FWA-1010VC SW Define LED & Button

Project Name: *FWA-1010VC*

Author(s): *Justin.Yang*

Last saved: *09/04/2018*

Version: *V1.00*

APPROVALS			
Name (printed)	**Title**	**Signature**	**Date**

HISTORY

Revision	Date	Person	Description
1.0	09/03/2018	Justin.Yang	1st draft ref.

1. INTRODUCTION

This doc. will describe the FWA-1010VC hardware design specification on SW Define LED & SW_BTN, and the related H/W information for GPIO programming.

1.1 Mainboard Diagram Overview

This section describes detail block diagram and functionality of mainboard. Following is mainboard diagram, as the design doc, the SW Define LED pin (F5) & SW_BTN (F6) are controlled from Intel Rangeley SOC.

Figure 3: FWA-1010VC System Front View

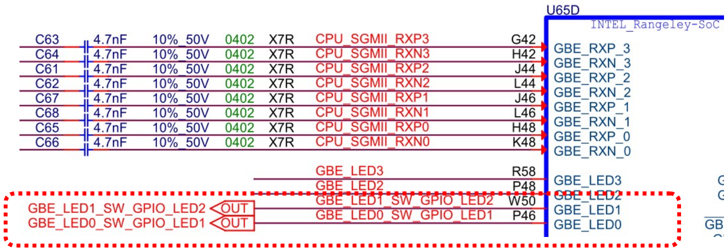
Item	Component	Description
F1	Power LED	Power status
F2	Disk Activity LED	SSD disk activity
F3	Wireless Status LED	Wireless activity
F4	Wi-Fi Status LED	Wi-Fi activity
F5	Software Defined LED	Status signaling
F6	Software Controlled Event Button	Event button for user interaction

2. FWA-1010VC SOC GPIO DESIGN

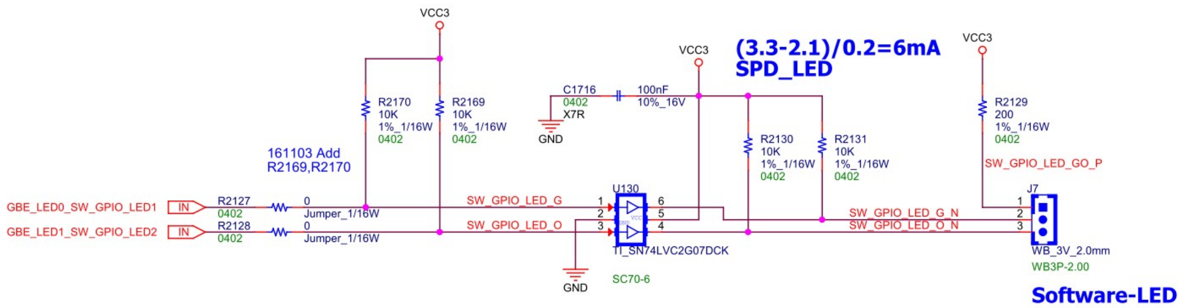
2.1 SOC GPIO Configuration

In FWA-1010VC design, the Software Define LED (F5) is controlled by GBE_LED0 & GBE_LED1 of Intel Rangeley SOC, and SW Define Button (F6) is controlled by UART1_TXD.

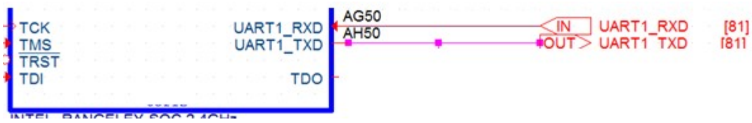
GBE_LED0 & GBE_LED1 of Intel Rangeley SOC:



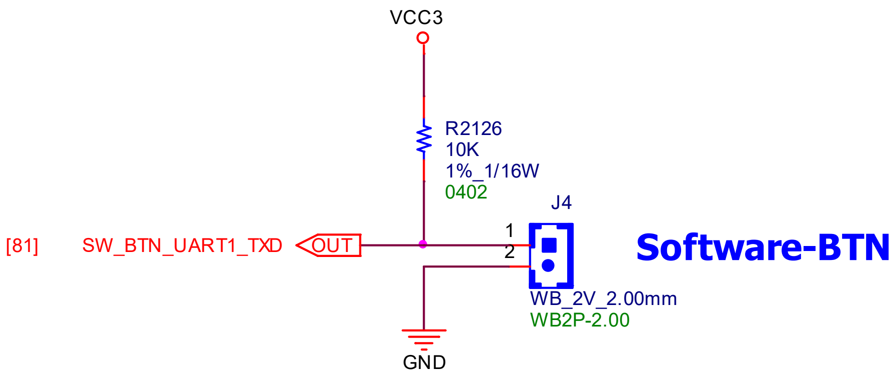
Circuit of Software Define LED connector:



UART1_TXD of Intel Rangeley SOC:



Circuit of SW Define Button:



2.2 Pin Definitions for SW Define LED & SW_BTN

Following table is Pin definition on SW Define LED

Pin Number	Name	Pin Description
1	Vcc3	3.3V voltage.
2	SW_GPIO_LED_G_N	Green LED control pin
3	SW_GPIO_LED_O_N	Orange LED control pin

Following table is Pin definition on SW_BTN

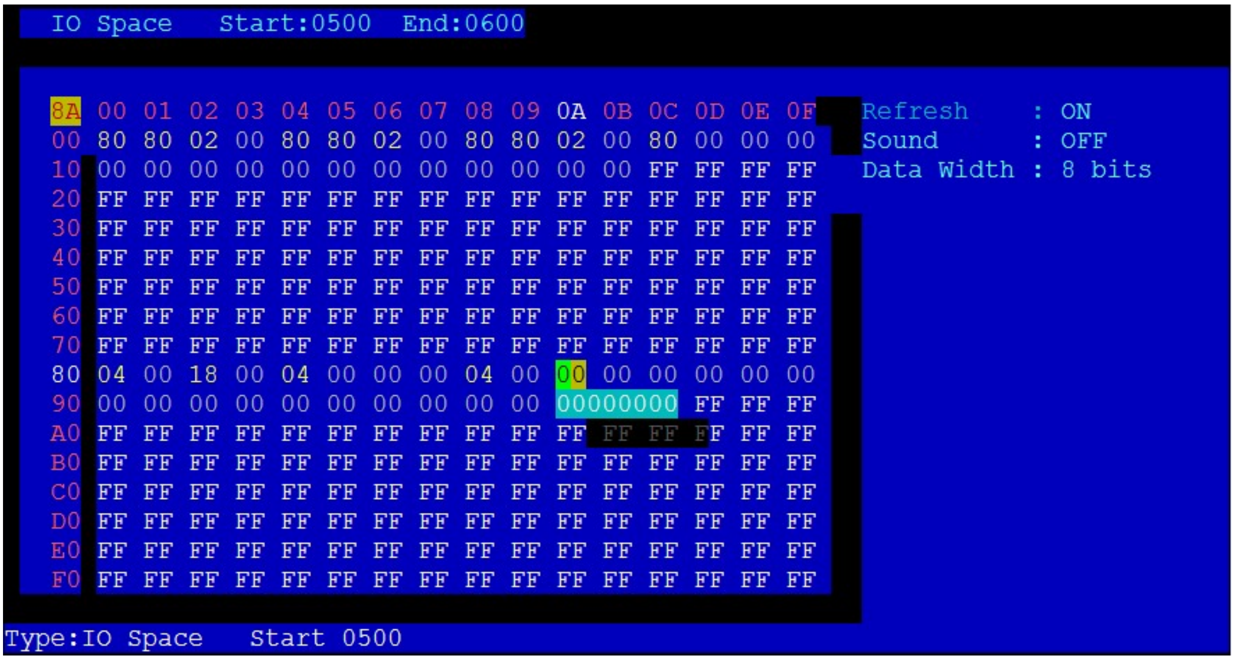
Pin Number	Name	Pin Description
1	SW_BTN_UART1_TXD	SW_BTN signal input pin
2	GND	Ground

2.3 GPIO of SW Define LED

Following tables are GBE_LED0 & GBE_LED1 GPIO registers setting.

GBE_LED0	As BIOS Starts	GBE_LED0	O	None	V3P3A
	SUS_USE_SEL = 1	GPIO_SUS19	Set by SW	Design Specific	V3P3A
GBE_LED1	As BIOS Starts	GBE_LED1	O	None	V3P3A
	SUS_USE_SEL = 1	GPIO_SUS20	Set by SW	Design Specific	V3P3A

User can check reg status from IOPORT 0x58A



```
 IO Space    Start:0500   End:0600

8A 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   Refresh     : ON
00 80 80 02 00 80 80 02 00 80 80 02 00 80 00 00 00   Sound       : OFF
10 00 00 00 00 00 00 00 00 00 00 00 00 FF FF FF FF   Data Width  : 8 bits
20 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
30 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
40 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
50 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
60 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
70 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
80 04 00 18 00 04 00 00 00 04 00 00 00 00 00 00 00
90 00 00 00 00 00 00 00 00 00 00 00000000 FF FF FF
A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
F0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

Type:IO Space     Start 0500
```

Here we use Linux ioport utility to demonstrate LED control (http://people.redhat.com/rjones/ioport/)

Note for ioport utilities

These commands enable command line and script access directly to I/O ports on PC hardware.

The *inb*, inw and inl commands perform an input (read) operation on the given I/O port, and print the result.

The outb, *outw* and outl commands perform an output (write) operation to the given I/O port, sending the given data. Note that the order of the parameters is ADDRESS DATA.

The size of the operation is selected according to the suffix, with b meaning byte, wmeaning word (16 bits) and l meaning long (32 bits).

Download and Install ioport utility

```
# wget http://people.redhat.com/rjones/ioport/files/ioport-1.2
# tar zxvf ioport-1.2.tar.gz
# cd ioport-1.2/
# ./configure
checking for a BSD-compatible install... /usr/bin/install -c
~~ skip process ~~
# make
make    all-am
make[1]: Entering directory '/root/ioport-1.2'
~~ skip process ~~
```

Set LED

```
# ./outw 0x58a 0x08    - Orange
# ./outw 0x58a 0x10    - Green
# ./outw 0x58a 0x18    - OFF
```

2.4 GPIO of SW Define Button

Following tables are URAT1_TXD GPIO register setting

Customer GPIO when SC_USE_SEL Bit = 1	SoC Ball/Pin Number	SoC Power Well	Bit in GBASE + 00h-17h Registers	Native Signal when SC_USE_SEL Bit = 0
GPIOS_0	AL56	Core	0	NMI
GPIOS_1	AL63	Core	1	ERROR2_B
GPIOS_2	AL62	Core	2	ERROR1_B
GPIOS_3	AL65	Core	3	ERROR0_B
GPIOS_4	AM52	Core	4	IERR_B
GPIOS_5	AL52	Core	5	MCERR_B
GPIOS_6	AG50	Core	6	UART1_RXD
GPIOS_7	AH50	Core	7	UART1_TXD

UART1_TXD	Strap Sampling	0 = Override SPI Flash Descriptor Security	I	20K PU	V3P3S
	As BIOS Starts	UART1_TXD	O	None	V3P3S
	SC_USE_SEL = 1	GPIOS_7	Set by SW	Design Specific	V3P3S

A software defined button (F6 in Figure 3) is provided on the FWA-1010VC. BIOS is programmed well to monitor button event trigger. The acpid service (ACPI event daemon) is used to handle the corresponding action of this button event. When user press software controlled button, the button event will be triggered and the corresponding action will be executed.

Here's an example on how SW defined button behaves under CentOS 7

1. Make sure there is no other service that handle the sleep button event. CentOS 7 default will use systemd-logind to handle the sleep button event.

A. Disable it by modifying "/etc/systemd/logind.conf". Set HandleSuspendKey=ignore

```
HandleSuspendKey=ignore
HandleHibernateKey=ignore
HandleLidSwitch=ignore
HandleLidSwitchDocked=ignore
```

B. Restart the service

```
# systemctl restart systemd-logind.service
```

2. Use acpid to handle the action of sleep button.

A. install acpid by yum install

```
# yum install acpid
```

B. Create and edit /etc/acpi/events/sleepconf

```
event=button/sleep.*
action=/etc/acpi/actions/sleep.sh
```

*Event file name can have any name you like.

*The "event" line is a regular expression specifying the events we're interested in. in this case we use sleep event.

*The "action" line is the command to be executed when these events are dispatched.

Here we call the sleep.sh residing in /etc/acpi/actions, you can write some complex actions in this script.

e.g. vim /etc/acpi/actions/sleep.sh

```
#!/bin/sh
echo "SW button test"
```

C. Start / Restart acpid (you can check how many rule loaded by checking acpid status.)

```
# systemctl start acpid.service
# systemctl status acpid.service
â—   acpid.service - ACPI Event Daemon
   Loaded: loaded (/usr/lib/systemd/system/acpid.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2018-09-04 11:49:16 UTC; 2s ago
  Process: 2515 ExecStart=/usr/sbin/acpid $OPTIONS (code=exited, status=0/SUCCESS)
 Main PID: 2516 (acpid)
   CGroup: /system.slice/acpid.service
           â””â”€2516 /usr/sbin/acpid

Sep 04 11:49:16 1010 systemd[1]: Starting ACPI Event Daemon...
Sep 04 11:49:16 1010 acpid[2516]: starting up with netlink and the input layer
Sep 04 11:49:16 1010 acpid[2516]: skipping incomplete file /etc/acpi/events...nf
Sep 04 11:49:16 1010 acpid[2516]: 2 rules loaded
Sep 04 11:49:16 1010 acpid[2516]: waiting for events: event logging is off
Sep 04 11:49:16 1010 systemd[1]: Started ACPI Event Daemon.
Hint: Some lines were ellipsized, use -l to show in full.
```

D. Execute #acpi_listen so you can see the button is pressed.

```
# acpi_listen
```

E . When button is pressed, you will see

```
button/sleep SBTN 00000080 00000000
```