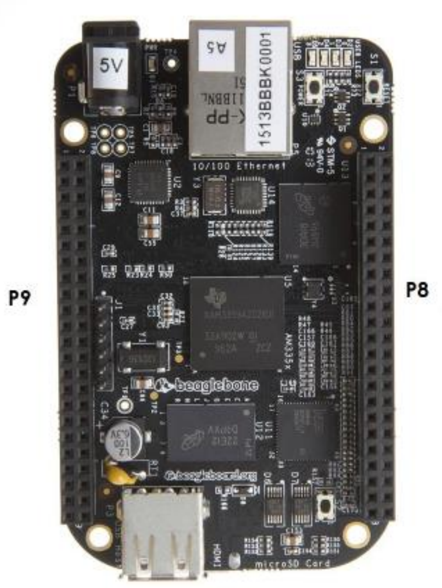
EMBEDDED DEVICE DRIVERS

Linux Device Drivers on Beaglebone Black

BBB: Expansion headers

- BBB has 2 expansion headers
  - 46-pins each
  - **3.3V compatible only**
- We will use some pins
  - In GPIO mode
  - To control some hardware



NOTE: DO NOT APPLY VOLTAGE TO ANY I/O PIN WHEN POWER IS NOT SUPPLIED TO THE BOARD. IT WILL DAMAGE THE PROCESSOR AND VOID THE WARRANTY.
NO PINS ARE TO BE DRIVEN UNTIL AFTER THE SYS_RESET LINE GOES HIGH.

BBB – Expansion pin map

BeagleBone Black

P9

GPIO	Name	Pin	Pin	Name	GPIO
	GND	P9_01	P9_02	GND	
	DC_3.3V	P9_03	P9_04	DC_3.3V	
	VDD_5V	P9_05	P9_06	VDD_5V	
	SYS_5V	P9_07	P9_08	SYS_5V	
	PWR_BUT	P9_09	P9_10	SYS_RESETn	
30	UART4_RXD	P9_11	P9_12	GPIO1_28	60
31	UART4_TXD	P9_13	P9_14	EHRPWM1A	50
48	GPIO1_16	P9_15	P9_16	EHRPWM1B	51
5	I2C1_SCL	P9_17	P9_18	I2C1_SDA	4
13	I2C2_SCL	P9_19	P9_20	I2C2_SDA	12
3	UART2_TXD	P9_21	P9_22	UART2_RXD	2
49	GPIO1_17	P9_23	P9_24	UART1_TXD	15
117	GPIO3_21	P9_25	P9_26	UART1_RXD	14
115	GPIO3_19	P9_27	P9_28	SPI1_CS0	113
111	SPI1_D0	P9_29	P9_30	SPI1_D1	112
110	SPI1_SCLK	P9_31	P9_32	VADC	
	AIN4	P9_33	P9_34	AGND	
	AIN6	P9_35	P9_36	AIN5	
	AIN2	P9_37	P9_38	AIN3	
	AIN0	P9_39	P9_40	AIN1	
20	CLKOUT2	P9_41	P9_42	GPIO0_7/3_18	7/114
	GND	P9_43	P9_44	GND	
	GND	P9_45	P9_46	GND	

P8

GPIO	Name	Pin	Pin	Name	GPIO
	GND	P8_01	P8_02	GND	
38	GPIO1_6	P8_03	P8_04	GPIO1_7	39
34	GPIO1_2	P8_05	P8_06	GPIO1_3	35
66	TIMER4	P8_07	P8_08	TIMER7	67
69	TIMER5	P8_09	P8_10	TIMER6	68
45	GPIO1_13	P8_11	P8_12	GPIO1_12	44
23	EHRPWM2B	P8_13	P8_14	GPIO0_26	26
47	GPIO1_15	P8_15	P8_16	GPIO1_14	46
27	GPIO0_27	P8_17	P8_18	GPIO2_1	65
22	EHRPWM2A	P8_19	P8_20	GPIO1_31	63
62	GPIO1_30	P8_21	P8_22	GPIO1_5	37
36	GPIO1_4	P8_23	P8_24	GPIO1_1	33
32	GPIO1_0	P8_25	P8_26	GPIO1_29	61
86	GPIO2_22	P8_27	P8_28	GPIO2_24	88
87	GPIO2_23	P8_29	P8_30	GPIO2_25	89
10	UART5_CTSN	P8_31	P8_32	UART5_RTSN	11
9	UART4_RTSN	P8_33	P8_34	UART3_RTSN	81
8	UART4_CTSN	P8_35	P8_36	UART3_CTSN	80
78	UART5_TXD	P8_37	P8_38	UART5_RXD	79
76	GPIO2_12	P8_39	P8_40	GPIO2_13	77
74	GPIO2_10	P8_41	P8_42	GPIO2_11	75
72	GPIO2_8	P8_43	P8_44	GPIO2_9	73
70	GPIO2_6	P8_45	P8_46	GPIO2_7	71

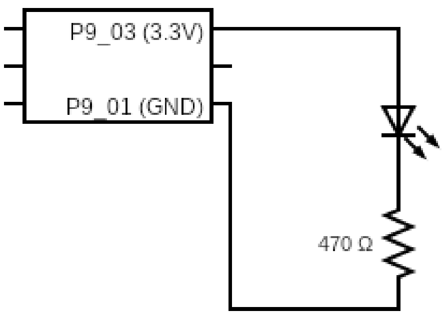
USB

BBB – Kernel GPIO map

- Kernel GPIO number – HW GPIO number – Pin number
- Examples:
  - #1:
    - Kernel GPIO # 60
      - Corresponds to
    - HW GPIO1_28
      - Which is
    - Pin P9_12
  - #2:
    - Kernel GPIO # 32
      - Corresponds to
    - HW GPIO1_0
      - Which is
    - Pin P8_25

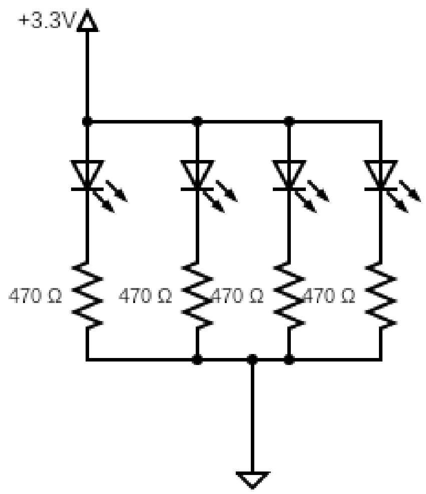
BBB – Hardware check

- Elementary power and component check
  - Disconnect the BBB power
  - Connect an LED
  - With current-limiting resistor 470 Ω
  - In series
    - 3.3V (P9_03) and GND (P9_01)
  - LED should light up once the BBB is powered on



BBB – 4-LED ladder setup

- Set up a ladder circuit
  - 4 LEDs
  - In series with 470 Ω resistors each
  - All connected and powered up
  - By 3.3V output from BBB



BBB – GPIO control on Linux

- ***config-pin***
  - BBB utility
    - To control GPIO pins
    - From user space

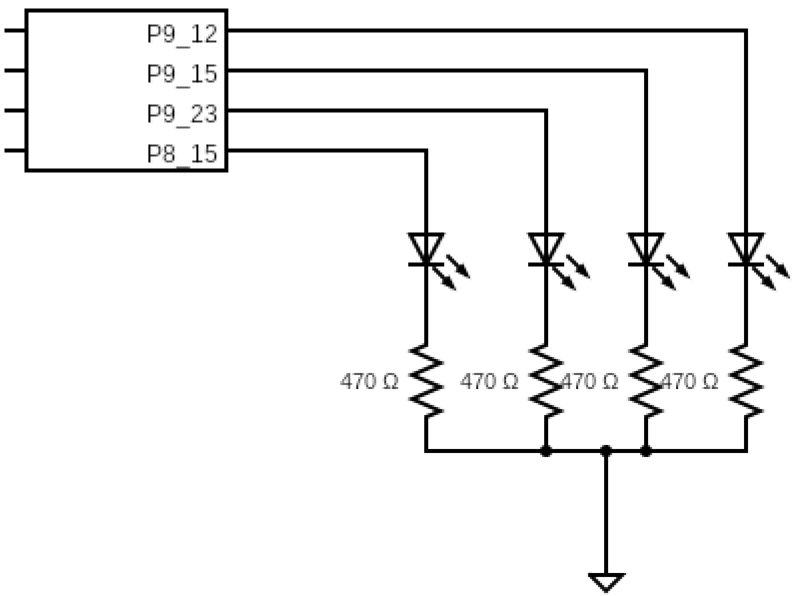
```
root@BeagleBone:~# config-pin

GPIO Pin Configurator

Usage: config-pin -c <filename>
       config-pin -l <pin>
       config-pin -q <pin>
       config-pin <pin> <mode>
```

BBB – GPIO control circuit

- Connect the circuit as per the following schematic
  - Pins refer to Beaglebone Black Pin-out



BBB – 4 LED setup (userspace)

- Export GPIOs to user space (run twice)
  (P9_12) # echo 60 > /sys/class/gpio/export
  (P9_15) # echo 48 > /sys/class/gpio/export
  (P9_23) # echo 49 > /sys/class/gpio/export
  (P8_15) # echo 47 > /sys/class/gpio/export

- Set the direction as output for these
  # config-pin P9_12 out
  # config-pin P9_15 out
  # config-pin P9_23 out
  # config-pin P8_15 out

- Set the “value” variable in all these to 1 – LED should glow
  # echo 1 > /sys/class/gpio60/value
  # echo 1 > /sys/class/gpio48/value
  # echo 1 > /sys/class/gpio49/value
  # echo 1 > /sys/class/gpio47/value

BBB – LED shell script control

- **ctrl.sh** – shell script for controlling lights

```
#!/bin/bash

set -x

gpios=(47 48 49 60)
state=${1:-0}

echo "Usage: $0 0/1 for off/on"

for i in "${gpios[@]}"
do
        echo $state > /sys/class/gpio/gpio$i/value
done
```

 # ./ctrl.sh 1 # for on
 # ./ctrl.sh 0 # for off

BBB – Decimal to binary

- Refer the script **to_bin.sh** – converts to binary

```
#!/bin/bash

set -x

echo "Usage: $0 <decimal_number < 16>"

num=${1:-15}

# MSB -> LSB
# 47 49 48 60
state60=$(( (num & 0x01) > 0 ))
state48=$(( (num & 0x02) > 0 ))
state49=$(( (num & 0x04) > 0 ))
state47=$(( (num & 0x08) > 0 ))

echo $state47 > /sys/class/gpio/gpio47/value
echo $state49 > /sys/class/gpio/gpio49/value
echo $state48 > /sys/class/gpio/gpio48/value
echo $state60 > /sys/class/gpio/gpio60/value
```

- Try running it in a loop covering all ints from 0-15!

LKM: Linux GPIO

- GPIO
  - General Purpose Input Output
    - Flexible software-controlled digital signal
    - Bidirectional
      - Input or Output
    - Values are usually binary (0/1)
    - Inputs can frequently be used as IRQ signals
      - Edge-triggered / level-triggered

- Kernel GPIO support

  #include <linux/gpio.h>

LKM: Linux GPIO APIs

- Set up

  *inline int **gpio_request**(unsigned gpio, const char *label);*
  *inline void **gpio_free**(unsigned gpio);*
  *inline int **gpio_export**(unsigned gpio, bool dir_may_change);*
  *inline void **gpio_unexport**(unsigned gpio);*

- Direction

  *inline int **gpio_direction_input**(unsigned gpio);*
  *inline int **gpio_direction_output**(unsigned gpio, int value);*

- Value changes

  *inline int **gpio_get_value**(unsigned gpio);*
  *inline void **gpio_set_value**(unsigned gpio, int value);*

- Behaviour

  *inline bool **is_gpio_valid**(unsigned gpio);*
  *inline int **get_set_debounce**(unsigned gpio, unsigned debounce);*
  *inline int **gpio_to_irq**(unsigned gpio);*

LKM: GPIO LED exercise

- Refer ***mod11*** directory
  - File ***mod11-1.c*** contains code for the module
    - We register a GPIO (GPIO60)
      - Set it for output mode
    - We create a sysfs group (***/sys/cdac_dev***) for control
      - blinkPeriod (set to default 1000 msecs)
      - mode
    - We enable module params for ***blinkPeriod***
    - We create a separate task for blinking the LED

  - Compile and load the module on a fresh-booted BBB
    - Observe ***/sys/cdac_dev*** entries
      - Note that you can change them from command line
      - And the driver will change its behavior in runtime

LKM: Interrupts: What?

- Interrupt
  - Signal sent to CPU core
  - From attached hardware device / software application
  - To indicate an event occurred requiring attention

- CPU stops what it was doing
  - Switches to a special program
    - Called Interrupt Service Routine (ISR) / Handler
  - Switches back to regular mode after ISR runs

LKM: Interrupts in the kernel

- Interrupt handler API

  *irq_handler_t **irq_handler**(unsigned int irq, void *dev_id, struct pt_regs *regs)*

- Register this handler with the kernel

  *int **request_irq**(unsigned int irq_num, irq_handler_t handler, unsigned long flags, const char *name, void *dev);*

- Free the IRQ registration

  *const void ***free_irq**(unsigned int irq, void *dev_id);*

- Header file:

  #include <linux/interrupt.h>

LKM: Interrupt handling

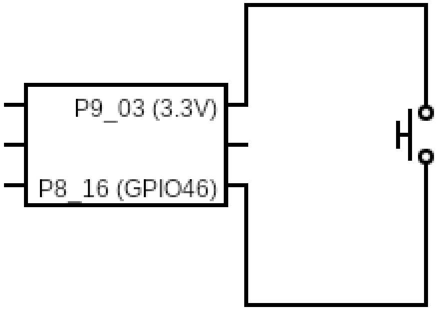
- Write an IRQ handler
  - To service the interrupt
    - Could include talking to hardware, (re)setting registers, etc.

- In the module init
  - Associate the handler with an IRQ number
    - We will use ***gpio_to_irq()***
    - If GPIO, set direction for GPIO to **"input"**
  - Register the IRQ handler
    - ***request_irq()***

- In the module exit
  - Free the IRQ registration
    - ***free_irq()***
  - Clean up and exit

LKM: Button press circuit

- We detect a button press on a BBB GPIO pin
  - Connect the push-button as per this schematic



- We set up GPIO 46 (Pin P8_16) as input
  - When button is pressed, the pin goes ***high***
  - We detect this press as an interrupt

LKM: Interrupt exercise

- Refer ***mod11*** directory
  - File ***mod11-2.c*** contains the code
    - We set up ***GPIO 46 (P8_16)*** as ***input***
    - We seek the IRQ number for the same using ***gpio_to_irq()***
    - We write an IRQ handler
      - Which prints a message and increments ***numPresses***
    - We request an IRQ for this IRQ number with this handler

  - Compile and load the module on fresh-booted BBB
    - Press the button
    - Observe ***dmesg*** output
  - Unload the module
    - Read off number of times the button was pressed from ***dmesg***

LKM: /proc/interrupts

- Once an IRQ handler is registered
  - It becomes visible in /proc/interrupts

- /proc/interrupts
  - Column-wise details:
    - Linux global IRQ number
    - No of IRQs occurred on CPU 0
    - IRQ chip receiving the IRQ
    - HW IRQ number
    - IRQ trigger type
    - Installed IRQ handler (if any)

LKM: BBB /proc/interrupts

```
root@BeagleBone:/home/debian# cat /proc/interrupts
           CPU0
 16:     892199      INTC  68 Level     clockevent
 17:          0      INTC   3 Level     arm-pmu
 19:          0      INTC  96 Level     44e07000.gpio
 20:        425      INTC  72 Level     44e09000.serial
 21:        279      INTC  70 Level     44e0b000.i2c
 22:          0      INTC  16 Level     TI-am335x-adc.0.auto
 23:          1      INTC  78 Level     wkup_m3_txev
 25:          0      INTC  75 Level     rtc0
 26:          0      INTC  76 Level     rtc0
 29:          0      INTC  71 Level     4802a000.i2c
 30:          0      INTC  65 Level     48030000.spi
 31:          0      INTC  80 Level     48038000.mcasp_tx
 32:          0      INTC  81 Level     48038000.mcasp_rx
 38:         19      INTC  98 Level     4804c000.gpio
 39:      10088      INTC  64 Level     mmc0
 40:          0      INTC  77 Level     mbox-wkup-m3
 41:        110      INTC  30 Level     4819c000.i2c
 42:          0      INTC 125 Level     481a0000.spi
 46:          0      INTC  32 Level     481ac000.gpio
 47:          0      INTC  62 Level     481ae000.gpio
 50:       1179      INTC  28 Level     mmc1
 54:          0      INTC  36 Level     tilcdc
 55:          0      INTC 111 Level     48310000.rng
 57:          0      INTC  41 Level     4a100000.ethernet
 58:          0      INTC  42 Level     4a100000.ethernet
 59:       2830      INTC  43 Level     4a100000.ethernet
 60:       3321      INTC  12 Level     49000000.dma_ccint
 62:         20      INTC  14 Level     49000000.dma_ccerrint
 66:       4139      INTC  18 Level     musb-hdrc.0
 67:          1      INTC  19 Level     musb-hdrc.1
 68:          0      INTC  17 Level     47400000.dma-controller
 69:          0      INTC 109 Level     53100000.sham
 71:          0      INTC   7 Level     tps65217-irq
 73:          0  tps65217   0 Edge      vbus
 74:          0  tps65217   2 Edge      tps65217_pwr_but
 89:         26  4804c000.gpio  14 Edge      my_button_handler
145:          0  4804c000.gpio  25 Level     tda998x
146:          0  44e07000.gpio   6 Edge      48060000.mmc cd
Err:          0
```

THANK YOU!