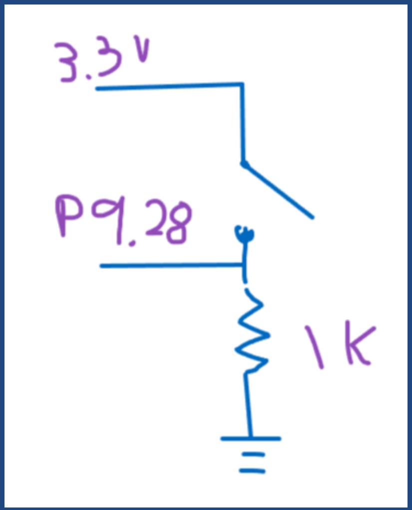
GPIO input

- Configure pin with pruin

```
(bbg)$ config-pin -l P9_28
Available modes for P9_28 are:
default gpio gpio_pu gpio_pd spi_cs pwm pwm2 pruout pruin

(bbg)$ config-pin P9_28 pruin
Current mode for P9_28 is:     pruin

(bbg)$ config-pin -q P9_28
Current mode for P9_28 is:     pruin
```



- Interactive Demo: Copy 14SegFun to buttonFun
  - Make LED mirror the button state
  - N'th time button is pressed, flash LED N times
  - Press and hold to reset N to 0

Review Questions

- What does PRU-ICSS stand for?
- Why would we use the PRU-ICSS?
- Explain the following pin: pr1_pru0_pru_r31_1
- What is the build process for developing PRU code?
- In our examples, how does the PRU manage time?

Summary

- Use PRU to meet hard real-time deadlines for RT tasks
- Process
  - Develop on host
  - Compile & install on Target
- GPIO via PRU pins
  - pr1_pru**0**_r3**0**_**5**: PRU 0, Output (0), Pin 5
  - Configure pins via Linux:

```
(bbg)$ config-pin p9_27 pruout
(bbg)$ config-pin p9_28 pruin
```

  - See P8/P9 pin description PDF.