Migrating from ATMega88 to LGT8F88A

This documentation is mainly details the difference between ATMega88 and LGT8F88A, from its internal RISC architecture to application design. We try to cover all the difference between this two types of microcontroller.

If you have using experience of AVR series MCU from ATMEL, here you can get everything you need to start using LGT8F88A.

We will make comparison from follow aspects:

- RISC architecture and instruction
- Clock and power management
- Peripherals
- Fuse settings
- Pin-out and package
- Development and Debug
- In system programming

At the end of this documentation, we also provide the overview of registers definition and instruction details of LGT8F88A, It's useful when you need cross- reference

LogicGreen

8-bit LGT8XM

RISC Microcontroller with 8192 Bytes In-System Programmable FLASH Memory

LGT8F88A

Application Documentation

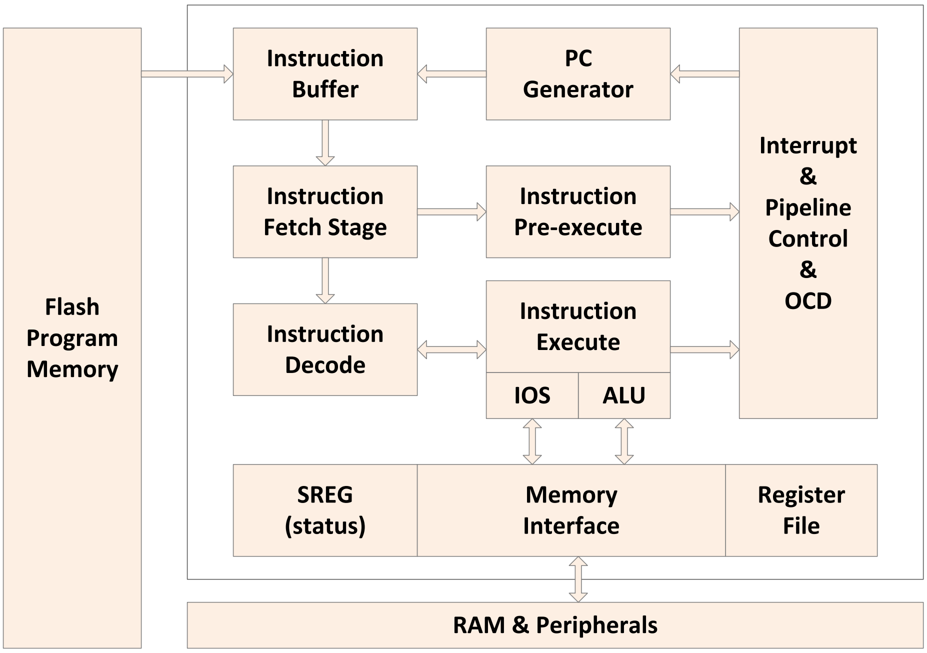
Migrating from ATMega88 to LGT8F88A

Version 1.0

2013/12/15

RISC ARCHITECTURE AND INSTRUCTION

LGT8F88A is implemented based on LGT8XM, a most optimized 8bit RISC core. LGT8XM is designed to execute most of the AVR8 instructions, but it has a very different implementation.



LGT8XM uses a Harvard architecture, like traditional AVR, with separate memories and buses for program and data. But LGT8XM implement a two-stage pipeline. Compare to most 3-stage pipeline design, 2-stage can reduce most of redundant fetch operations, so reduce flash memory accessing and save more power consumption. Compare to 1-stage pipeline used by most AVR architecture, LGT8XM uses the additional stage to implement instruction pre-decoding. LGT8XM has a very flexible SRAM interface, which can improve the performance when it used as a stack. All of those features make LGT8XM even more powerful and efficient to execute RISC instructions.

LGT8XM has the same register file as most AVR implementation. But LGT8XM's register file implements a very powerful data forwards channel, which active data to all pipeline stage immediately when updated.

The LGT8XM is designed with performance bear in mind. We optimize its executing flow cycle by cycle and finally get a very powerful AVR implementation which never happened.

Here we summary instruction cycles of several multi-cycle ones to demo our difference:

Instruction	Function	Cycle of AVR	Cycle of LGT8XM
ADIW	Add immediate to word	2	1
SBIW	Subtract immediate to word	2	1
MUL/S/SU	8bit multiply	2	1
FMUL/S/SU	Fractional multiply	2	1
RJMP/RCALL	Relative jump/call	2/3	1
IJMP/ICALL	Indirect jump/call	2/3	2
RET/IRET	Return	4	2
CPSE	Compare, skip if equal	1/2/3	1/2
SBRC/SBRS SBIC/SBIS	Skip if set or skip if cleared	1/2/3	1/2

LD/LDD	Load indirect	2	1
ST/STD	Store indirect	2	1
LPM	Load program memory	3	2
PUSH/POP	Stack access	2	1

Thanks to the 16bit ALU inside LGT8XM, we can archive one cycle fast when do 16bit data move and arithmetic operations. As you can find from above list, LGT8XM completes all its instructions inside two cycles.

Another improvement is on interrupt acknowledgement, more fast instruction execution cycle brings more fast interrupts acknowledge timing. LGT8XM acknowledge any interrupt request inside 4 system cycle. After 4 cycles, the first instruction of interrupt server routine has enter pipeline and ready for execution.

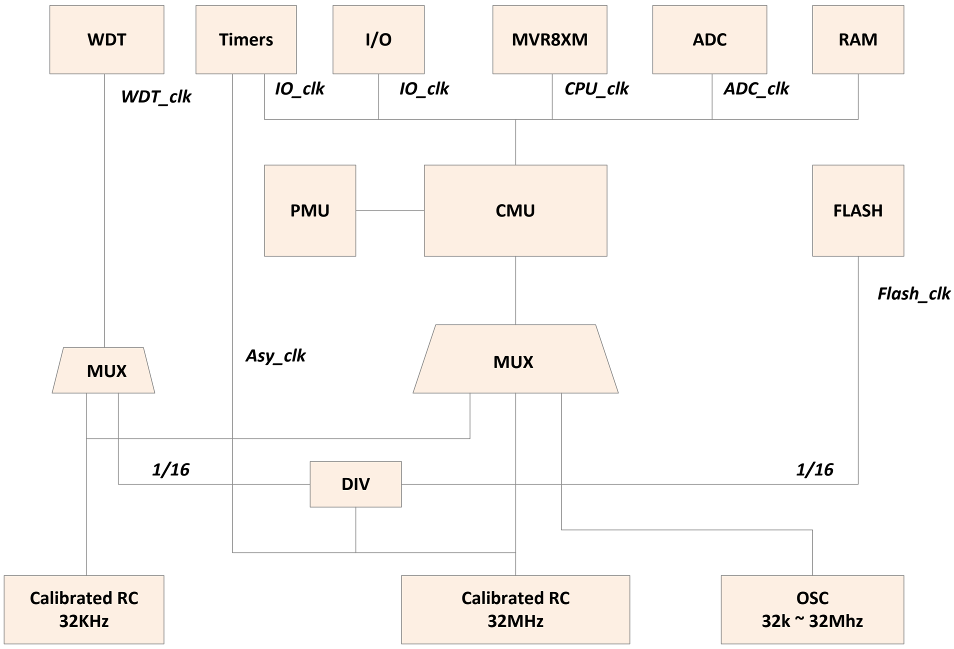
As we outline above, LGT8XM is a more powerful AVR implementation. It can execute all the AVR instructions, so it's compatible to mostly all the existing firmware wrote for AVR microcontroller. But LGT8XM is not a cycle compatible with traditional AVR from ATMEL, so you should take care when transfer from ATMEL's AVR to our LGT8XM implementation, especially for control timing which implemented based on instructions cycles.

CLOCK AND POWER MANAGEMENT

LGT8F88A uses a very different clock system compare to ATMega88. Most AVR microcontroller including a special memory called FUSE to store kinds of system configurations and informations. This FUSE feature is very flexible and practical, but also can bring mis-operations, especially when you are starting out with AVR.

Based on those considerations, we reduce most of the fuse settings, and implement them by I/O registers. Only those that must be used during system power up, for example, BOR settings. For more details about system configuration of LGT8F88A, please refer to following "**Fuse setting**" section of this documentation.

Different from ATMega88, LGT8F88A uses **PMCR** register to manager its clock system. LGT8F88A itself also has a bit difference clock subsystem compare to ATMega88. Following is clock distribution of LGT8F88A:



As you can see from above diagram, LGT8F88A including three type of main clock source:

1. 32MHz internal RC oscillator, with 1% resolution after factory calibration
2. 32KHz internal RC oscillator, which also support 1% resolution after calibration
3. 32.768 KHz or 400K~32MHz external crystal I/O, which also support directly clock input.

System clock configure is implemented using PMCR register, which defined as following:

PMCR – Power Management Control Register							
Address: 0xF2				Default Value: 0x01			
PMCR	PMCE	-	WCLKS	OSCK_EN	OSCM_EN	RC32K_EN	RC32M_EN
R/W	R/W	-	R/W	R/W	R/W	R/W	R/W
Initial	0	-	0	0	0	0	1
Bit Definitions							
[0]	RC32M_EN	Internal 32MHz RC oscillator enable control. 1: enable, 0: disable					
[1]	RC32K_EN	Internal 32KHz RC oscillator enable control， 1: enable, 0: disable					
[2]	OSCM_EN	External high frequency crystal (400K~32M) control 1: enable, 0: disable					
[3]	OSCK_EN	External low frequency crystal (32KHz) control 1: enable, 0: disable					
[4]	WCLKS	WDT clock source selector, refer to Peripherals/WDT section					
[5]	CLKSS	Main clock source selector, refer to following description					
[6]	CLKFS	Main clock source selector, refer to following description					
[7]	PMCE	PMCR update enable control bit Set this bit before any update to PMCR, then do write within next 4 cycles					

System clock configuration:

CLKFS	CLKSS	System clock source
0	0	Internal 32MHz RC oscillator (default setting)
0	1	External 400K ~ 32MHz crystal I/O
1	0	Internal 32KHz RC oscillator
1	1	External 32K ~ 400KHz crystal I/O

System clock is selected and feed to a clock pre-scalar module to divide into slower clocks for application requirements. You can configure this clock scalar by **CLKPR** register which is compatible with ATMega88.

LGT8F88A is clocked by internal 32MHz RC oscillator after system power up. This clock source is divided by 8 by system clock per-scalar, so LGT8XM core is working at 4MHz.

You can update system clock settings to meet your application by software at any time you want. But mostly, It’s recommended to finish clock configure while system initialization. It’s also recommended to disable any clock source to reduce power consumption if they are not needed. But one thing should kept in mind, when you restart a clock source, it required to waiting for a proper while for its output becoming stable.

Code segments to demo system clock configuration:

Code to demo system clock configuration

```
#define PMCR *(volatile unsigned char *)0xf2
#define rtmp *(volatile unsigned char *)0x4A  // GPIOR1

void init_clock(void)
{
   PMCR = 0x80;  // PMCE enable
   PMCR |= 0x04; // enable crystal i/o
   delay_us(1);  // waiting crystal i/o to ready

   rtmp = PMCR;
   rtmp &= 0x9f;
   rtmp |= 0x20;  // switch system clock to crystal
   PMCR = 0x80;   // PMCE enable
   PMCR = rtmp;
}
```

The Power management of LGT8F88A is also a bit different from ATMeag88. You can keep using **[SMCR]** register and ***SLEEP*** instruction to bring MCU to low power mode, but the power mode definition is lightly different. LGT8F88A has less low power modes compare to ATMega88, but user can feel free to disable any modules if not needed. For example, before set to IDLE mode, you can switch system clock to internal 32KHz RC oscillator, then disable 32MHz RC oscillator to archive very low power consumption.

Power mode definition of LGT8F88A:

<table>
<tr><th colspan="6">SMCR – Sleep Mode Control Register</th></tr>
<tr><td colspan="2">SMCR: 0x33(0x53)</td><td colspan="4">Default Value: 0x00</td></tr>
<tr><td colspan="6"></td></tr>
<tr><td>SMCR</td><td></td><td>SM2</td><td>SM1</td><td>SM0</td><td>SE</td></tr>
<tr><td>R/W</td><td>-</td><td>R/W</td><td>R/W</td><td>R/W</td><td>R/W</td></tr>
<tr><td>Initial</td><td>-</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td colspan="6">Bit definition</td></tr>
</table>

<table>
<tr><td>[0]</td><td>SE</td><td colspan="4">Sleep mode enable control. Set SE bit following by a Sleep instruction, will bring MCU to sleep mode. It’s recommended to clear this bit after waked up from sleep mode.</td></tr>
<tr><td rowspan="8">[3:1]</td><td rowspan="8">SM</td><td colspan="4">Sleep mode selection</td></tr>
<tr><td>SM2</td><td>SM1</td><td>SM0</td><td>Descriptions</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>IDLE Mode</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>ADC Noise cancel</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Power/Down</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Power/Off</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Power/Off Lock</td></tr>
<tr><td colspan="3">Others</td><td>Reserved</td></tr>
<tr><td>[7:4]</td><td>-</td><td colspan="4">Reserved</td></tr>
</table>

Power/Off is a deepest sleep mode, which can archive lowest power consumption. In Power/Off mode, most of logic inside microcontroller are powered off, means no power supply. Only POR and part of waking up logic is active. The waking up procedure from Power/Off mode is the same as a new power on cycle. There are only two waking-up source for Power/Off mode, INT0/INT1. Any high level kept as long as 1ms will bring microcontroller back to working state.

Power/Off lock is used to lock wake-up status when system waked up from Power/Off status. User should write Power/Off lock in very early stage, e.g., at the top of **main** function or other user defined initial stage.

Code to demo power/off usage

```
void main(void)
{
   SMCR = 0x08;  // lock wake-up status
   ……
   // user tasks
   ……
   while(1) {
       if(can_be_sleep) {
         SMCR = 0x01;  // sleep mode enable
         SMCR = 0x07;  // set power/off mode
         Sleep();
        } else {
          ……
        }
   }
}
```

Any long enough high level on external INT0/1 can be used to wake up from Power/Off mode. So if a system need to implement Power/Off mode, INT0/1 should be given a default pull down state. After wake-up, those two I/O can be used as its normal functions, like external interrupts or general purpose I/O.

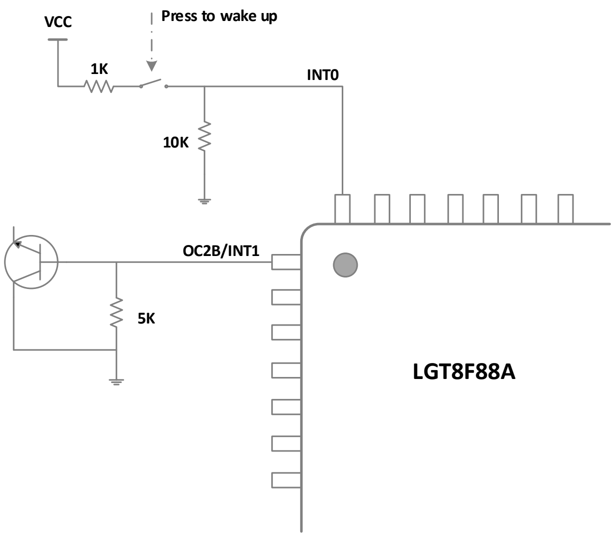
Here is a typical usage of INT0/1 with power/off mode. We use INT0 as a wake-up source for Power/Off, INT1 is configured to drive external circuit. But all of them given a default pull down resistance which is necessary to make Power/off working correctly.

It's also recommended to given a filter capacitance near the wake-up switch to generate smooth input.

[Import Notes]

When PD4/PE5 are not used, they should be given a known state in Power/Off mode to avoid extra leakage.

All other I/O can be floating is not used, but for EMC consideration, it's also recommended to tie to ground.



PERIPHERALS

Most peripherals of LGT8F88A is compatible with ATMega88, including WDT, AC, ADC, SPI, USART, TWI and Timers. Software written for SPI/USART/TWI and Timers can be ported to LGT8F88A without any change. There are only a little difference for WDT, ADC and E2PROM.

For WDT, the only difference is about its clock source. ATMega88 uses a dedicated 128KHz internal RC for WDT. LGT8F88A support two clock source for WDT: internal 32KHz RC oscillator or internal 32MHz RC divided by 16. So you can use **PMCR[4]** to select 32KHz or 2MHz as clock source to drive WDT's counter.

PMCR[4]	Clock source for WDT
0	Internal 32MHz/16 (default)
1	Internal 32KHz

For ADC, the only difference is about its reference voltage configuration. LGT8F88A provide two internal reference voltage, 1.25V and 2.56V, so the **ADMUX[7:6]** has a little difference:

ADMUX[7:6]	Reference voltage for ADC
00	External AVREF as reference
01	Using VCC as reference
10	Internal 1.25V reference
11	Internal 2.56V reference

Additionally, LGT8F88A's ADC has more features than ATMega88:

1. three couple of differential input channels and a shared gain control circuit
2. Internal thermal sensor
3. A dedicated channel for capacitive touch key circuit

To support above features, LGT8F88A extend ADMUX register to include more channels:

ADMUX[4:0]	Single-ended	Differential (+)	Differential (-)
0x00	PC0/ADC0		
0x01	PC1/ADC1		
0x02	PC2/ADC2		
0x03	PC3/ADC3		
0x04	PC4/ADC4		
0x05	PC5/ADC5		
0x06	PE1/ADC6		
0x07	PE3/ADC7		
0x08		ADC3	ADC0
0x09		ADC4	ADC1
0x0A		ADC5	ADC2
0x0B~0x17	Reserved		
0x18	Internal thermal sensor		
0x19	Internal voltage reference (1.25V)		
0x1A	External VREF or touch-key channel		
0x1B~0x1F	Reserved		

The gain parameter control is implemented using a whole new register called ADTMR, which defined as follow:

<table>
<tr><th colspan="9">ADTMR – ADC Test Mode Register</th></tr>
<tr><td colspan="4">Address: 0x7D</td><td colspan="5">Default Value: 0x01</td></tr>
<tr><td colspan="9"></td></tr>
<tr><td>Bit</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Name</td><td>GAIN1</td><td>GAIN0</td><td>-</td><td>-</td><td>-</td><td>ADTM2</td><td>ADTM1</td><td>ADTM0</td></tr>
<tr><td>R/W</td><td>R/W</td><td>R/W</td><td>-</td><td>-</td><td>-</td><td>R/W</td><td>R/W</td><td>R/W</td></tr>
<tr><td>Initial</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td colspan="9"></td></tr>
<tr><td>Bit</td><td>Name</td><td colspan="7">Descriptions</td></tr>
<tr><td rowspan="6">7:6</td><td rowspan="6">GAIN[1:0]</td><td colspan="7">ADC gain control for differential mode</td></tr>
<tr><td colspan="2">GAIN[1:0]</td><td colspan="5">Gain</td></tr>
<tr><td colspan="2">0</td><td colspan="5">7.5</td></tr>
<tr><td colspan="2">1</td><td colspan="5">15</td></tr>
<tr><td colspan="2">2</td><td colspan="5">22.5</td></tr>
<tr><td colspan="2">3</td><td colspan="5">30</td></tr>
<tr><td>5:3</td><td>-</td><td colspan="7">Reserved</td></tr>
<tr><td>2:0</td><td>ADTM[2:0]</td><td colspan="7">ADC test mode select. Leaving it as its default value.</td></tr>
</table>

For E2PROM interface, Read operation has a bit difference: After issue read operation, you should waiting for at least two system cycles for data ready. This is all what you need to care.

FUSE SETTING

LGT8F88A has no FUSE memory inside. It implements fuse-like settings by a spare area of FLASH memory, which is also used to implement our E2PROM emulator. The accessing of this spare area is controlled by hardware. Some cells are only writable during factory test, for example GUID parts and RC calibration bytes which just readable from software side. Other cells can be read or written by user application, including 504 bytes E2PROM emulator memory and several system configuration bytes.

Unlike ATMega88 which has rich fuse settings, LGT8F88A implements much of them by I/O registers and discards some of them by using a totally different way. For example, LGT8F88A have no LOCK bits for flash protection. We using a very different method which based on the status change of flash controller.

The calibration bytes for 32MHz internal RC is programmed while factory test. If you want to trim it for another purpose, you can do it by writing **RCCAL** register in I/O memory.

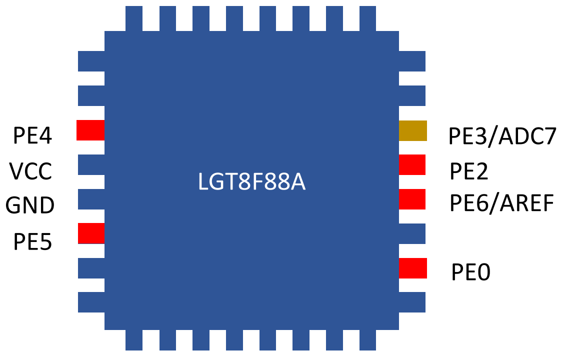
BOR settings can be set by system configuration bytes, which take effect immediately after system power up. But you can also update its settings from I/O register. LGT8F88A support 32MHz with full range of 1.8V ~ 5.5V power supply. In system flash programming can also be done when power supply drop down to 1.8V, so for most applications, BOR is not necessary.

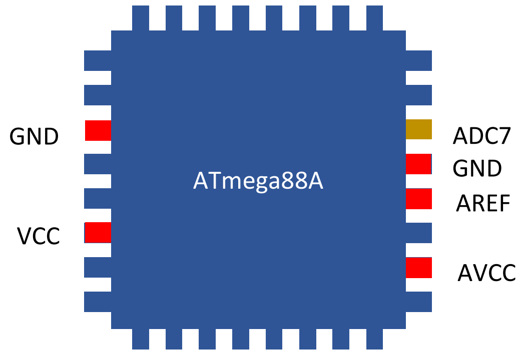
In one word, when using LGT8F88A, just forget FUSE settings, that's totally no problem.

If you want to learn more about system configuration bytes of LGT8F88A, please refer to its programming manual.

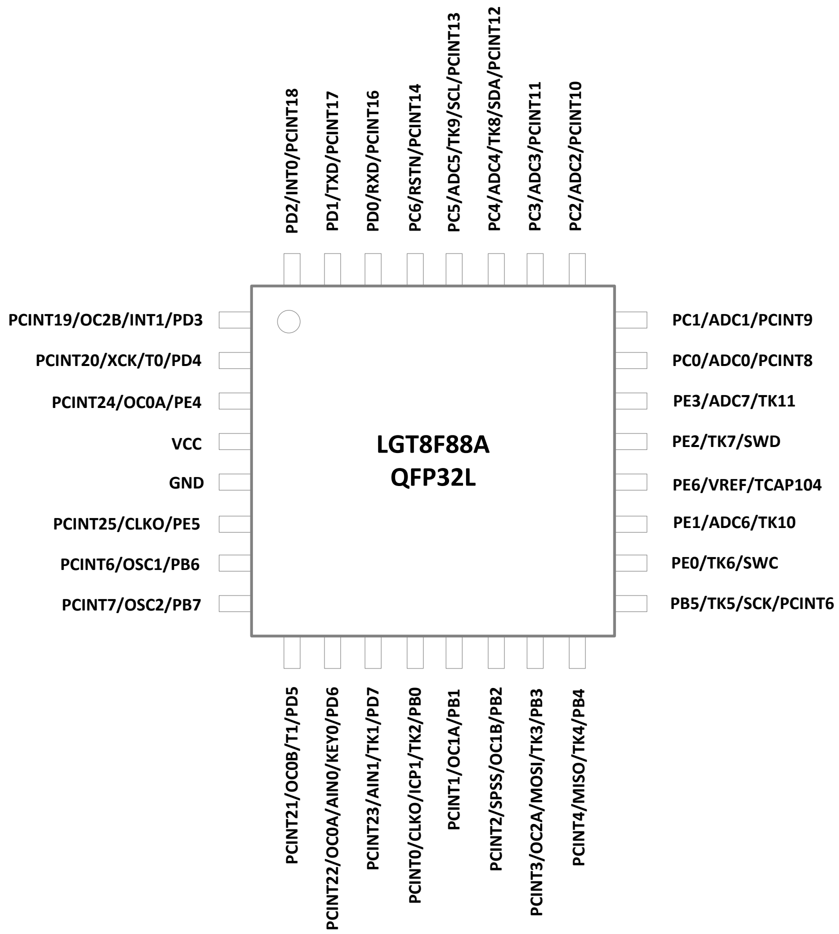
PINOUT AND PACKAGE

LGT8F88A provide QFP32L package which can be fully compatible with ATMega48/88 QFP32L. Some of power Pin of ATMega88 are given different features, so be careful while driving those Pins. LEAVING those Pins as input if they are tied to power supply or ground in print circuit board.





Pin assignment of LGT8F88A:



DEVELOPMENT AND DEBUG

LGT8F88A is fully compatible with ATMega88 from architecture view. It support most of instructions from AVR8L sets, except for ***SPM*** which implemented with another way. So any software development environments for ATMega88 are also working for LGT8F88A. Here we give some of them for example:

AVR Studio 4.18, AVR Studio 6.0 build 1843 & 1996 (support hardware debug)
IAR Workbench for AVR (any version, support hardware debug)
Image craft compiler for AVR
Code Vision AVR C compiler
WinAVR (support debug by GDB)
CrossPack for AVR (for Mac OSX)
AVRGCC tool chain (for Linux)

But for development hardware support, the difference comes again. You cannot use debug or ISP hardware for ATMega88 to connect with LGT8F88A. Instead, we provide our own debugger and ISP hardware.

We provide SWDICE_mkII hardware to support debugger & ISP. This is only hardware you need to do debug and program.

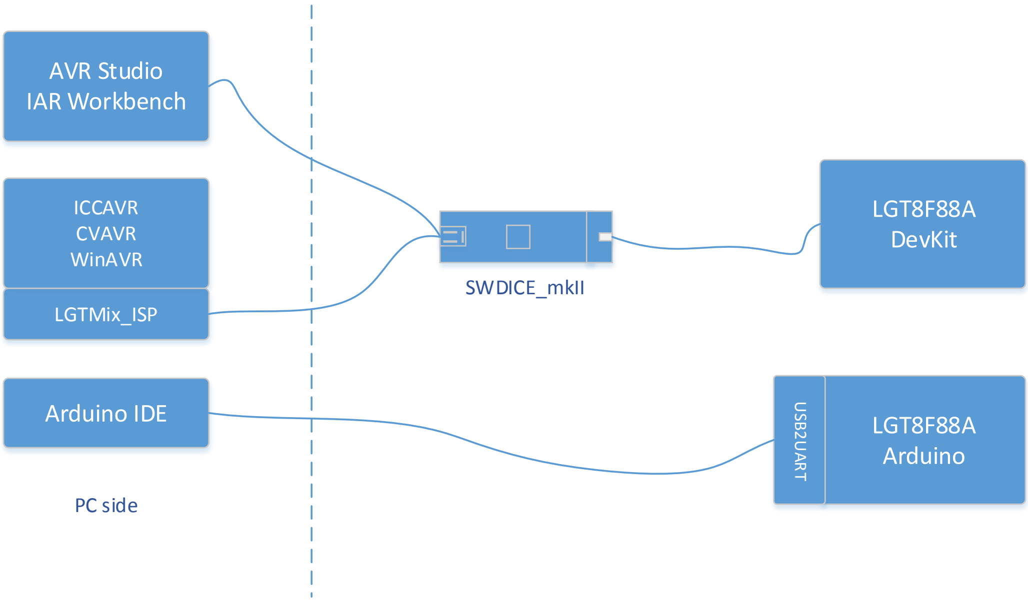
SWDICE_mkII is also compatible with AVR Studio, you can just treat it as JTAGICE_mkII, but is only designed for LGT8F88A. Besides a debugger, SWDICE_mkII can be reused as a programmer hardware.

There are many popular ASP&ISP toolkit for AVR, the **avrdude** is the famous one. SWDICE_mkII can also support **avrdude** through jtag2isp protocol.

We also provide a dedicated ISP toolkit for LGT8F88A, LGTMix_ISP, can be get from our official web site.

LGTMix_ISP support SWDICE_mkII, can be used as a full feature ISP toolkit for LGT8F88A and also for all other microcontrollers designed by LogicGreen.

For usage details of LGTMix_ISP toolkit, please refer to "In system programming" section.



As shown from above picture, ICCAVR and CVAVR which have no debugger and programmer support can be used together with LGTMix_ISP and SWDICE_mkII to burn generated firmware code to target boarder.

We have also port ***optiboot*** to support LGT8F88A, so if you got a LGT8F88A Arduino compatible board, you can update firmware by Arduino or avrdude directly.

More words for SWDICE_mkII:

When using SWDICE_mkII as debugger, we provide three version of firmware to support AVR Studio 4.18, Studio 6.0 build 1843 and Studio build 1996. You should update SWDICE_mkII firmware according to the Studio version you select.

For IAR Workbench and LGTMix_ISP, any version of SWDICE_mkII is ok.

Steps to update firmware of SWDICE_mkII:

1. Get right firmware from LogicGreen official web site
2. Find update jumper at the backside of SWDICE_mkII, short it
3. Connect SWDICE_mkII to PC via a micro-USB cable
4. Waiting for an USB storage driver, it's about 32Kbyte
5. Copy firmware binary code to this driver, then unplug from PC
6. Remove the shorter, now it's ready.

IN SYSTEM PROGRAMMING

In this section, we will talk more about official ISP toolkit – LGTMix_ISP.

LGT8F88A integrates an in system programming and on-chip debugger logic inside. A serial wire debug (SWD) interface is exported to communication with outer world. SWD follows a two-wire protocol, using PE0 as SWC for clock signal and PE2 as SWD for transfer data. Please refer to "Pinout and Package" section for its location.

SWDICE_mkII is the only official hardware to support debugger and programmer for LGT8F88A. As we talked in above section, SWDICE_mkII can be used with AVR Studio and IAR workbench to debug LGT8F88A, just like using JTAGICE_mkII to debug ATMega88.

For programming, we provide a full featured in system programming toolkit, LGTMix_ISP. Which is also maintained to support our following 8bit microcontroller products.

LGTMix_ISP need SWDICE_mkII to act as a programmer hardware to communicate with LGT8F88A. LGTMix_ISP talks with SWDICE_mkII via USB protocol, implemented by libusb interface. But consider to be compatible with ATMEL's JTAGICE_mkII driver for debug, it's recommended to install libusb filter driver at top of ATMEL's driver.

So here is all of the software and hardware you needed to kick start:

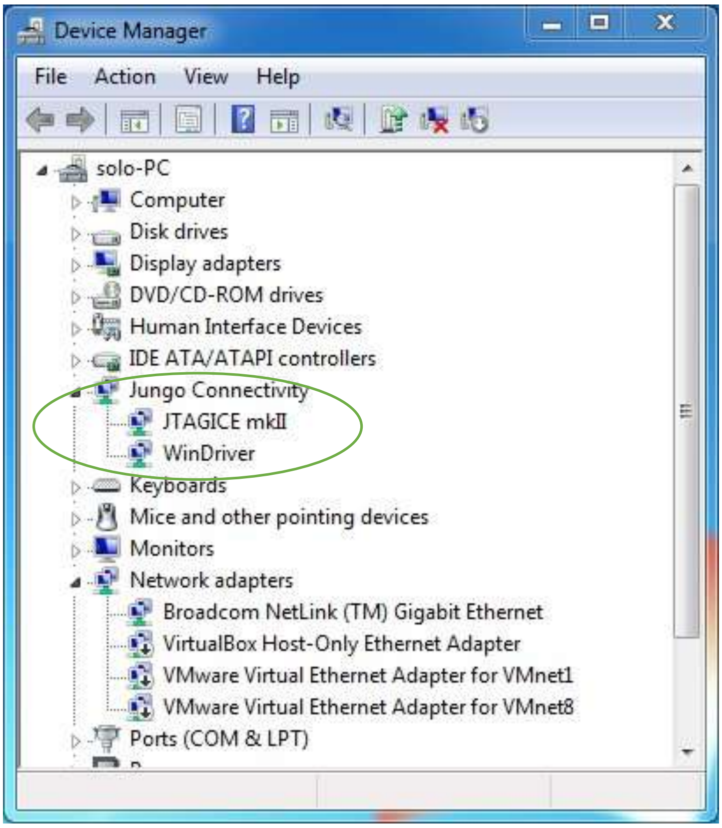
1. LGTMix_ISP toolkit installation file. (include libusb filter driver)
2. ATMEL JTAGICE_mkII driver (installed with AVR Studio, but also be got standalone from here)
3. SWDICE_mkII hardware
4. LGT8F88A development boarder

Steps to use above resources:

1. Install ATMEL JTAGICE_mkII driver. If you have installed AVR Studio, skip to next step.
2. Connect SWDICE_mkII to PC, check if it can be found rightly.
3. Install LGTMix_ISP. That's easy, click and follow all its next step to complete.
4. Start libusb filter driver installer from start menu, LGTMix_ISP program group
5. After filter installation, all prepare works are done.

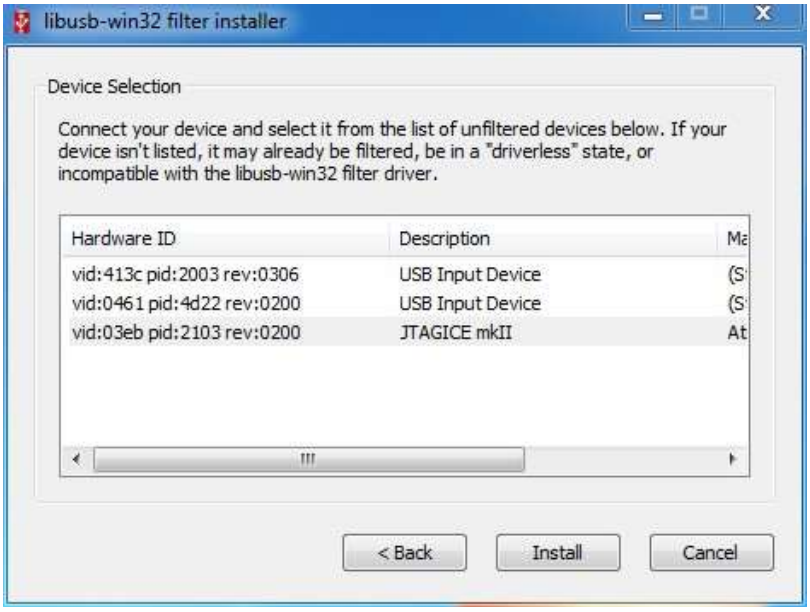
Now let's show you more details with picture demo:

1. If JTAGICE_mkII installed properly, connect SWDICE_mkII, you can find our device inside windows device manager:



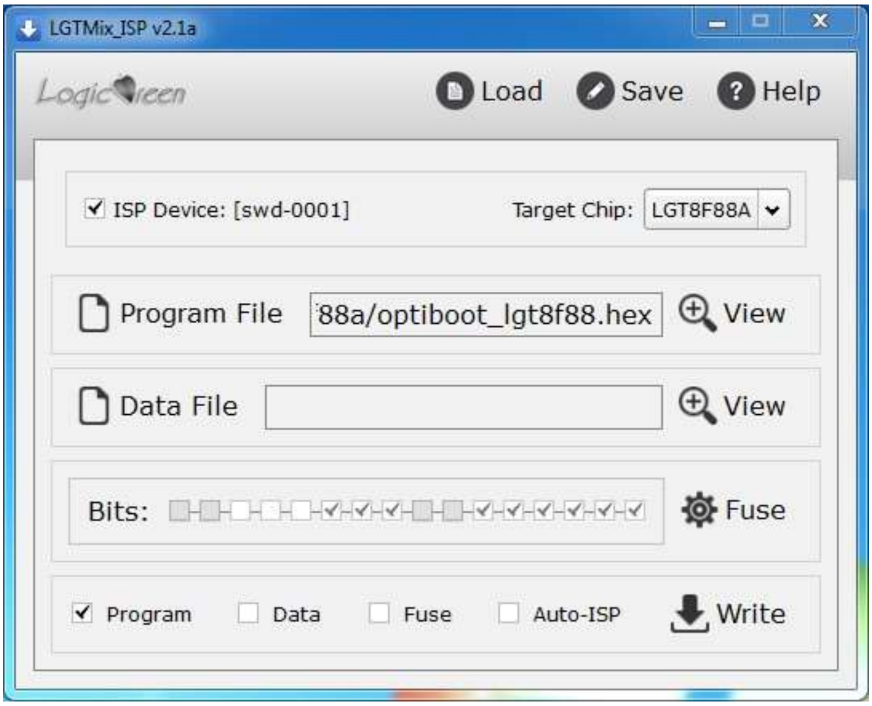
2. After LGTMix_ISP installed, you can start "SWDICE_mkII filter installer" from start menu-> LogicGreen-> LGTMix_ISP->SWDICE_mkII filter installer. It then just start libusb-win32 filter installer.
   Firstly, select "Install a device filter"

Then direct to next step, and select JTAGICE_mkII device you find.

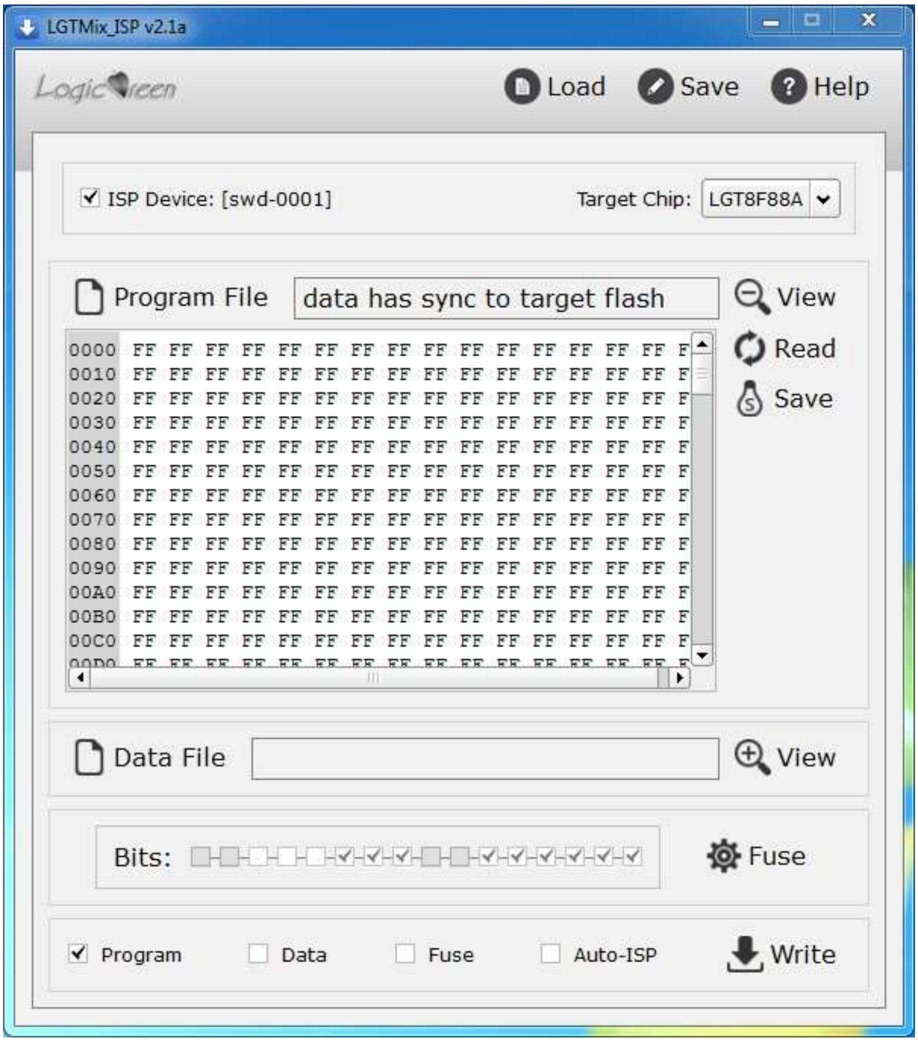


After select the "JTAGICE_mkII", click [Install] button, the filter driver will installed and configured automatically. Wait a while, close libusb-win32 filter installer after installation is completed.

3. Now, it's time to connect SWDICE_mkII with your target board, power it on. Then open LGTMix_ISP from Start Menu -> LogicGreen -> LGTMix_ISP. If anything goes well, LGTMix_ISP will find SWDICE_mkII and identify the target chip automatically:



4. If you see above window, congratulations! Everything works now. Then you can try to expand the [View] button and read data from target chip.



You can also select your local firmware code (hex or binary) by click [Program File] button, and make sure the Program check box is selected, click [Write] button to start programming.

REGISTERS INDEX

Address	Name	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Extended IO Register									
$F6	GUID3	GUID Byte 3							
$F5	GUID2	GUID Byte 2							
$F4	GUID1	GUID Byte 1							
$F3	GUID0	GUID Byte 0							
$F2	PMCR	PMCE	LFEN	EXTEN	WCES	OSCKEN	OSCMEN	RCKEN	RCMEN
$F1	DSCR	DSCE	-	-	DSC4	DSC3	DSC2	DSC1	DSC0
$F0	IOCR	IOCE	-	-	-	-	-	REFIOEN	RSTIOEN
$E2	PSSR	PSS1	-	-	-	-	-	-	PSR1
$CF	DIDR3	-	-	-	-	TIN11D	TIN10D	TIN9D	TIN8D
$CE	DIDR2	TIN7D	TIN6D	TIN5D	TIN4D	TIN3D	TIN2D	TIN1D	TIN0D
$CD	TKCSR	TKPD	TKPSEL			TKMUX			
$C6	UDR0	USART Data							
$C5	UBRR0H	-	-	-	-	USART Baud Rate Register High			
$C4	UBRR0L	USART Baud Rate Register Low							
$C2	UCSR0C	UMSEL0		UPM0		USBS0	UCSZ01/ UDORD0	UCSZ00/ UCPHA0	UCPOL0
$C1	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
$C0	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
$BD	TWAMR	TWI Address Mask							-
$BC	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
$BB	TWDR	TWI Data							
$BA	TWAR	TWI Address							TWGCE
$B9	TWSR	TWI Status					-	TWPS	
$B8	TWBR	TWI Bit Rate							
$B6	ASSR	-	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB
$B4	OCR2B	Timer/Counter 2 Output Compare Register B							
$B3	OCR2A	Timer/Counter 2 Output Compare Register A							
$B2	TCNT2	Timer/Counter 2 Counter Register							
$B1	TCCR2B	FOC2A	FOC2B	-	-	WGM22	CS2		
$B0	TCCR2A	COM2A		COM2B		-	-	WGM21	WGM20

$A9	PORTE	Port Output E							
$A8	DDRE	Data Direction E							
$A7	PINE	Port Input E							
$8B	OCR1BH	Timer/Counter 1 Output Compare B High							
$8A	OCR1BL	Timer/Counter 1 Output Compare B Low							
$89	OCR1AH	Timer/Counter 1 Output Compare A High							
$88	OCR1AL	Timer/Counter 1 Output Compare A Low							
$87	ICR1H	Timer/Counter 1 Input Capture High							
$86	ICR1L	Timer/Counter 1 Input Capture Low							
$85	TCNT1H	Timer/Counter 1 Counter High							
$84	TCNT1L	Timer/Counter 1 Counter Low							
$82	TCCR1C	FOC1A	FOC1B	-	-	-	-	-	-
$81	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS1		
$80	TCCR1A	COM1A		COM1B		-	-	WGM11	WGM10
$7F	DIDR1	-	-	-	-	-	-	AIN1D	AIN0D
$7E	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
$7D	ADTMR	GAIN		-	-	-	ADTM		
$7C	ADMUX	REFS		ADLAR	-	MUX			
$7B	ADCSRB	-	ACME	-	ICTL	-	ADTS		
$7A	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS		
$79	ADCH	ADC Data High							
$78	ADCL	ADC Data Low							
$77	EEDRH	EEPROM Data High							
$75	IVBASE	Interrupt Vector Base Address							
$70	TIMSK2	-	-	-	-	-	OCIE2B	OCIE2A	TOIE2
$6F	TIMSK1	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1
$6E	TIMSK0	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
$6D	PCMSK2	PCINT[23:16]							
$6C	PCMSK1	PCINT[15:8]							
$6B	PCMSK0	PCINT[7:0]							
$69	EICRA	-	-	-	-	ISC1		ISC0	
$68	PCICR	-	-	-	-	-	PCIE2	PCIE1	PCIE0
$66	OSCCAL	-	-	OSC Calibration					
$65	PRR1	-	-	PRWDT	-	-	PREFL	PRPCI	-
$64	PRR	PRTWI	PRTIM2	PRTIM0	-	PRTIM1	PRSPI	PRUSART0	PRADC

$62	VDTCR	VDTCE	SWRSTN	-	-	-	VDTSEL		VDTEN
$61	CLKPR	CLKPCE	CLKOEN0	CLKOEN1	-	CLKPS			
$60	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
$5F($3F)	SREG	I	T	H	S	V	N	Z	C
$5E($3E)	SPH	Stack point high byte							
$5D($3D)	SPL	Stack point low byte							
$55($35)	MCUCR	-	BODS	BODSE	PUD	-	-	IVSEL	IVCE
$54($34)	MCUSR	SWDD	-	-	OCDRF	WDRF	BORF	EXTRF	PORF
$53($33)	SMCR	-	-	-	-	SM			SE
$50($30)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS	
$4E(0x2E)	SPDR	SPI Data Register							
$4D($2D)	SPSR	SPIF	WCOL	-	-	-	DUAL	-	SPI2X
$4C($2C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR	
$4B($2B)	GPIOR2	General purpose I/O register 2							
$4A($2A)	GPIOR1	General purpose I/O register 1							
$48($28)	OCR0B	Timer/counter 0 output compare register B							
$47($27)	OCR0A	Timer/counter 0 output compare register A							
$46($26)	TCNT0	Timer/Counter 0 counter							
$45($25)	TCCR0B	FOC0A	FOC0B	OC0AS	-	WGM02	CS0		
$44($24)	TCCR0A	COM0A		COM0B		-	-	WGM01	WGM00
$43($23)	GTCCR	TSM	-	-	-	-	-	PSRASY	PSRSYNC
$42($22)	EEARH	EEPROM Address high byte							
$41($21)	EEARL	EEPROM Address low byte							
$40($20)	EEDR	EEPROM Data							
$3F($1F)	EECR	EEPM2	-	EEPM1	EEPM0	EERIE	EEMWE	EEWE	EERE
$3E($1E)	GPIOR0	General purpose IO register 0							
$3D($1D)	EIMSK	-	-	-	-	-	-	INT1	INT0
$3C($1C)	EIFR	-	-	-	-	-	-	INTF1	INTF0
$3B($1B)	PCIFR	-	-	-	-	-	PCIF2	PCIF1	PCIF0
$37($17)	TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2
$36($16)	TIFR1	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1
$35($15)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0
$2B($0B)	PORTD	Port output D							
$2A($0A)	DDRD	Data direction D							

\$29(\$09)	PIND	Port input D
\$28(\$08)	PORTC	Port output C
\$27(\$07)	DDRC	Port direction C
\$26(\$06)	PINC	Port input C
\$25(\$05)	PORTB	Port output B
\$24(\$04)	DDRB	Port direction B
\$23(\$03)	PINB	Port input B

INSTRUCTION INDEX

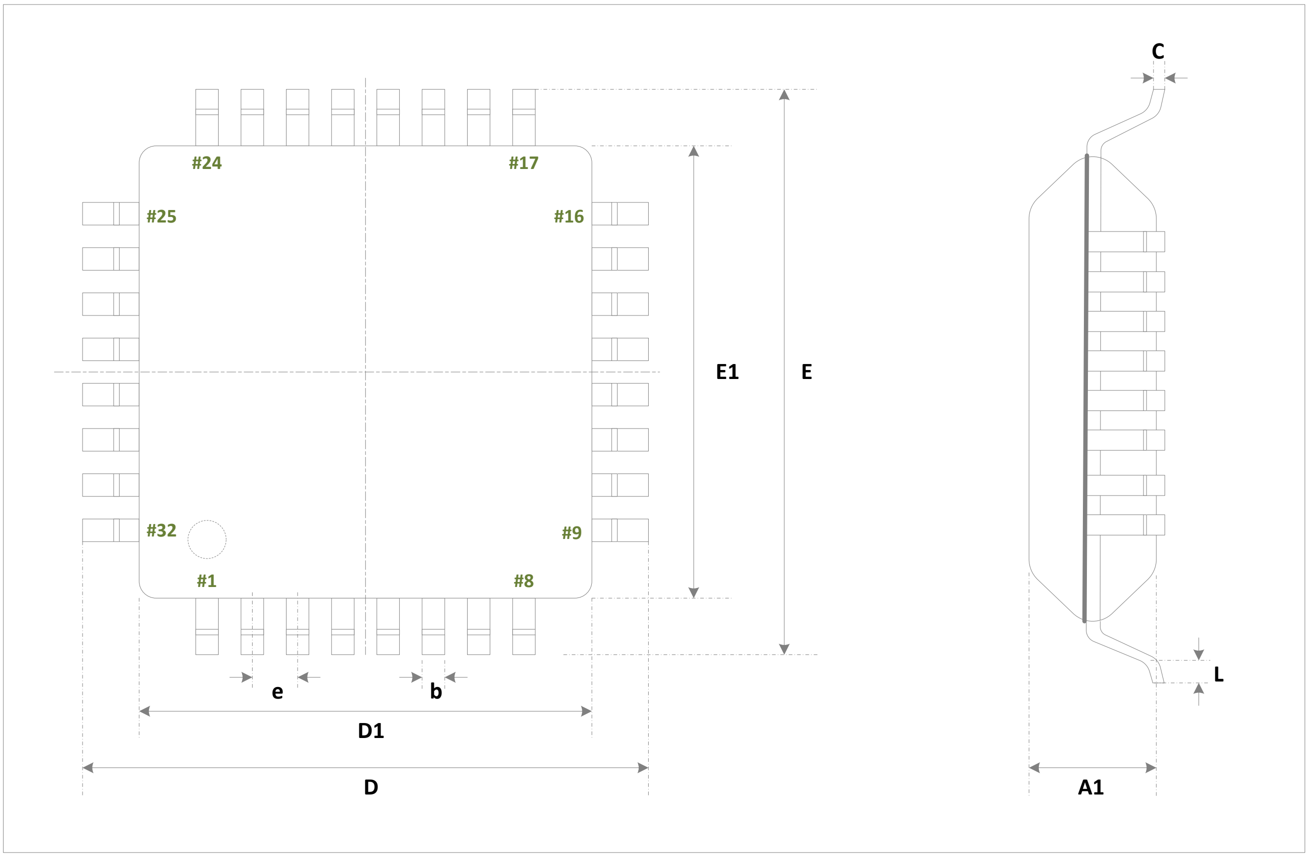
INST.	OPC.	FUNCITONS	OPERATION	FLAG	CYCLE
Arithmetic and Logic operation					
ADD	R_d, R_r	Add two registers	$R_d \leftarrow R_d + R_r$	Z,C,N,V,H	1
ADC	R_d, R_r	Add with carry two regiters	$R_d \leftarrow R_d + R_r + C$	Z,C,N,V,H	1
ADIW	R_{dl}, K	Add immediate to word	$R_{dh}:R_{dl} \leftarrow R_{dh}:R_{dl} + K$	Z,C,N,V,S	1
SUB	R_d, R_r	Subtract two registers	$R_d \leftarrow R_d - R_r$	Z,C,N,V,H	1
SUBI	R_d, K	Subtract constant from registers	$R_d \leftarrow R_d - K$	Z,C,N,V,H	1
SBC	R_d, R_r	Subtract with carry	$R_d \leftarrow R_d - R_r - C$	Z,C,N,V,H	1
SBCI	R_d, K	Subtract with carry constant	$R_d \leftarrow R_d – K - C$	Z,C,N,V,H	1
SBIW	R_{dl}, K	Subtract immediate from word	$R_{dh}:R_{dl} \leftarrow R_{dh}:R_{dl} - K$	Z,C,N,V,S	1
AND	R_d, R_r	Logical AND	$R_d \leftarrow R_d \& R_r$	Z,N,V	1
ANDI	R_d, K	Logical AND register and constant	$R_d \leftarrow R_d \& K$	Z,N,V	1
OR	R_d, R_r	Logical OR	$R_d \leftarrow R_d \mid R_r$	Z,N,V	1
ORI	R_d, K	Logical OR register and constant	$R_d \leftarrow R_d \mid K$	Z,N,V	1
EOR	R_d, R_r	Exclusive OR	$R_d \leftarrow R_d \oplus R_r$	Z,N,V	1
COM	R_d	One’s complement	$R_d \leftarrow \$FF - R_d$	Z,C,N,V	1
NEG	R_d	Two’s complement	$R_d \leftarrow \$00 - R_d$	Z,C,N,V,H	1
SBR	R_d, K	Set bit(s) in Register	$R_d \leftarrow R_d \vee K$	Z,N,V	1
CBR	R_d, K	Clear bit(s) in Rigister	$R_d \leftarrow R_d \vee (\$FF – K)$	Z,N,V	1
INC	R_d	Increment	$R_d \leftarrow R_d + 1$	Z,N,V	1
DEC	R_d	Decrement	$R_d \leftarrow R_d - 1$	Z,N,V	1
TST	R_d	Test for zero or minus	$R_d \leftarrow R_d \& R_d$	Z,N,V	1
CLR	R_d	Clear register	$R_d \leftarrow R_d \oplus R_d$	Z,N,V	1
SER	R_d	Set register	$R_d \leftarrow \$FF$	None	1
MUL	R_d, R_r	Multiply unsigned	$R_1: R_0 \leftarrow R_d \times R_r$	Z,C	1
MULS	R_d, R_r	Multiply signed	$R_1: R_0 \leftarrow R_d \times R_r$	Z,C	1
MULSU	R_d, R_r	Multiply signed with unsigned	$R_1: R_0 \leftarrow R_d \times R_r$	Z,C	1
FMUL	R_d, R_r	Fractional MUL	$R_1: R_0 \leftarrow (R_d \times R_r) << 1$	Z,C	1
FMULS	R_d, R_r	Fractional MULS	$R_1: R_0 \leftarrow (R_d \times R_r) << 1$	Z,C	1
FMULSU	R_d, R_r	Fractional MULSU	$R_1: R_0 \leftarrow (R_d \times R_r) << 1$	Z,C	1
Branch Instructions					
RJMP	K	Relative jump	$PC \leftarrow PC + K + 1$	None	1
IJMP		Indirect jump to (Z)	$PC \leftarrow Z$	None	2
JMP	K	Direct jump	$PC \leftarrow K$	None	2
RCALL	K	Relative subroutine call	$PC \leftarrow PC + K + 1$	None	1
ICALL		Indirect call to (Z)	$PC \leftarrow Z$	None	2
CALL	K	Direct subroutine call	$PC \leftarrow K$	None	2
RET		Subroutine return	$PC \leftarrow Stack$	None	2
RETI		Interrupt return	$PC \leftarrow Stack$	I	2

INST.	OPC.	FUNCITONS	OPERATION	FLAG	CYCLE
Branch Instructions (Cont'd)					
CPSE	R_d, R_r	Compare, skip if equal	If(R_d=R_r) PC ← PC + 2 or 3	None	1/2
CP	R_d, R_r	Compare	R_d - R_r	Z,N,V,C,H	1
CPC	R_d, R_r	Compare with carry	R_d - R_r – C	Z,N,V,C,H	1
CPI	R_d, K	Compare with immediate	R_d - K	Z,N,V,C,H	1
SBRC	R_r, b	Skip if bit in register cleared	If(R_r(b)=0) PC ← PC + 2 or 3	None	1/2
SBRS	R_r, b	Skip if bit in register set	If(R_r(b)=1) PC ← PC + 2 or 3	None	1/2
SBIC	P, b	Skip if bit in I/O cleared	If(P(b)=0) PC ← PC + 2 or 3	None	1/2
SBIS	P, b	Skip if bit in I/O set	If(P(b)=1) PC ← PC + 2 or 3	None	1/2
BRBS	s, k	Branch if status flag set	If(SREG(S)=1) PC ← PC + K + 1	None	1/2
BRBC	s, k	Branch if status flag cleared	If(SREG(S)=0) PC ← PC + K + 1	None	1/2
BREQ	k	Branch if equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if not equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if carry set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if carry cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if same or higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if greater or equal, signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if less than zerio, signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if half carry flag set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if half carry flag cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T flag set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if overflow flag is set	f (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if overflow flag cleared	f (V = 0) then PC ← PC + k + 1	None	1/2
BRIE	k	Branch if interrupt enabled	f (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if interrupt disabled	f (I = 0) then PC ← PC + k + 1	None	1/2
DATA TRANSFER Instructions					
MOV	Rd, Rr	Move between registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy register word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load immediate	Rd ← K	None	1
LD	Rd, X	Load indirect	Rd ← (X)	None	1
LD	Rd, X+	Load indirect and post-inc.	Rd ← (X), X ← X + 1	None	1
LD	Rd, -X	Load indirect and pre-dec	X ← X - 1, Rd ← (X)	None	1
LD	Rd, Y	Load indirect	Rd ← (Y)	None	1
LD	Rd, Y+	Load indirect and post-inc	Rd ← (Y), Y ← Y + 1	None	1
LD	Rd, -Y	Load indirect and pre-dec	Y ← Y - 1, Rd ← (Y)	None	1
LDD	Rd, Y+q	Load indirect with displacement	Rd ← (Y + q)	None	1

LD	Rd, Z	Load indirect	Rd ← (Z)	None	1
LD	Rd, Z+	Load indirect and post-inc	Rd ← (Z), Z ← Z+1	None	1
LD	Rd, -Z	Load indirect and pre-dec	Z ← Z - 1, Rd ← (Z)	None	1
LDD	Rd, Z+q	Load indirect with displacement	Rd ← (Z + q)	None	1
LDS	Rd, k	Load direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store indirect	(X) ← Rr	None	1
ST	X+, Rr	Store indirect and post-inc	(X) ← Rr, X ← X + 1	None	1
ST	-X, Rr	Store indirect and pre-dec	X ← X - 1, (X) ← Rr	None	1
ST	Y, Rr	Store indirect	(Y) ← Rr	None	1
ST	Y+, Rr	Store indirect and post-inc	(Y) ← Rr, Y ← Y + 1	None	1
ST	-Y, Rr	Store indirect and pre-dec	Y ← Y - 1, (Y) ← Rr	None	1
STD	Y+q, Rr	Store indirect with displacement	(Y + q) ← Rr	None	1
ST	Z, Rr	Store indirect	(Z) ← Rr	None	1
ST	Z+, Rr	Store indirect and post-inc	(Z) ← Rr, Z ← Z + 1	None	1
ST	-Z, Rr	Store indirect and pre-dec	Z ← Z - 1, (Z) ← Rr	None	1
STD	Z+q, Rr	Store indirect with displacement	(Z + q) ← Rr	None	1
STS	k, Rr	Store direct	(k) ← Rr	None	2
LPM		Load program memory	R0 ← (Z)	None	2
LPM	Rd, Z	Load program memory	Rd ← (Z)	None	2
LPM	Rd, Z+	Load program and post-inc	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, Z+	Load	Rd ← (Z), Z ← Z+1	None	1
LD	Rd, -Z	Load indirect and pre-dec	Z ← Z - 1, Rd ← (Z)	None	1
LDD	Rd, Z+q	Load indirect with displacement	Rd ← (Z + q)	None	1
LDS	Rd, k	Load direct from SRAM	Rd ← (k)	None	2
IN	Rd, P	In port	Rd ← P	None	1
OUT	P, Rr	Out port	P ← Rr	None	1
PUSH	Rr	Push register on stack	STACK ← Rr	None	1
POP	Rd	Pop register from stack	Rd ← STACK	None	1
BIT and BIT-TEST Instructions					
SBI	P, b	Set bit in I/O register	I/O(P, b) ← 1	None	1
CBI	P, b	Clear bit in I/O register	I/O(P, b) ← 0	None	1
LSL	Rd	Logical shift left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical shift right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z	1
ROL	Rd	Rotate left through carry	Rd(0)←C, Rd(n+1) ← Rd(n), C←Rd(7)	Z	1
ROR	Rd	Rotate right through carry	Rd(7)←C, Rd(n) ← Rd(n+1), C←Rd(0)	Z	1
ASR	Rd	Arithmetic shift right	Rd(n) ← Rd(n+1), n=0:6	Z	1
SWAP	Rd	Swap nibbles	Rd(3:0) ← Rd(7:4), Rd(7:4) ← Rd(3:0)	None	1
BSET	s	Flag set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit store from register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1

CLC		Clear carry	C ← 0	C	1
SEN		Set negative flag	N ← 1	N	1
CLN		Clar negative flag	N ← 0	N	1
SEZ		Set zero flag	Z ← 1	Z	1
CLZ		Clear zero flag	Z ← 0	Z	1
SEI		Global interrupt enable	I ← 1	I	1
CLI		Global interrupt disable	I ← 0	I	1
SES		Set signed test flag	S ← 1	S	1
CLS		Clear signed test flag	S ← 0	S	1
SEV		Set 2’s complement overflow	V ← 1	V	1
CLV		Clear 2’s complement overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
MCU Control Instructions					
NOP		No operation		None	1
SLEEP		Sleep		None	1
WDR		Watchdog reset		None	1
BREAK		Software break	Only for debug purpose	None	N/A

PACKAGE DEFINITION



LQFP32L Dimension

Simboly	Min.	Typical.	Max.	Unit
D	8.90	9.00	9.10	mm
D1	6.90	7.00	7.10	mm
b	0.15	0.20	0.25	mm
e	0.75	0.80	0.85	mm
E	8.90	9.00	9.10	mm
E1	6.90	7.00	7.10	mm
C	-	0.10	-	mm
L	0.55	0.60	0.65	mm
A1	-	1.40	-	mm