11. GPIO

1.1 GPIO (   )

CPU

①

②

③ SWI

④

⑤

①

②

③ ADC

④ UART

⑤

CPU CPU CPU

1.2.1 GPIO

GPIO GPIO GPIO 0 1 1 1 0 CPU

GPIO CPU CPU

GPIO 0x18 IRQ

0 IMX6ULL vector base vector base 0x80000000 DDR

GPIO LED ID

CP15

Image not found or type unknown

GIC Generic Interrupt Controller GIC FIQ IRQ CPU GPIO UART

1.2 GIC

1.2.1 IMX6ULL GIC

IMX6ULL Cortex-A7 GIC V2 Generic Interrupt Controller GIC arm

GIC CPU

Image not found or type unknown

ARM GIC VIRQ IRQ VFIQ FIQ IRQ FIQ VIRQ VFIQ IRQ FIQ GPIO

GIC

Image not found or type unknown

SPI Shared Peripheral Interrupt PPI Private Peripheral Interrupt SGI request Software-gener
Interrupt SPI

GIC Distributor CPU CPU interface

Distributor CPU

CPU CPU interface CPU CPU CPU

CPU CPU interface CPU interface sig

1.2.2 IMX6ULL GIC

GIC Distributor register CPU interface register

1.2.2.1 GICC_IAR

Image not found or type unknown

Image not found or type unknown

GICC_IAR CPU interface register ID GICC_IAR ID

1.2.2.2 GICC_EOIR

Image not found or type unknown

Image not found or type unknown

GICC_EOIR CPU interface register GICC_EOIR ID IRQ

1.2.3 CP15

1.2.3.1 CP15

ARM CP15 ARM MCR MRC cache MMU bootlo

SCTLR System Control Register VBAR Vector Base Address

1.2.3.2 SCTLR System Control Register

SCTLR cache MMU

Image not found or type unknown

Image not found or type unknown

Image not found or type unknown

Image not found or type unknown

Image not found or type unknown

Bit[13]: 0 0x00000000 vector base

Bit[12] Bit[2]: cache cache CPU cache cache cache

Bit[11]:

Bit[1]: CPU CPU

Bit[0]: MMU MMU MMU

MRC p15, 0, < Rt >, c1, c0, 0: SCTLR ARM Rt

MRC p15, 0, < Rt >, c1, c0, 0: ARM Rt SCTLR

1.2.3.3 VBAR Vector Base Address

VBAR CPU CPU

Image not found or type unknown

Image not found or type unknown

MRC p15, 0, < Rt >, c12, c0, 0: VBAR ARM Rt

MRC p15, 0, < Rt >, c12, c0, 0:   ARM   Rt   VBAR

1.3 IMX6ULL GPIO

1.3.1 GPIO interrupt configuration register1 (GPIOx_ICR1)

GPIO   1

Image not found or type unknown

ICRn[1:0]

00

01

10

11

ICR0~ICR15   GPIO interrupt 0-15

1.3.2 GPIO interrupt configuration register2 (GPIOx_ICR2)

GPIO   2

Image not found or type unknown

GPIOx_ICR1

ICR0~ICR15   GPIO interrupt 16-31

1.3.3 GPIO interrupt mask register (GPIOx_IMR)

GPIO

Image not found or type unknown

Bit[n]   interrupt n

0 interrupt n

1 interrupt n

1.3.4 GPIO interrupt status register (GPIOx_ISR)

GPIO

Image not found or type unknown

- GPIO ICR n GPIO_IMR 1

1.3.5 GPIO edge select register (GPIOx_EDGE_SEL)

GPIO

Image not found or type unknown

GPIO_EDGE_SEL [n]   GPIO   ICR [n]

1.4

1.4.1

Image not found or type unknown

KEY1 GPIO5_1 SNVS_TAMPER1 pad ALT5 KEY4 GPIO4_14 NAND_CE1_B pad ALT5 IOMUXC_SetPinMux GPIO GPIO chapter3 CORTEX A7inte interrupt ID 74 + 32 = 106 KEY2 GIC interrupt ID 72 + 32 = 104

Image not found or type unknown

1.4.2 GIC

Table 2-1. System memory map

Image not found or type unknown

gic 0xA0000

gic CP15

mrc p15, 4, r0, c15, c0, 0

gic mrc r0

1.4.3 GIC

CP15 GIC GICD_TYPER GICD_ ICENABLERn 0xFFFFFFFF SGI PPI SPI group0 distributor CPU interface

** Git NoosProgramProject/(11_GPIO /008_exception/gic.c)**

```
void gic_init(void)
{
	u32 i, irq_num;

	GIC_Type *gic = get_gic_base();

	/* the maximum number of interrupt IDs that the GIC supports */
	irq_num = (gic->D_TYPER & 0x1F) + 1;

	/* On POR, all SPI is in group 0, level-sensitive and using 1-N model */

	/* Disable all PPI, SGI and SPI */
	for (i = 0; i < irq_num; i++)
	  gic->D_ICENABLER[i] = 0xFFFFFFFFUL;

	/* The priority mask level for the CPU interface. If the priority of an
	 * interrupt is higher than the value indicated by this field,
	 * the interface signals the interrupt to the processor.
	 */
	gic->C_PMR = (0xFFUL << (8 - 5)) & 0xFFUL;

	/* No subpriority, all priority level allows preemption */
	gic->C_BPR = 7 - 5;

	/* Enables the forwarding of pending interrupts from the Distributor to the CPU interfaces.
	 * Enable group0 distribution
	 */
	gic->D_CTLR = 1UL;

	/* Enables the signaling of interrupts by the CPU interface to the connected processor
	 * Enable group0 signaling
	 */
	gic->C_CTLR = 1UL;
}
```

1.4.4

0x18 pc IRQ_Handler IRQ_Handler lr_irq handler irq C cpsie i

```
ldr r0, =_vector_table
□mcr p15, 0, r0, c12, c0, 0  /* set VBAR, Vector Base Address Register*/
```

**   Git   NoosProgramProject/(11_GPIO   /008_exception/**008_exception\start.S

```
.text
.global  _start, _vector_table
_start:
_vector_table:
□ldr  □pc, =Reset_Handler□□□ /* Reset□□□□   */
□ldr  □pc, =Undefined_Handler□□ /* Undefined instructions */
□ldr  □pc, =SVC_Handler□□□ /* Supervisor Call □   */
□b halt//ldr  □pc, =PrefAbort_Handler□□ /* Prefetch abort□□   */
□b halt//ldr  □pc, =DataAbort_Handler□□ /* Data abort□□□   */
□.word□0□□□□□□□ /* RESERVED□□□   */
□ldr  □pc, =IRQ_Handler□□□ /* IRQ interrupt□□   */
□b halt//ldr  □pc, =FIQ_Handler□□□ /* FIQ interrupt□□   */
………
.align 2
IRQ_Handler:
□/*       :
□ * 1.  lr_irq
 * 2.  SPSR_irq        CPSR
□ * 3.  CPSR  M4-M0    10010,    irq
□ * 4.    0x18
□ */

□/*      */
□/*  irq            r0-r12,       */
□/* lr-4           ,       */
□sub lr, lr, #4
□stmdb sp!, {r0-r12, lr}

□/*   irq   */
□bl handle_irq_c

□/*      */
□ldmia sp!, {r0-r12, pc}^  /* ^  spsr_irq     cpsr  */□
.align 2
Reset_Handler:
```

```
□/* Reset SCTlr Settings */
□mrc □p15, 0, r0, c1, c0, 0□  /* read SCTRL, Read CP15 System Control register□□*/
□bic □r0,  r0, #(0x1 << 13)□  /* Clear V bit 13 to use normal exception vectors  □*/
□bic □r0,  r0, #(0x1 << 12)□  /* Clear I bit 12 to disable I Cache□□□□□*/
□bic □r0,  r0, #(0x1 <<  2)□  /* Clear C bit  2 to disable D Cache□□□□□*/
□bic □r0,  r0, #(0x1 << 2)□  /* Clear A bit  1 to disable strict alignment □□*/
□bic □r0,  r0, #(0x1 << 11)□  /* Clear Z bit 11 to disable branch prediction□□*/
□bic □r0,  r0, #0x1□□□  /* Clear M bit  0 to disable MMU□□□□□□*/
□mcr □p15, 0, r0, c1, c0, 0□  /* write SCTRL, Write to CP15 System Control register□*/

    cps     #0x1B                /* Enter undef mode                */
    ldr     sp, =0x80300000     /* Set up undef mode stack      */

    cps     #0x12                /* Enter irq mode                */
    ldr     sp, =0x80400000     /* Set up irq mode stack      */

    cps     #0x13                /* Enter Supervisor mode         */
    ldr     sp, =0x80200000     /* Set up Supervisor Mode stack  */

□ldr r0, =_vector_table
□mcr p15, 0, r0, c12, c0, 0  /* set VBAR, Vector Base Address Register*/
□//mrc p15, 0, r0, c12, c0, 0  //read VBAR

□bl clean_bss

□bl system_init
□cpsie□i□□□□□ /* Unmask interrupts□□□  */

□bl main

halt:
□b  halt



clean_bss:
□/*   BSS  */
□ldr r1, =__bss_start
□ldr r2, =__bss_end
□mov r3, #0
```

```
clean:
	cmp r1, r2
	strlt r3, [r1]
	add r1, r1, #4
	blt clean

	mov pc, lr
```

1.4.5 C

gic GICC_IAR irq_handler request_irq GICC_EOIR

Git NoosProgramProject/(11_GPIO /008_exception\gic.c)

```
void handle_irq_c(void)
{
	int nr;

	GIC_Type *gic = get_gic_base();
	/* The processor reads GICC_IAR to obtain the interrupt ID of the
	 * signaled interrupt. This read acts as an acknowledge for the interrupt
	 */
	nr = gic-> C_IAR;
	printf("irq %d is happened\r\n", nr);
irq_table[nr].irq_handler(nr, irq_table[nr].param);

	/* write GICC_EOIR inform the CPU interface that it has completed
	 * the processing of the specified interrupt
	 */
	gic->C_EOIR = nr;
}
```

1.4.6 GPIO

KEY1 GPIO5_01 EDGE_SEL IMR 1 ISR 1 request_irq

Git NoosProgramProject/(11_GPIO /008_exception\main.c

```
void key_gpio5_handle_irq(void)
{
	/* read GPIO5_DR to get GPIO5_IO01 status*/
	if((GPIO5->DR >> 1 ) & 0x1) {
		printf("key 1 is release\r\n");
		/* led off, set GPIO5_DR to configure GPIO5_IO03 output 1 */
		GPIO5->DR |= (1<<3); //led on
	} else {
		printf("key 1 is press\r\n");
		/* led on, set GPIO5_DR to configure GPIO5_IO03 output 0 */
		GPIO5->DR &= ~(1<<3); //led off
	}
	/* write 1 to clear GPIO5_IO03 interrput status*/
	GPIO5->ISR |= (1 << 1);
}
void key_gpio4_handle_irq(void)
{
	/* read GPIO4_DR to get GPIO4_IO014 status*/
	if((GPIO4->DR >> 14 ) & 0x1)
		printf("key 2 is release\r\n");
	else
		printf("key 2 is press\r\n");
	/* write 1 to clear GPIO4_IO014 interrput status*/
	GPIO4->ISR |= (1 << 14);
}

void key_irq_init(void)
{
	/* if set detects any edge on the corresponding input signal*/
	GPIO5->EDGE_SEL |= (1 << 1);
	/* if set 1, unmasked, Interrupt n is enabled */
	GPIO5->IMR |= (1 << 1);
	/* clear interrupt first to avoid unexpected event */
	GPIO5->ISR |= (1 << 1);

	GPIO4->EDGE_SEL |= (1 << 14);
	GPIO4->IMR |= (1 << 14);
	GPIO4->ISR |= (1 << 14);
	request_irq(GPIO5_Combined_0_15_IRQn, (irq_handler_t)key_gpio5_handle_irq, NULL);
	request_irq(GPIO4_Combined_0_15_IRQn, (irq_handler_t)key_gpio4_handle_irq, NULL);
}
```

1.4.7

gic_enable_irq GIC GICD_ISENABLERn 1 gic_ disable _irq GIC

Git NoosProgramProject/(11_GPIO /008_exception/gic.c

```
void gic_enable_irq(IRQn_Type nr)
{
	GIC_Type *gic = get_gic_base();

	/* The GICD_ISENABLERs provide a Set-enable bit for each interrupt supported by the GIC.
	 * Writing 1 to a Set-enable bit enables forwarding of the corresponding interrupt from the
	 * Distributor to the CPU interfaces. Reading a bit identifies whether the interrupt is
enabled.
	 */
	gic->D_ISENABLER[nr >> 5] = (uint32_t)(1UL << (nr & 0x1FUL));

}

void gic_disable_irq(IRQn_Type nr)
{
	GIC_Type *gic = get_gic_base();

	/* The GICD_ICENABLERs provide a Clear-enable bit for each interrupt supported by the
	 * GIC. Writing 1 to a Clear-enable bit disables forwarding of the corresponding interrupt
from
     * the Distributor to the CPU interfaces. Reading a bit identifies whether the interrupt
is enabled.
	 */
	gic->D_ICENABLER[nr >> 5] = (uint32_t)(1UL << (nr & 0x1FUL));
}
```

1.4.8

system_init_irq_table key_irq_init GPIO gic_init GIC gic_ena

Git NoosProgramProject/(11_GPIO /008_exception/main.c

```
void system_init()
{
    init_pins();
    led_gpio_init();
    led_ctl(0);//turn off led
    boot_clk_gate_init();
    boot_clk_init();
    uart1_init();
    puts("hello world\r\n");
    system_init_irq_table();
    key_irq_init();
    gic_init();
    gic_enable_irq(GPIO5_Combined_0_15_IRQn);
    gic_enable_irq(GPIO4_Combined_0_15_IRQn);
}
```

1.4.9 4-1.4

** Git NoosProgramProject/(11_GPIO /008_exception)**

1.4.10 4-1.4

KEY1 KEY2 KEY2

Image not found or type unknown

1.5

1.5.1

** Git NoosProgramProject/(11_GPIO /011_gpio_eint)**

1.5.1.1 start.S

1

```
.text
.global  _start, _vector_table
_start:
_vector_table:
ldr pc, =Reset_Handler
    /* Reset         */
    b halt
    /* Undefined instructions */
    b halt
    /* Supervisor Call        */
    b halt
    /* Prefetch abort  */
    b halt
    /* Data abort   */
    .word 0
    /* RESERVED       */
    ldr pc, =IRQ_Handler
    /* IRQ interrupt  */
    b halt
    /* FIQ interrupt  */
.align 2
IRQ_Handler:
b halt
.align 2
    Reset_Handler:
b halt
halt:
b halt
```

_start 05 0x00 06 0x4 11 0x18 11.1.2

0x00 ldr Reset Handler pc CPU Reset Handler CPU IRQ_Handler

2

```
Reset_Handler:
/* Reset SCTlr Settings */
```

```
    mrc  p15, 0, r0, c1, c0, 0    /* read SCTRL, Read CP15 System Control register */
    bic  r0,  r0, #(0x1 << 13)    /* Clear V bit 13 to use normal exception vectors  */
    bic  r0,  r0, #(0x1 << 12)    /* Clear I bit 12 to disable I Cache    */
    bic  r0,  r0, #(0x1 <<  2)    /* Clear C bit  2 to disable D Cache    */
    bic  r0,  r0, #(0x1 << 1)       /* Clear A bit  1 to disable strict alignment  */
    bic  r0,  r0, #(0x1 << 11)    /* Clear Z bit 11 to disable branch prediction  */
    bic  r0,  r0, #0x1        /* Clear M bit  0 to disable MMU     */
    mcr  p15, 0, r0, c1, c0, 0    /* write SCTRL, Write to CP15 System Control register */

    cps     #0x1B    /* Enter undef mode               */
    ldr     sp, =0x80300000     /* Set up undef mode stack      */

    cps     #0x12    /* Enter irq mode                */
    ldr     sp, =0x80400000     /* Set up irq mode stack       */

    cps     #0x13    /* Enter Supervisor mode         */
    ldr     sp, =0x80200000     /* Set up Supervisor Mode stack  */

    ldr r0, =_vector_table
    mcr p15, 0, r0, c12, c0, 0   /* set VBAR, Vector Base Address Register*/
    //mrc p15, 0, r0, c12, c0, 0   //read VBAR

    bl clean_bss

    bl system_init
    cpsie i      /* Unmask interrupts     */

    bl main

 halt:
    b  halt



 clean_bss:
    /*   BSS  */
    ldr r1, =__bss_start
    ldr r2, =__bss_end
    mov r3, #0
 clean:
    cmp r1, r2
```

```
□□strlt r3, [r1]
□□add r1, r1, #4
□□blt clean

□□mov pc, lr
```

Reset_Handler Icache Dcache MMU CPS cps #0x1B undef mode und mode CPS #0x13 Supervisor mode Supervisor mode C

3 IRQ

```
IRQ_Handler: □
/*       : □
* 1. lr_irq                    □
* 2. SPSR_irq        CPSR □
* 3. CPSR  M4-M0   10010,    irq   □
* 4.   0x18        □ */  □
/*      */ □
/*  irq           r0-r12,       */ □
/* lr-4          ,      */ □
sub lr, lr, #4 □
stmdb sp!, {r0-r12, lr}   □ □

/*   irq   */ □
bl handle_irq_c □ □

/*      */ □
ldmia sp!, {r0-r12, pc}^  /* ^  spsr_irq    cpsr  */□
```

IRQ_Handler handle_irq_c r0-r12

1.5.1.2 interrupt.c

1 GIC

```
void key_exit_init(void)
□{
□□GPIO5_IMR□□□□□□□= (volatile unsigned int *)(0x20AC014);
□□GPIO5_EDGE_SEL□□□□□□= (volatile unsigned int *)(0x20AC01C);
□□GPIO5_ISR□□□□□□□= (volatile unsigned int *)(0x20AC018);
□□GPIO5_DR                                = (volatile unsigned int *)(0x20AC000);
□□GPIO4_IMR□□□□□□□= (volatile unsigned int *)(0x20A8014);
□□GPIO4_EDGE_SEL□□□□□□= (volatile unsigned int *)(0x20A801C);
□□GPIO4_ISR□□□□□□□= (volatile unsigned int *)(0x20A8018);
□□GPIO4_DR□□□□□□□= (volatile unsigned int *)(0x20A8000);

□□gic_init();
□□gic_enable_irq(GPIO5_Combined_0_15_IRQn);
□□gic_enable_irq(GPIO4_Combined_0_15_IRQn);

□□/*   GPIOx_EDGE_SEL
□□ * GPIO_EDGE_SEL bit is set, then a rising edge or falling edge in the corresponding
□□ *      signal generates an interrupt.
□□ * GPIO5_EDGE_SEL  0x20AC01C
□□ * bit[1] = 0b1
□□ * GPIO4_EDGE_SEL  0x20A801C
□□ * bit[14] = 0b1
□□ */
□□*GPIO5_EDGE_SEL |= (1<<1);
□□*GPIO4_EDGE_SEL |= (1<<14);

□□/*   GPIOx_IMR
□□ * GPIO_IMR contains masking bits for each interrupt line.
□□ * GPIO5_IMR  0x20AC014
□□ * bit[1] = 0b1
□□ * GPIO4_IMR  0x20A8014
□□ * bit[14] = 0b1
□□ */
□□*GPIO5_IMR |= (1<<1);
□□*GPIO4_IMR |= (1<<14);
□}
```

key_exit_init gic_init GIC gic_enable_irq GPIO5_00~GPIO5_15 GPIO4_00~

GPIOx_EDGE_SEL GPIOx_IMR GPIO5_1 key1 GPIO4_14 key2

2 C

```
void handle_irq_c(void)
□{
□□int nr;
```

```
    GIC_Type *gic = get_gic_base();
    /* The processor reads GICC_IAR to obtain the interrupt ID of the
     * signaled interrupt. This read acts as an acknowledge for the interrupt
     */
    nr = gic->C_IAR;
    printf("irq %d is happened\r\n", nr);

    switch(nr)
    {
        case GPIO5_Combined_0_15_IRQn:
        {
            /* read GPIO5_DR to get GPIO5_IO01 status*/
            if((*GPIO5_DR >> 1 ) & 0x1) {
                printf("key 1 is release\r\n");
                /* led off, set GPIO5_DR to configure GPIO5_IO03 output 1 */
                led_ctl(0);
            } else {
                printf("key 1 is press\r\n");
                /* led on, set GPIO5_DR to configure GPIO5_IO03 output 0 */
                led_ctl(1);
            }
            /* write 1 to clear GPIO5_IO03 interrput status*/
            *GPIO5_ISR |= (1 << 1);
            break;
        }

        case GPIO4_Combined_0_15_IRQn:
        {
            /* read GPIO4_DR to get GPIO4_IO014 status*/
            if((*GPIO4_DR >> 14 ) & 0x1)
            {
                printf("key 2 is release\r\n");
                led_ctl(0);
            }
            else
            {
                printf("key 2 is press\r\n");
                led_ctl(1);
            }
```

```
                /* write 1 to clear GPIO4_IO014 interrput status*/
                *GPIO4_ISR |= (1 << 14);
                break;
            }

            default:
                break;
        }

        /* write GICC_EOIR inform the CPU interface that it has completed
         * the processing of the specified interrupt
         */
        gic->C_EOIR = nr;
    }
```

handle_irq_c GICC_IAR ID ID ID GPIOx_ISR

1.5.2

1.5.2.1 SDK

```
#include "gic.h"
#include "my_printf.h"

GIC_Type * get_gic_base(void)
{
    GIC_Type *dst;

    __asm volatile ("mrc p15, 4, %0, c15, c0, 0" : "=r" (dst));

    return dst;
}

void gic_init(void)
{
    u32 i, irq_num;

    GIC_Type *gic = get_gic_base();
```

```
	/* the maximum number of interrupt IDs that the GIC supports */
	irq_num = (gic->D_TYPER & 0x1F) + 1;

	/* On POR, all SPI is in group 0, level-sensitive and using 1-N model */

	/* Disable all PPI, SGI and SPI */
	for (i = 0; i < irq_num; i++)
	  gic->D_ICENABLER[i] = 0xFFFFFFFFUL;

	/* The priority mask level for the CPU interface. If the priority of an
	 * interrupt is higher than the value indicated by this field,
	 * the interface signals the interrupt to the processor.
	 */
	gic->C_PMR = (0xFFUL << (8 - 5)) & 0xFFUL;

	/* No subpriority, all priority level allows preemption */
	gic->C_BPR = 7 - 5;

	/* Enables the forwarding of pending interrupts from the Distributor to the CPU interfaces.
	 * Enable group0 distribution
	 */
	gic->D_CTLR = 1UL;

	/* Enables the signaling of interrupts by the CPU interface to the connected processor
	 * Enable group0 signaling
	 */
	gic->C_CTLR = 1UL;
}

void gic_enable_irq(IRQn_Type nr)
{
	GIC_Type *gic = get_gic_base();

	/* The GICD_ISENABLERs provide a Set-enable bit for each interrupt supported by the GIC.
	 * Writing 1 to a Set-enable bit enables forwarding of the corresponding interrupt from the
	 * Distributor to the CPU interfaces. Reading a bit identifies whether the interrupt is
enabled.
	 */
	gic->D_ISENABLER[nr >> 5] = (uint32_t)(1UL << (nr & 0x1FUL));
```

```
}

void gic_disable_irq(IRQn_Type nr)
{
	GIC_Type *gic = get_gic_base();

	/* The GICD_ICENABLERs provide a Clear-enable bit for each interrupt supported by the
	 * GIC. Writing 1 to a Clear-enable bit disables forwarding of the corresponding interrupt from
	 * the Distributor to the CPU interfaces. Reading a bit identifies whether the interrupt is enabled.
	 */
	gic->D_ICENABLER[nr >> 5] = (uint32_t)(1UL << (nr & 0x1FUL));
}
```

gic_init   GIC

gic_enable_irq   GPIO

gic_disable_irq   GPIO

get_gic_base   GIC   GIC   GIC

1.5.2.2 Makefile

```
PREFIX=arm-linux-gnueabihf-
CC=$(PREFIX)gcc
LD=$(PREFIX)ld
AR=$(PREFIX)ar
OBJCOPY=$(PREFIX)objcopy
OBJDUMP=$(PREFIX)objdump

INCLUDEDIR	:= $(shell pwd)/include
CFLAGS		:= -Wall
CPPFLAGS	:= -nostdinc -fno-builtin -I$(INCLUDEDIR)
LDFLAGS         := -L /usr/arm/gcc-linaro-6.2.1-2016.11-x86_64_arm-linux-gnueabihf/lib/gcc/arm-linux-gnueabihf/6.2.1 -lgcc
objs :=  start.o main.o led.o key.o interrupt.o uart.o eabi_compat.o my_printf.o gic.o

TARGET := eint
```

```
$(TARGET).img : $(objs)
	$(LD) -T imx6ull.lds -o $(TARGET).elf $^ $(LDFLAGS)
	$(OBJCOPY) -O binary -S $(TARGET).elf  $(TARGET).bin
	$(OBJDUMP) -D -m arm  $(TARGET).elf  > $(TARGET).dis	
	./tools/mkimage -n ./tools/imximage.cfg.cfgtmp -T imximage -e 0x80100000 -d $(TARGET).bin
$(TARGET).imx
	dd if=/dev/zero of=1k.bin bs=1024 count=1
	cat 1k.bin $(TARGET).imx > $(TARGET).img

%.o: %.c
	${CC} $(CPPFLAGS) $(CFLAGS) -c -o $@ $<

%.o: %.S
	${CC} $(CPPFLAGS) $(CFLAGS) -c -o $@ $<

clean:
	rm -f $(TARGET).dis  $(TARGET).bin $(TARGET).elf $(TARGET).imx $(TARGET).img *.o
```

1.5.3 4-1.4

Git NoosProgramProject/(11_GPIO /011_gpio)_eint

1.5.3.1 4-1.4

key1 key2 ID Led MCIMX6Y2.h GPIO5_Combined_0_15_IRQn GPIO4_Combi

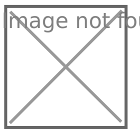
Image not found or type unknown

Revision #1
Created 3 March 2022 02:44:14 by
Updated 3 March 2022 02:44:30 by