This is the final list of new Registers that the AVR does not have.

| Addr | Name | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|---|
| 0xF6 | GUID3 | GUID Byte 3 | | | | | | | |
| 0xF5 | GUID2 | GUID Byte 2 | | | | | | | |
| 0xF4 | GUID1 | GUID Byte 1 | | | | | | | |
| 0xF3 | GUID0 | GUID Byte 0 | | | | | | | |
| 0xF2 | PMCR | PMCE | CLKFS | CLKSS | WCLKS | OSCKEN | OSCMEN | RCKEN | RCMEN |
| 0xF0 | PMX2 | WCE | STOSC1 | STOSC0 | - | - | XIEN | E6EN | C6EN |
| 0xEE | PMX0 | PMXCE | C1BF4 | C1AF5 | C0BF3 | C0AC0 | SSB1 | TXD6 | RXD5 |
| 0xED | PMX1 | - | - | - | - | - | C3AC | C2BF7 | C2AF6 |
| 0xEC | TCKCSR | - | F2XEN | TC2XF1 | TC2XF0 | - | AFCKS | TC2XS1 | TC2XS0 |
| 0xE2 | PSSR | PSS1 | PSS3 | - | - | - | - | PSR3 | PSR1 |
| 0xE1 | OCPUE | PUE7 | PUE6 | PUE5 | PUE4 | PUE3 | PUE2 | PUE1 | PUE0 |
| 0xE0 | HDR | - | - | HDR5 | HDR4 | HDR3 | HDR2 | HDR1 | HDR0 |
| 0xDE | DAPTE | DAPTE | - | - | - | - | - | - | - |
| 0xDD | DAPTR | DAPTP | | | DAP Trimming | | | | |
| 0xDC | DAPCR | DAPEN | GA1 | GA0 | DNS2 | DNS1 | DNS0 | DPS1 | DPS0 |
| 0xCF | LDOCR | WCE | | | | PDEN | VSEL2 | VSEL1 | VSEL0 |
| 0xCE | VCAL2 | Calibration value for 2.048V internal reference | | | | | | | |
| 0xCD | VCAL1 | Calibration value for 1.024V internal reference | | | | | | | |
| 0xCC | VCAL3 | Calibration value for 4.096V internal reference | | | | | | | |
| 0xC8 | VCAL | Internal Voltage Reference calibration register | | | | | | | |
| 0xAF | DPS2R | - | - | - | - | DPS2E | LPRCE | TOS1 | TOS0 |
| 0xAE | IOCWK | IOCD7 | IOCD6 | IOCD5 | IOCD4 | IOCD3 | IOCD2 | IOCD1 | IOCD0 |
| 0xAD | ADCSRD | BGEN | REFS2 | IVSEL1 | IVSEL0 | - | VDS2 | VDS1 | VDS0 |
| 0xAC | ADMSC | AMOF | - | - | - | AMFC3 | AMFC2 | AMFC1 | AMFC0 |
| 0xAB | ADT1H | ADC Auto-monitor Overflow threshold high byte | | | | | | | |
| 0xAA | ADT1L | ADC Auto-monitor Overflow threshold low byte | | | | | | | |
| 0xA9 | PORTE | Port Output E (for compatible with LGT8FX8D) | | | | | | | |
| 0xA8 | DDRE | Data Direction E (for compatible with LGT8FX8D) | | | | | | | |
| 0xA7 | PINE | Port Input E (for compatible with LGT8FX8D) | | | | | | | |
| 0xA6 | ADT0H | ADC Auto-monitor Underflow threshold high byte | | | | | | | |
| 0xA5 | ADT0L | ADC Auto-monitor Underflow threshold low byte | | | | | | | |
| 0xA4 | OFR1 | ADC positive offset trimming | | | | | | | |
| 0xA3 | OFR0 | ADC negative offset trimming | | | | | | | |
| 0xA1 | DALR | DAC data register | | | | | | | |
| 0xA0 | DACON | - | - | - | - | DACEN | DAOE | DAVS1 | DAVS0 |
| 0x9F | OCR3CH | Compare output register high byte of Timer3 C channel | | | | | | | |
| 0x9E | OCR3CL | Compare output register low byte of Timer3 C channel | | | | | | | |
| 0x9D | DTR3H | Dead-band register high byte of Timer3 | | | | | | | |
| 0x9C | DTR3L | Dead-band register low byte of Timer3 | | | | | | | |
| 0x9B | OCR3BH | Compare output register high byte of Timer3 B channel | | | | | | | |
| 0x9A | OCR3BL | Compare output register low byte of Timer3 B channel | | | | | | | |
| 0x99 | OCR3AH | Compare output register high byte of Timer3 A channel | | | | | | | |
| 0x98 | OCR3AL | Compare output register low byte of Timer3 A channel | | | | | | | |
| 0x97 | ICR3H | Input capture register high byte of Timer3 | | | | | | | |
| 0x96 | ICR3L | Input capture register low byte of Timer3 | | | | | | | |
| 0x95 | TCNT3H | Counter register high byte of Timer3 | | | | | | | |
| 0x94 | TCNT3L | Counter register low byte of Timer3 | | | | | | | |
| 0x93 | TCCR3D | Control register D of Timer3 | | | | | | | |
| 0x92 | TCCR3C | Control register C of Timer3 | | | | | | | |
| 0x91 | TCCR3B | Control register B of Timer3 | | | | | | | |
| 0x90 | TCCR3A | Control register A of Timer3 | | | | | | | |
| 0x8D | DTR1H | Dead-band register high byte of Timer1 | | | | | | | |
| 0x8C | DTR1L | Dead-band register low byte of Timer1 | | | | | | | |
| 0x83 | TCCR1D | DSX17 | DSX16 | DSX15 | DAX14 | - | - | DSX11 | DSX10 |
| 0x7D | ADCSRC | OFEN | - | SPN | AMEN | - | SPD | DIFS | ADTM |
| 0x76 | DIDR2 | - | PB5D | - | - | - | - | - | - |
| 0x75 | IVBASE | Interrupt Vector Base Address | | | | | | | |
| 0x74 | PCMSK4 | | | | | | | | |
| 0x73 | PCMSK3 | PCINT[39:32] | | | | | | | |
| 0x71 | TIMSK3 | | | ICIE3 | - | OCIE3C | OCIE3B | OCIE3A | TOIE3 |
| 0x67 | RCKCAL | RC32K Calibration | | | | | | | |
| 0x65 | PRR1 | - | - | PRWDT | - | PRTIM3 | PREFL | PRPCI | - |
| 0x62 | VDTCR | WCE | SWR | - | | VDTS | | VDREN | VDTEN |
| 0x5C | E2PD3 | E2PCTL Data register byte 3 | | | | | | | |
| 0x5B | C1TR | AC1 trimming data | | | | | | | |
| 0x5A | E2PD1 | E2PCTL Data register byte1 | | | | | | | |
| 0x59 | DSAH | DSA[31:16] access port of uDSC | | | | | | | |
| 0x58 | DSAL | DSA[15:0] access port of uDSC | | | | | | | |
| 0x56 | ECCR | WEN | EEN | ERN | SWM | CP1 | CP0 | ECS1 | ECS0 |
| 0x52 | C0TR | AC0 Trimming register | | | | | | | |
| 0x51 | C0XR | - | C0OE | C0HYSE | C0PS0 | C0WKE | C0FEN | C0FS1 | C0FS0 |
| 0x4F | DTR0 | TC0 Dead-band timing control register | | | | | | | |
| 0x49 | TCCR0C | DSX07 | DSX06 | DSX05 | DSX04 | - | - | DSX01 | DSX00 |
| 0x3A | C1XR | - | C1OE | C1HYSE | C1PS0 | C1WKE | C1FEN | C1FS1 | C1FS0 |
| 0x39 | SPFR | RDFULL | RDEMPT | RDPTR1 | RDPTR0 | WRFULL | WREMPT | WRPTR1 | WRPTR0 |
| 0x38 | TIFR3 | - | - | ICF3 | - | - | OCF3B | OCF3A | TOV3 |
| 0x34 | PORTF | Port Output of Group F | | | | | | | |
| 0x33 | DDRF | Data Direction of Group F | | | | | | | |
| 0x32 | PINF | Port Input of Group F | | | | | | | |
| 0x31 | DSDY | DSDY access port of uDSC | | | | | | | |
| 0x30 | DSDX | DSDX access port of uDSC | | | | | | | |
| 0x2F | C1SR | C1D | C1BG | C1O | C1I | C1IE | C1IC | C1IS | |
| 0x2E | PORTE | Port Output of Group E | | | | | | | |
| 0x2D | DDRE | Data Direction of Group E | | | | | | | |
| 0x2C | PINE | Port Input of Group E | | | | | | | |
| 0x22 | DSSD | DSSD access port of uDSC | | | | | | | |
| 0x21 | DSIR | Instruction regiter of uDSC | | | | | | | |
| 0x20 | DSCR | DSUEN | MM | D1 | D0 | - | DSN | DSZ | DSC |



# 8-bit LGT8XM

RISC Microcontroller with In-System Programmable FLASH Memory

# LGT8F88P

# LGT8F168P

# LGT8F328P

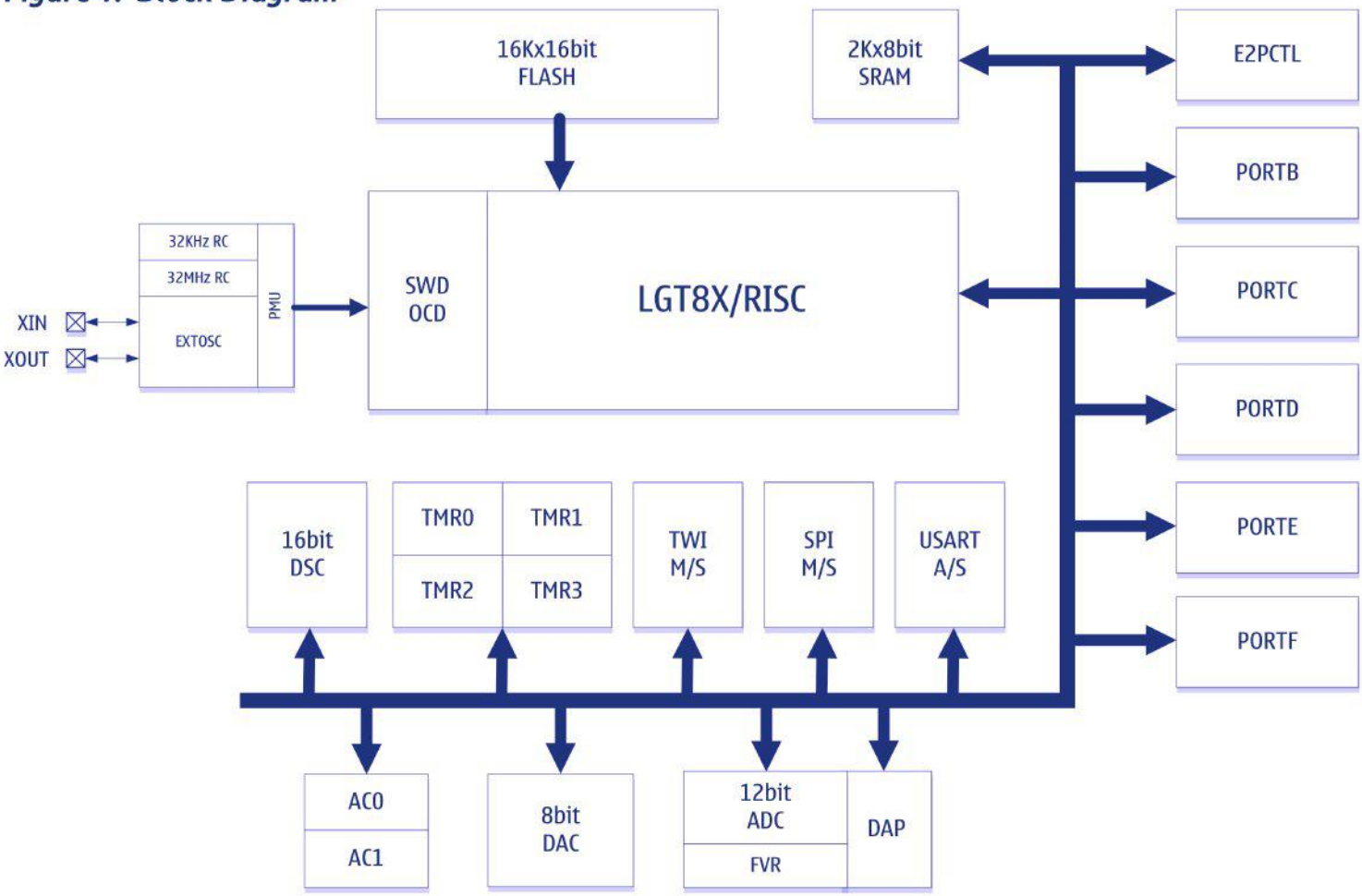
Data book

Version 1.0.4 J

Note: This document has been created for personal use. It is a combination of sources and information. This is not official documentation and is completely unaffiliated with any distribution company or manufacturer. This document is shared freely along with it's source on github. This document is provided as-is in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. In no event will the authors and/or contributors be held liable for any damages arising from the use of this document.

# SECTION 1
# LGT8XM MCU OVERVIEW

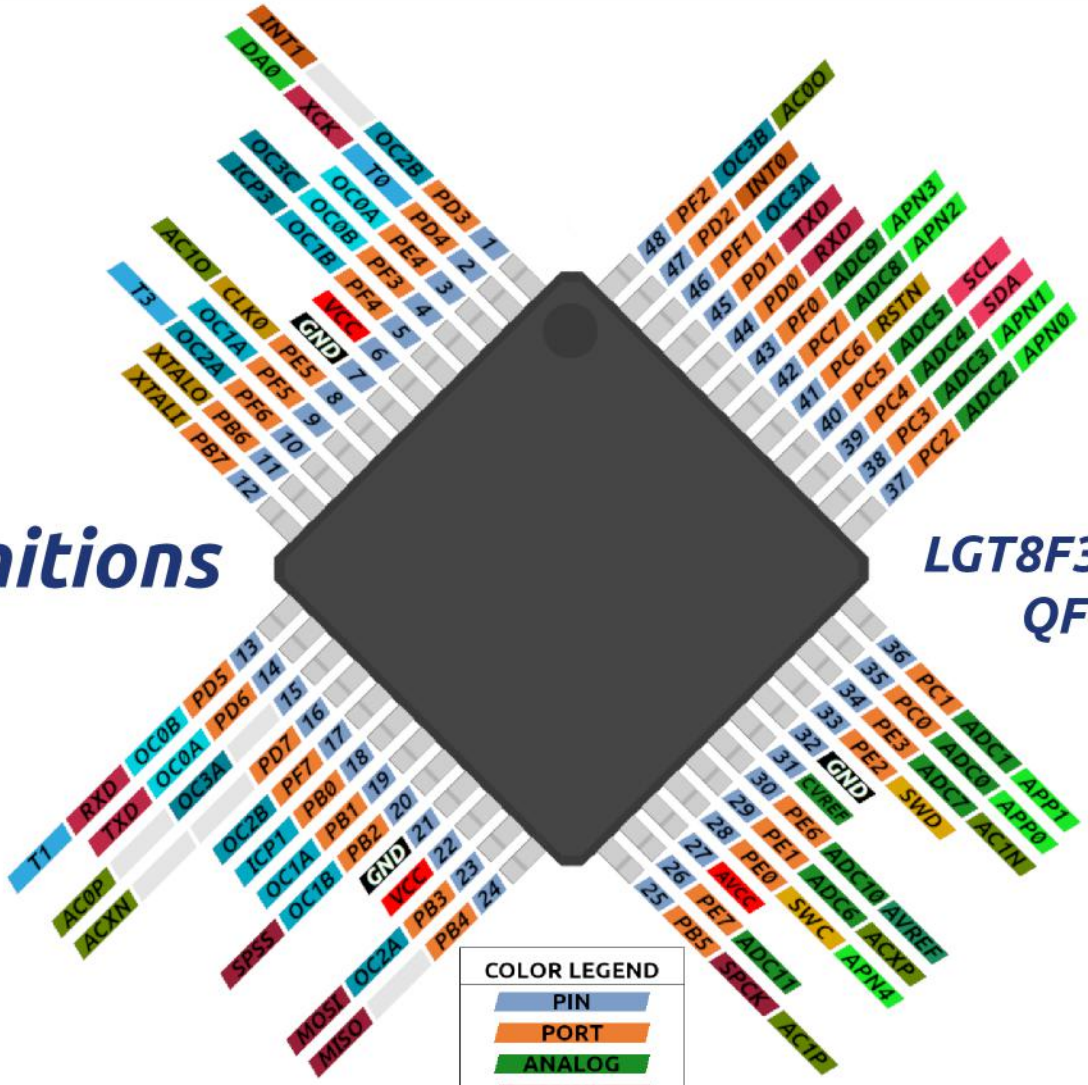
## Figure 1. Block Diagram

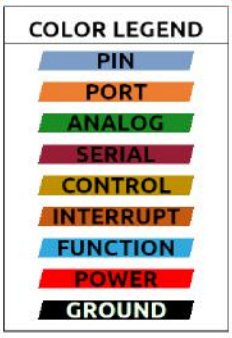



| NAME | DESCRIPTION |
|---|---|
| SWD | Debug Module and ISP Features |
| LGT8X | 8 Bit CPU |
| E2PCTL | MemCon FLASH Memory Controller |
| PMU | Power Management Module |
| PORTB/C/D/E/F | General-Purpose Input and Output Ports |
| DSC | 16 Bit Arithmetic Acceleration Unit |
| ADC | 8 Channel 12 Bit ADC with Programmable Gain |
| DAP | Differential Amplifier |
| IVREF | 1.024V / 2.048V / 4.096V Internal Reference |
| ACO/1 | Analog Comparator |
| TMRO/ 1/2/3 | 8/16 Bit Timer / Counter, PWM Controller |
| WDT | Watchdog Reset Module |
| SPI (M/S) | SPI Controller (Master or Slave) |
| TWI (M/S) | Two-Wire Controller (Master or Slave) |
| USART | Synchronous / Asynchronous Serial Transceiver |
| DAC | 8 Bit DAC |

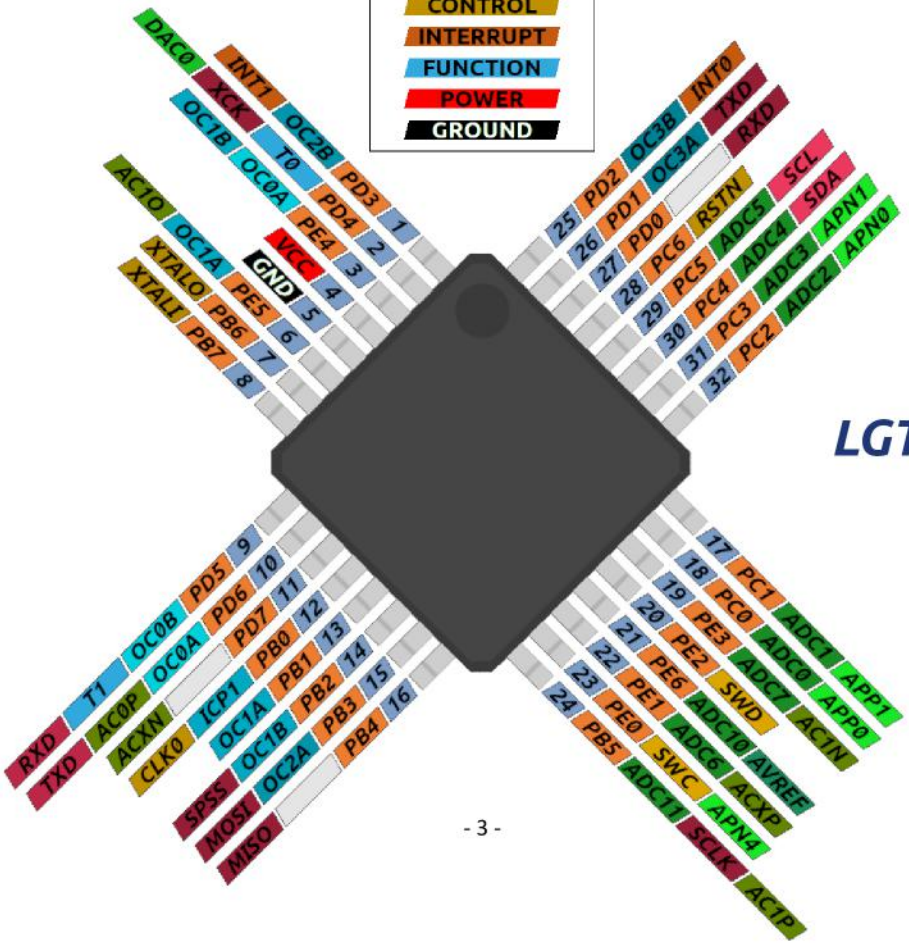
# Pin Definitions

## LGT8F328P QFP48L




## LGT8F328P QFP32L



# Pin Definitions (Cont.)

## LGT8F328P

## SSOP20

| COLOR LEGEND |
| --- |
| PIN |
| PORT |
| ANALOG |
| SERIAL |
| CONTROL |
| INTERRUPT |
| FUNCTION |
| POWER |
| GROUND |

# Pin Description

| QFP48 | QFP32 | SSOP20 | |
|---|---|---|---|
| 01 | 01 | 03 | PD3/INT1/OC2B*<br>PD3: I/O Pin D3<br>INT1: External Interrupt 1 Input<br>OC2B: Timer/Counter 2 Output Compare Match Output B |
| 02 | 02 | | PD4/DAO/T0/XCK<br>PD4: I/O Pin D4<br>DAO: Digital to Analog Converter DAC<br>T0: Timer0 Timer/Counter 0<br>XCK: USART USART external clock |
| 03 | 03 | - | PE4/OC0A*<br>PE4: I/O Pin E4<br>OC0A: Timer/Counter 0 Output Compare Match Output A |
| 04 | - | - | PF3/OC3C/OC0B*<br>PF3: I/O Pin F3<br>OC3C: Timer/Counter 3 Output Compare Match Output C<br>OC0B: Timer/Counter 0 Output Compare Match Output B |
| 05 | 03 | 03 | PF4/OC1B*/ICP3<br>PF4: I/O Pin F4<br>OC1B: Timer/Counter 1 Output Compare Match Output B<br>ICP3: Timer 3 Capture Input |
| 06 | 04 | 04 | VCC |
| 07 | 05 | 05 | GND |
| 08 | 06 | - | PE5/AC1O/CLKO*<br>PE5: I/O Pin E5<br>C1O: Comparator AC1 Output<br>CLKO: System Clock Output |
| 09 | | 06 | PF5/OC1A*<br>PF5: I/O Pin F5<br>OC1A: Timer/Counter 1 Output Compare Match Output A |
| 10 | - | - | PF6/T3/OC2A*<br>PF6: I/O Pin F6<br>T3: Timer 3 External clock input<br>OC2A: Timer/Counter 2 Output Compare Match Output A |
| 11 | 07 | 06 | PB6/XTALO<br>PB6: I/O Pin B6<br>XTALO: Crystal Oscillator Output |

| QFP48 | QFP32 | SSOP20 | |
|---|---|---|---|
| 12 | 08 | 07 | PB7/XTALI<br>PB7: I/O Pin B7<br>XTALI: Crystal Oscillator Input |
| 13 | 09 | 08 | PD5/RXD*/T1/OC0B<br>PD5: I/O Pin D5<br>RXD: USART Receive Data<br>T1: Timer 1 External Clock Input<br>OC0B: Timer/Counter 0 Output Compare Match B |
| 14 | 10 | 09 | PD6/TXD*/OC0A<br>PD6: I/O Pin D6<br>TXD: USART Transmit Data<br>OC0A: Timer/Counter 0 Output Compare Match A |
| 15 | | | AC0P/OC3A<br>AC0P: Analog Comparator 0 Positive Input<br>OC3A: Timer/Counter 3 Output Compare Match A |
| 16 | 11 | 10 | PD7/ACXN<br>PD7: I/O Pin D7<br>ACXN: Analog Comparator 0/1 Inverting Input |
| 17 | - | | PF7/OC2B<br>PF7: I/O Pin F7<br>OC2B: Timer/Counter 2 Output Compare Match B |
| 18 | 12 | 11 | PB0/ICP1<br>PB0: I/O Pin B0<br>ICP1: Timer 1 Capture Input |
| 19 | 13 | | PB1/OC1A<br>PB1: I/O Pin B1<br>OC1A: Timer/Counter 1 Output Compare Match A |
| 20 | 14 | 12 | PB2/OC1B/SPSS<br>PB2: I/O Pin B2<br>OC1B: Timer/Counter 1 Output Compare Match B<br>SPSS: SPI Slave Select |
| 21 | - | - | GND |
| 22 | - | - | VCC |
| 23 | 15 | 12 | PB3/MOSI/OC2A<br>PB3: I/O Pin B3<br>MOSI: SPI Master Output Slave Input<br>OC2A: Timer/Counter 2 Output Compare Match A |
| 24 | 16 | 13 | PB4/MISO<br>PB4: I/O Pin B4<br>MISO: SPI Master Input Slave Output |
| 25 | 17 | 14 | PB5/SPCK/AC1P<br>PB5: I/O Pin B5<br>SPCK: SPI Clock<br>AC1P: Analog Comparator 1 Noninverting Input |
| QFP48 | QFP32 | SSOP20 | |

| QFP48 | QFP32 | SSOP20 | |
|---|---|---|---|
| 26 | - | - | PE7/ADC11<br>PE7: I/O Pin E7<br>ADC11: ADC Input Channel 11 |
| 27 | - | - | AVCC: Internal Analog Circuit Positive Power Supply |
| 28 | 18 | 15 | PE0/SWC/APN4<br>PE0: I/O Pin E0<br>SWC: SWD Debug Interface Clock<br>APN4: Differential Amplifier Inverting Input Channel 4 |
| 29 | 19 | | PE1/ADC6/ACXP<br>PE1: I/O Pin E1<br>ADC6: ADC Input Channel 6<br>ACXP: Analog Comparator 0/1 Noninverting Input |
| 30 | 20 | 16 | PE6/ADC10/AVREF<br>PE6: I/O Pin E6<br>ADC10: ADC Input Channel 10<br>AVREF: ADC External Reference Voltage Input |
| 31 | - | - | CVREF: ADC Reference Voltage External Filter Capacitor (0.1uF) |
| 32 | - | - | AGND: Internal Analog Circuit Power Supply Ground |
| 33 | 21 | 16 | PE2/SWD<br>PE2: I/O Pin E2<br>SWD: SWD Debug Interface Data |
| 34 | 22 | | PE3/ADC7/AC1N<br>PE3: I/O Pin E3<br>ADC7: ADC Input Channel 7<br>AC1N: Analog Comparator 1 Inverting Input |
| 35 | 23 | 17 | PC0/ADC0/APP0<br>PC0: I/O Pin C0<br>ADC0: ADC Input Channel 0<br>APP0: Differential Amplifier Channel 0 Positive Input |
| 36 | 24 | 18 | PC1/ADC1/APP1<br>PC1: I/O Pin C1<br>ADC1: ADC Input Channel 1<br>APP1: Differential Amplifier Channel 1 Positive Input |
| 37 | 25 | - | PC2/ADC2/APN0<br>PC2: I/O Pin C2<br>ADC2: ADC Input Channel 2<br>APN0: Differential Amplifier Channel 0 Inverting Input |
| 38 | 26 | - | PC3/ADC3/APN1<br>PC3: I/O Pin C3<br>ADC3: ADC Input Channel 3<br>APN1: Differential Amplifier Channel 1 Inverting Input |
| QFP48 | QFP32 | SSOP20 | |

| QFP48 | QFP32 | SSOP20 | |
|---|---|---|---|
| 39 | 27 | 19 | PC4/ADC4/SDA<br>PC4: I/O Pin C4<br>ADC4: ADC Input Channel 4<br>SDA: I2C Data |
| 40 | 28 | 20 | PC5/ADC5/SCL<br>PC5: I/O Pin C5<br>ADC5: ADC Input Channel 5<br>SCL: I2C Clock |
| 41 | 29 | 1 | PC6/RESETN<br>PC6: I/O Pin C6<br>RESETN: External Reset Input |
| 42 | - | - | PC7/ADC8/APN2<br>PC7: I/O Pin C7<br>ADC8: ADC Input Channel 8<br>APN2: Differential Amplifier Channel 2 Inverting Input |
| 43 | - | - | PF0/ADC9/APN3<br>PF0: I/O Pin F0<br>ADC9: ADC Input Channel 9<br>APN3: Differential Amplifier Channel 3 Inverting Input |
| 44 | 30 | - | PD0/RXD<br>PD0: I/O Pin D0<br>RXD: USART Receive |
| 45 | 31 | - | PD1/TXD<br>PD1: I/O Pin D1<br>TXD: USART Transmit |
| 46 | | 1 | PF1/OC3A<br>PF1: I/O Pin F1<br>OC3A: Timer/Counter 3 Output Compare Match A |
| 47 | 32 | 2 | PD2/INT0/AC0O<br>PD2: I/O Pin D2<br>INT0: External Interupt 0 Input<br>AC0O: Analog Comparator 0 Output |
| 48 | | | PF2/OC3B<br>PF2: I/O Pin F2<br>OC3B: Timer/Counter 3 Output Compare Match B |
| QFP48 | QFP32 | SSOP20 | |

# SECTION 2
# LGT8XM ARCHITECTURE

- Low Power Design
- High Efficency RISC Archatecture
- 16 Bit LD/ST Extension (Dedicated uDSU)
- 130 Instructions, Over 80% execute in a Single Cycle
- Embedded In-Circuit Debugger (OCD)

## Overview

This section discusses the LGT8XM CPU architecture. The main function of the CPU is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

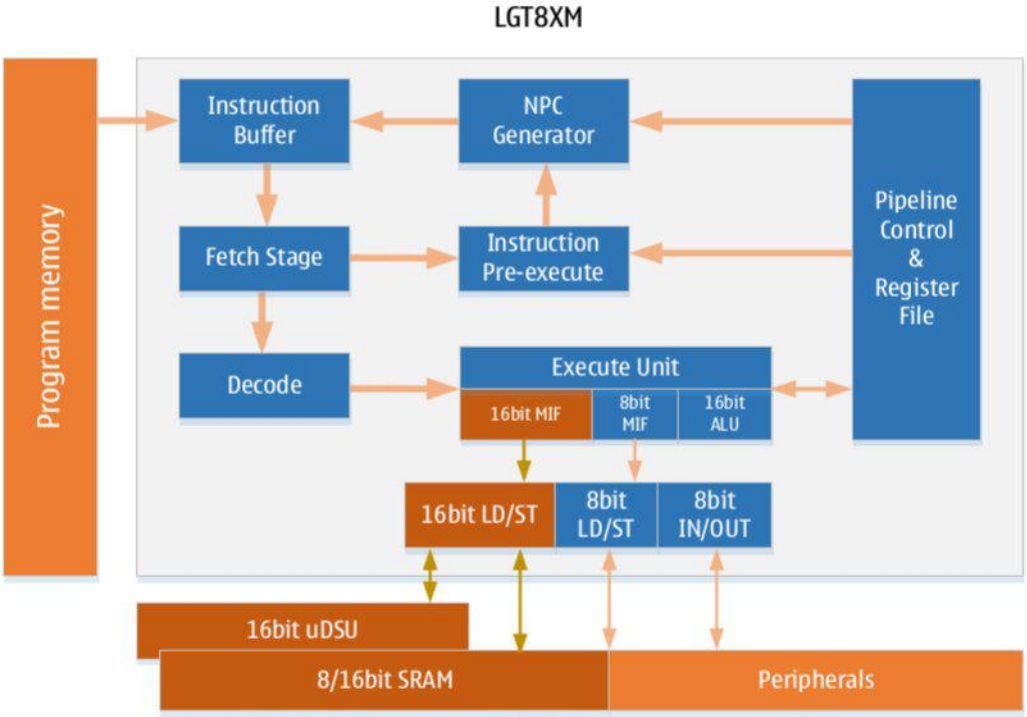



***Figure 2. Block Diagram of the LGT8XM Architecture***

The LGT8XM uses a Harvard like architecture – with separate buses for program and data. Instructions in program memory are executed with single level pipelining. While one instruction is being executed, the next instruction is prefetched from program memory. This concept enables instructions to be executed every clock cycle. The program memory is reprogrammable in circuit. The Register File contains 32 x 8-bit General Purpose Registers with single clock cycle access times. This allows single-cycle operation of the Arithmetic Logic Unit (ALU). In a typical ALU operation, two operands are fetched from the Register File, the operation is executed, and the result is pushed back in the Register File – in one CPU system clock cycle.

Six of the 32 extra working registers can be used to form a combination of three 16-bit registers. These 16-bit registers can be used to access external memory and FLASH programs by indirectly addressing the address pointer. The LGT8XM supports single-cycle 16-bit arithmetic, greatly improving the efficiency of indirect addressing. The three special 16-bit registers in the LGT8XM core are named X,Y, Z registers, which are described in more detail later.

The ALU supports arithmetic logic operations between registers and between a constant and a register. The operation of a single register can also be performed in the ALU. After the ALU operation is completed, the effect of the operation on the state of the system is updated to the status register (SREG).

Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space. Most LGT8XM instructions are 16 bits. Each program address space corresponds to a 16-bit or 32-bit LGT8XM instruction.

After the system responds to an interrupt or a subroutine call, the return address Program Counter (PC) is stored on the stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All applications that support interrupt or subroutine calls must first initialize the Stack Pointer Register (SP), which can be accessed through the IO space. Data SRAM can be accessed in five different addressing modes.

The internal memory of the LGT8XM is linearly mapped to a uniform address space. Please refer to the the storage memory chapter for further details.

The LGT8XM includes a flexible interrupt controller. It can be controlled by a global interrupt enable bit in the Status Register. Every interrupt has a separate interrupt vector. The priority of the interrupt has a corresponding relationship with the interrupt vector address. The smaller the interrupt address, the higher the priority of the interrupt.

The I/O space contains 64 register spaces that can be directly addressed by IN/OUT instructions. These registers are used for kernel control as well as control functions for status registers, SPI and other I/O peripherals. This space can be accessed directly or by their address mapped to the data memory space (0x20 - 0x5F). In addition, the LGT8FX8P also includes extended I/O space, which is mapped to data memory space 0x60-0xFF, which can only be accessed using ST/STS/STD and LD/LDS/LDD instructions.

To enhance the computing power of the LGT8XM core, a 16-bit LD/ST extension has been added to the instruction set. This 16-bit LD/ST expansion works with the 16/32-bit Dimensional Operation Acceleration Unit (uDSU) For efficient 16-bit data operations. At the same time, the kernel also increases the 16-bit access capability to the RAM space. The 16-bit LD/ST extension can pass 16 bits of data between the uDSU, RAM, and working registers. Please refer to the "Digital Operation Accelerator" section for details.

### *ALU - Arithmetic Logic Unit*

The LGT8XM internally contains a 16-bit arithmetic logic unit that can perform 16 bit arithmetic operations on data in one cycle. The highly efficient ALU is connected to 32 general purpose working registers. The ALU has the ability to perform two arithmetic operations between registers or registers and immediate data in one cycle. There are three types of ALU operations: arithmetic, logic, and bit operations. The ALU also includes a single-cycle hardware multiplier that implements direct signed or
unsigned operations on two 8-bit registers in a single cycle. Please refer to the detailed description in the instruction set section.

### *SREG - System Status Register*

The status register mainly stores the result information generated by the execution of the most recent ALU operation. This information is used to control the program execution flow. The status register is updated after the ALU operation has completely ended. This usually eliminates the need for separate compare instructions, resulting in a more compact and efficient code implementation. The value of the status register is not automatically saved and restored in response to an interrupt and exit from the interrupt, which requires software to implement.

## Register Definition - SREG

| SREG System Status Register | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Address: **0X3F (0X5F)** | | | | Defaults: **0X00** | | | | |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | I | T | H | S | V | N | Z | C |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |

| Bit Definition | | |
|---|---|---|
| [0] | C | Carry flag, indicating that the arithmetic or logic operation caused the carry. * |
| [1] | Z | A zero flag indicating that the result of an arithmetic or logical operation is zero. * |
| [2] | N | A negative flag indicates that an arithmetic or logic operation has produced a negative number. * |
| [3] | V | The overflow flag indicates that the result of the two's complement operation has overflowed. * |
| [4] | S | Sign bit, equivalent to the XOR operation result of N and V * |
| [5] | H | Semi-carry flag, useful in BCD operations, indicating the semi-carry generated by byte operations* |
| [6] | T | For temporary bits, bit copy (BLD) and bit store (BST) instructions, the T bit is used as a temporary memory bit to temporarily store the value of a bit in the general purpose register. * |
| [7] | I | The global interrupt enable bit must be set to 1 to enable the CPU to respond to interrupt events. Different interrupt sources are controlled by independent control bits. The global interrupt enable bit is the last barrier that controls the interrupt signal into the CPU. The I bit is automatically cleared in hardware when the CPU responds to the interrupt vector and is automatically set after the interrupt return instruction (RETI) is executed. The I bit can also be changed using the SEI and CLI instructions. * |

** Please refer to the instruction description for details.*

## General Purpose Register

The General Purpose Registers are optimized for the LGT8XM instruction set. To achieve the efficiency and flexibility required for core execution, the LGT8XM's internal working registers support several access modes:

- Simultaneous 8-bit read and 8-bit write
- Two 8-bit reads simultaneously with 8-bit write
- Two 8-bit reads simultaneously with one 16-bit write
- One 16-bit read simultaneously with one 16-bit write

## General Purpose Register (Cont.)

| | 7 0 | Address | |
|---|---|---|---|
| GENERAL WORKING REGISTER | R0 | 0x00 | |
| | R1 | 0x01 | |
| | R2 | 0x02 | |
| | ... | | |
| | R13 | 0x0D | |
| | R14 | 0x0E | |
| | R15 | 0x0F | |
| | R16 | 0x10 | |
| | R17 | 0x11 | |
| | ... | | |
| | R26 | 0x1A | X Register Low Byte |
| | R27 | 0x1B | X Register High Byte |
| | R28 | 0x1C | Y Register Low Byte |
| | R29 | 0x1D | Y Register High Byte |
| | R30 | 0x1E | Z Register Low Byte |
| | R31 | 0x1F | Z Register High Byte |

*Figure 3. General Purpose Register*

Most of the instructions operating on the Register File have direct access to all registers, and most of them are single cycle instructions. As shown in the figure above, each register is also assigned a data memory address, mapping them directly into the first 32 locations of the user Data Space. Although not being physically implemented as SRAM locations, this memory organization provides great flexibility in access of the registers, as the X-, Y- and Z-Pointer registers can be set to index any register in the file.

## X / Y / Z Register

Registers R26...R31 can be combined in pairs to form three 16-bit registers. These three 16-bit registers are mainly used as address pointers for indirect addr--essing access. The X/Y/Z registers are structured as follows:

| | 15 | XH | | XL | 0 |
|---|---|---|---|---|---|
| **X** REGISTER | 7 | 0 | 7 | | 0 |
| | R27 (0x1B) | | R26 (0x1A) | | |
| | 15 | YH | | YL | 0 |
| **Y** REGISTER | 7 | 0 | 7 | | 0 |
| | R29 (0x1D) | | R28 (0x1C) | | |
| | 15 | ZH | | ZL | 0 |
| **Z** REGISTER | 7 | 0 | 7 | | 0 |
| | R31 (0x1F) | | R30 (0x1E) | | |

These registers are used as fixed offset, auto-increment, and auto-decrement address pointers in different addressing modes. See the Inst--ruction Description section for details.

## Stack Pointer

The stack is used to store temporary data, local variables, and the return address of interrupts and subroutine calls. Note that the Stack is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack. The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer.

The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM. Please refer to the system data memory section for the address of the SRAM mapped in the system data store.

### Table 2. Stack Pointer Instructions

| Instruction | Stack Pointer | Description |
|---|---|---|
| PUSH | Decremented 1 | Data is pushed onto the stack |
| CALL<br>ICALL<br>RCALL | Decremented 2 | Return address is pushed onto the stack with a subroutine call or interrupt |
| POP | Incremented 1 | Data is popped from the stack |
| RET<br>RETI | Incremented 2 | Return address is popped from the stack with return from subroutine or return from interrupt |

The Stack Pointer consists of two 8-bit registers in I/O Memory. The actual length of the Stack Pointer is related to the system implementation.

### Register Definition - SPH & SPL

| SPH/SPL Stack Pointer Register | | |
|---|---|---|
| SPH: 0x3E (0x5E)<br>SPL: 0x3D (0x5D) | | Default: RAMEND |
| SP | SP[15:0] | |
| R/W | R/W | |
| Bit Definition | | |
| [7:0] | SPL | Stack pointer low 8 bits |
| [15:8] | SPH | Stack pointer high 8 bits |

## Instruction Execution Timing

The LGT8XM is driven by the CPU clock (clkCPU), which comes directly from the clock source selection circuit.

Figure 4 on page 14 shows the parallel instruction fetch and execution timing. This is based on Harvard architecture with the fast access register file concept. This is the basic pipelining concept that enables the core to achieve 1 MIPS/MHz execution efficiency.

## Instruction Execution Timing (Cont.)

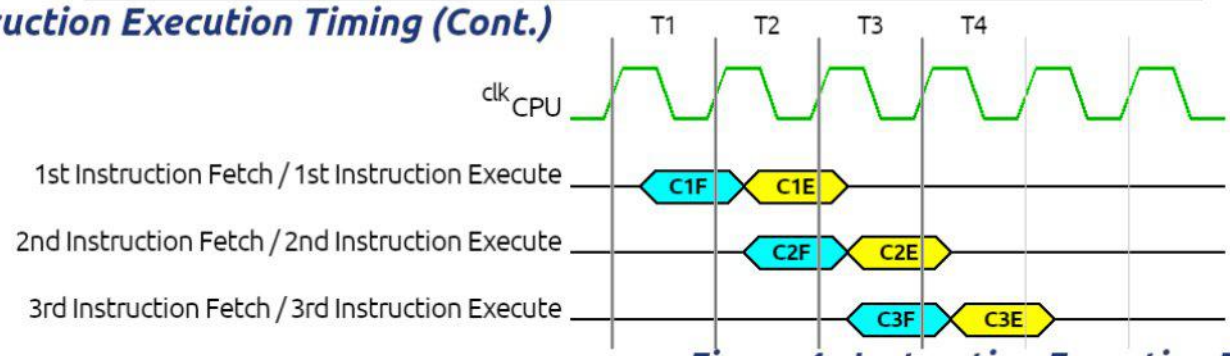



**Figure 4. Instruction Execution Timing**

As can be seen in the figure above, the second instruction will be fetched during the execution of the first instruction. When the second instruction enters execution, the third instruction is fetched. This eliminates the need for an additional CPU clock cycle to fetch the instruction before that instruction can be executed.

The following figure shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

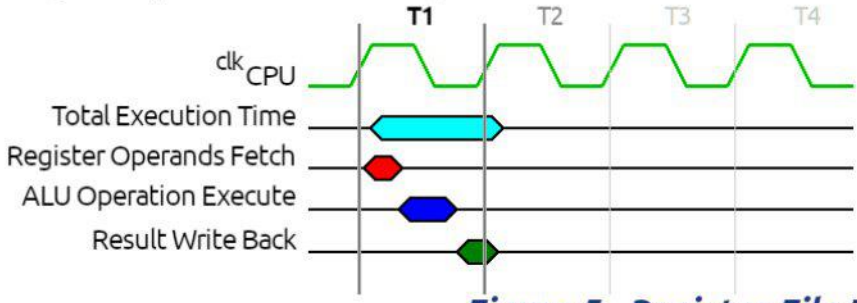



**Figure 5. Register File Timing**

## Reset and Interrupt Handling

The following figure shows the internal timing concept for the Register File. In a single clock cycle an ALU operation using two register operands is executed, and the result is stored back to the destination register.

The LGT8XM supports multiple interrupt sources. These interrupts and the separate Reset Vector each have a separate program vector in the program memory space. In general, all interrupts are assigned separate control bits. When this control bit is set and the Global Interrupt Enable bit of the Status Register is enabled, the CPU can respond to this interrupt.

The lowest addresses in the program memory space are reserved by default as the Reset and Interrupt Vector area. For a complete list of interrupts supported by the LGT8FX8P, please refer to the Interrupts section. This list also determines the priority of different interrupts. The lower the address, the higher the corresponding interrupt priority. RESET has the highest priority, then INT0 – External Interrupt Request 0. The start address of the interrupt vector table (except the RESET vector) can be moved to the beginning of any 256-byte alignment, via the MCU Control Register's IVSEL bit in (MCUCR) and the IVBASE vector base address register implementation.

When the CPU responds to an interrupt, the Global Interrupt Enable flag (I) is automatically cleared by hardware effectively disabling interrupts. The user can nest interrupts by enabling the I-bit in software. If this is done in software, any subsequent interrupts will interrupt the current interrupt service routine. The Global Interrupt Enable I-bit is automatically set after the execution of the interrupt return instruction (RETI), so that it can respond normally to subsequent interrupts.

There are basically two types of interrupts. The first type is triggered by an event that sets the interrupt flag. For this kind of interrupt, the Program Counter (PC) value is directly replaced with the actual interrupt vector address. This enables the PC to execute the interrupt handling routine. Hardware automatically clears the interrupt flag bit. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If another interrupt condition occurs while interrupts are disabled the new interrupt flag will be remembered until the interrupt enable (I-bit) is set. Then this new interrupt will be serviced. This flag can also be cleared by software to bypass the new/waiting interrupt. Similarly, if the global interrupt enable bit (SERG.I) is cleared when an interrupt occurs, the corresponding interrupt flag bit will also be set to record the interrupt event, wait until the global interrupt enable bit is set, and will then be executed by order of priority.

## Reset and Interupt Handling (Cont.)

The second type of interrupts will continue to respond as long as the interrupt condition persists. This type of interrupt does not require an interrupt flag bit. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the LGT8XM CPU exits from the interrupt service routine, the execution flow is returned to the main program. After executing one more instruction in the main program, it can respond to other pending interrupt requests.

It should be noted that the System Status Register (SREG) is not automatically saved after entering or returning from an interrupt routine. Managing the SREG must be handled in software.

The CLI command clears the Global Interrupt Flag (I) in SREG (Status Register). When interrupts are disabled using the CLI instruction, interrupts will be immediately disabled. Interrupts that occur after the CLI instruction will not get a response, nor will interrupts that occur simultaneously with the CLI instruction.

## Interrupt Response Time

Any LGT8XM enabled interrupt will respond within 4 CPU clock cycles. After these CPU clock cycles, the interrupt enters the actual interrupt service routine. In these four clocks, the Program Counter value from before the interrupt is Pushed onto the Stack, and the system execution flow Jumps to the interrupt vector address and corresponding interrupt service routine. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If the interrupt occurs while the system is in sleep mode (SLEEP), the interrupt response requires an additional 4 clock cycles. This increase comes in addition to the start-up time from the selected sleep mode. For a detailed description of the sleep mode, please refer to the relevant chapter on power management.

It takes 2 clock cycles to return from the interrupt service routine. During these 2 clock cycles, the Program Counter (two bytes) is Popped back from the Stack, the Stack Pointer is incremented by two, and the I-bit in SREG is automatically set.

# SECTION 3
# MEMORY

## Overview

The LGT8XM architecture has two main memory spaces, Data and Program Memory space. The LGT8FX8P contains internal FLASH that can emulate EEPROM data storage. There is also a special space for storing system configuration information and the chip's global ID number (GUID).

The LGT8FX8P series includes three different models: LGT8F88P/168P/328P. These three models are fully compatible. The differences are the size of FLASH and SRAM available. The following table describes the LGT8FX8P memory configuration options by model:

### Table 3. LGT8FX8P Memory Configurations

| DEVICE | FLASH | SRAM | EEPROM | INTERRUPTS |
|---|---|---|---|---|
| LGT8F88P | 8KB | 1KB | 2KB | 1 inst. words |
| LGT8F168P | 16KB | 1KB | 4KB | 2 inst. words |
| LGT8F328P | 32KB | 2KB | Can be configured as 0K/1K/2K/4K/8K (shared with FLASH) | 2 inst. words |

The emulated E2PROM Memory of the LGT8F328P is not stored in a separate location from FLASH Memory. The emulated E2PROM system is integrated into FLASH Memory. The user selects the appropriate FLASH partition configuration according to their requirements.

Due to the user configured emulated E2PROM, the system requires twice the program FLASH space to simulate an E2PROM memory space. For example, if the user configures a 1KB E2PROM, 2KB of FLASH memory will be reserved, leaving 30KB for program memory.

### Table 4. LGT8F328P FLASH & E2PROM Configuration

| DEVICE | FLASH | E2PROM |
|---|---|---|
| LGT8F328P | 32KB | 0KB |
| | 30KB | 1KB |
| | 28KB | 2KB |
| | 24KB | 4KB |
| | 16KB | 8KB |

## FLASH Program Memory

The LGT8FX8P series of microcontrollers each include 8K/16K/32K bytes of on-chip programmable FLASH memory. The FLASH memory is designed for at least 100,000 erase cycles. The LGT8FX8P integrates a FLASH interface controller that includes an in circuit programming feature. For specific implementation details, please see the information on page 23.

Program Memory can be directly read via the LPM instruction. This feature enables application-dependent constant lookup tables. The FLASH program memory is address mapped as a single monolithic block along with the system data storage memory (SRAM). FLASH Memory can also be accessed using the LD/LDD/LDS instructions. The program memory is mapped to the address range directly following the 16-bit data memory (0x4000). This is illustrated in Figure 6. on page 17.

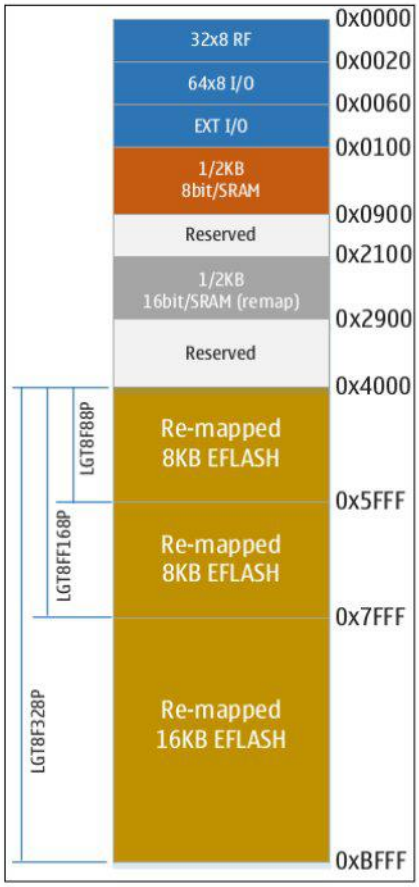
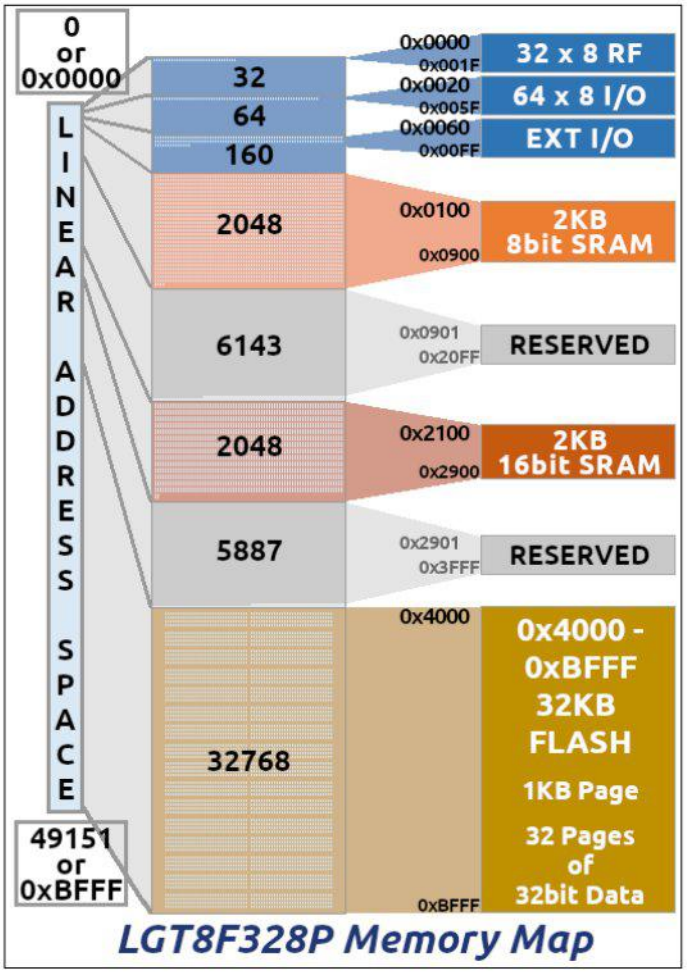



*Figure 6. LGT8FXP Memory Map*




*LGT8F328P Memory Map*

## SRAM Data Memory

The LGT8FX8P series of microcontrollers have 3 sections of address spaces in the first 256 bytes of RAM data memory. The first block section is the Register File. This is the 32 byte Special Function Register, and is addressed from 0x00 to 0x1F. The next 64 addresses, 0x20 to 0x5F, are the Direct I/O Register. The third section is the Extended I/O Register. This occupies from 0x60 to 0xFF. In decimal, this breaks down to addresses 0 to 31, 32 through 95, and 96 through 255.

The 32x8 Register File has already been described in the CPU Architecture section beginning on page 11. The 64 Direct I/O Registers can be accessed with the faster IN/OUT instructions. The next 160 Extended I/O Registers from 0x60-0xFF are mapped to the data storage space. These registers are only accessible with the slower data addressing instructions such as ST/STS/STD and LD/LDS/LDD.

The first 256 addresses are directly followed by a maximum of 2K bytes of data SRAM. The portion of the space from 0x4000 to the end is the FLASH Program Memory.

The 1KB (LGT8F88P/LGT8F168P) or 2KB (LGT8F328P) of SRAM Data Memory is mapped in two separate locations. The first space from 0x0100 to 0x0900 is read and written by the CPU in 8-bit bytes. The second SRAM space starting from 0x2100 to 0x2900 is a 16-bit wide access space. The system RAM is mapped to the high-order address starting from 0x2100 and is designed to work with the uDSU module for more efficient 16-bit data storage. When programming, the normal 8-bit address is offset by 0x2000 to switch to 16-bit access mode.

## SRAM Data Memory (continued)

The five different addressing modes for the data memory are: Direct, Indirect, Indirect with Displacement, Indirect with Pre-decrement, and Indirect with Post-increment. In the Register File, registers R26 to R31 are the indirect addressing pointer registers. Indirect access can address the entire data storage space.

The Indirect with Displacement addressing mode incorporates a 6 bit address as part of the the instruction word. These six bits are added to the the address given by the Y or Z Registers to access up to 63 address locations from the Y or Z Register base address.

When using the Indirect with Post-Increment or Pre-Decrement Adressing Modes, the X, Y, and Z registers are incremented or decremented. Please refer to the Instruction Set Description section for more details.

The 16-bit register X/Y/Z and its associated automatic addressing mode (increment, decrement) also play a very important role in the 16-bit extended mode. The 16-bit extended mode can use the LD/ST increment/decrement mode to implement auto-increment and decrement addressing with variables. This mode is very effective when performing operations on arrays. For details, please refer to the chapter, "Digital Operational Accelerator (uDSU)".

## General Purpose I/O Registers

The I/O space of the LGT8FX8P has three general-purpose I/O registers GPIOR2/1/0. These three registers can be accessed using the IN/OUT instruction to store user-defined data.

## I/O Memory

All LGT8FX8P I/Os and peripherals are mapped to the I/O space. All I/O locations may be accessed by the LD/LDS/LDD and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range of 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked using the SBIS and SBIC instructions. When using the Direct I/O specific commands IN and OUT, the only the Direct I/O Register addresses 0x00 - 0x3F may be used. When addressing any I/O Registers with the LD and ST data space instructions, 0x20 must be added to these addresses. Peripheral registers allocated in the Extended I/O space (0x60~0xFF) can only be accessed using ST/STS/STD and LD/LDS/LDD instructions.

For more details about the I/O space, please refer to the "Register Summary" section.

For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.

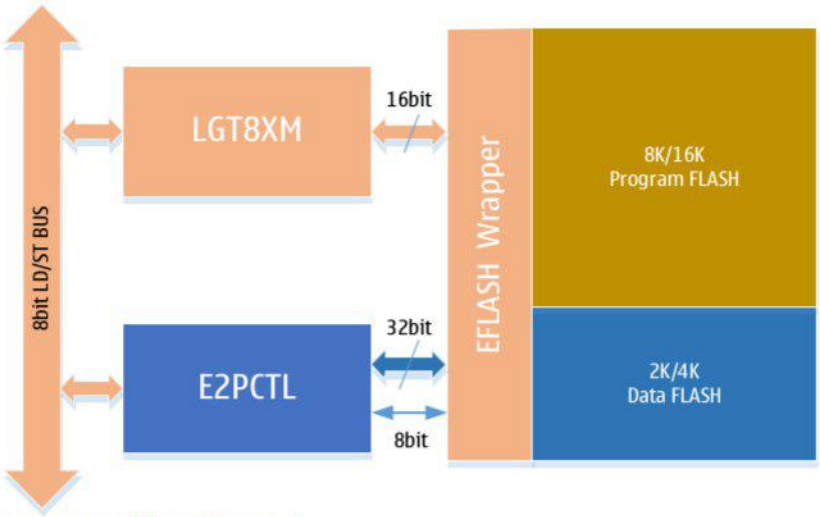
Some of the Status Flags are cleared by writing a logical one to them. Note that, the CBI and SBI instructions will only operate on the specified bit, and can be used on registers containing such Status Flags. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

## MemCon FLASH Controller (E2PCTL)

The LGT8FX8P has a special internal memory access controller, the E2PCTL. The names E2PCTL, Memory Controller, and MemCon are used throughout this document and are synonymous. Th Memory controller has access to the entire FLASH Memory, and handles the emulated E2PROM read and write access within the FLASH. The emulated E2PROM includes a system for memory equalization that might increase the life cycle of FLASH Memory by an order of magnitude, and hopefully more than 100,000 erase cycles.

In addition to managing the emulated E2PROM, the MemCon manages FLASH Program Memory and in circuit programming features for upgrading software inside the microcontroller. FLASH Program Memory supports Page erase only. Each page is 1024 bytes. FLASH Program Memory is 32-bits wide, although the MemCon supports Byte level access.

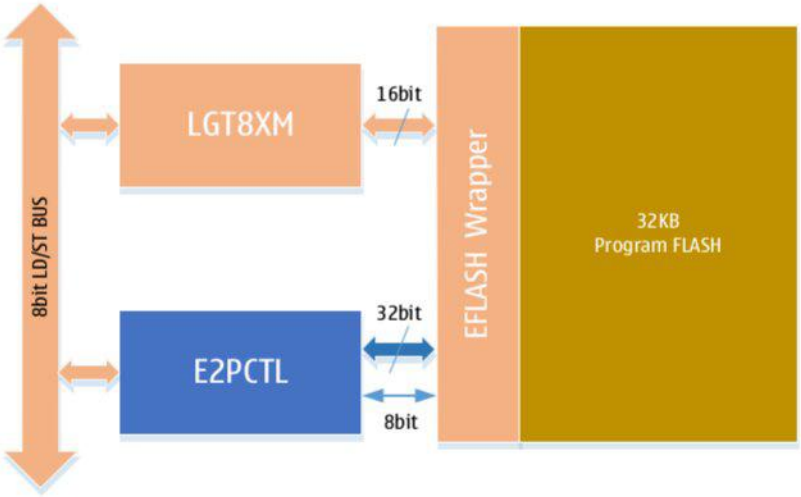
### *Figure 7. LGT8F88D/168D MemCon (E2PCTL) Block Diagram*




### *MemCon FLASH Controller (Cont.)*

When the MemCon (E2PCTL) is in the emulated E2PROM mode, it supports 8-bit and 32-bit read/write widths. It supports 32-bit read/write when accessing FLASH Program Memory. The entire FLASH Memory of the LGT8FX8P is mapped as 32 bits. Therefore it is recommended to use 32-bit access, especially for programming operations. The 32-bit read and write operation is more efficient, and also helps ensure optimal cycle life for the FLASH memory.

### *Figure 8. LGT8F328P MemCon (E2PCTL) Block Diagram*




The entire Program Memory is monolithically integrated inside the LGT8F328P. The LGT8F328P CPU shares the internal 32K bytes of FLASH memory with the emulated E2PROM. Users can partition the 32K bytes into separate Program Memory and emulated E2PROM Memory according to their needs. By configuring the MemCon settings, the partition size of the emulated E2PROM can be adjusted. The emulated E2PROM is implemented using a 1K byte page swap system. For example, to emulate 1K bytes of E2PROM memory, the MemCon requires 2K bytes of FLASH space. In this example a 1K-Byte Page has Current programmed data. In order to erase or write changes an additional 1K-Byte Swap Page is required for implementation and data security. Likewise, to achieve 4K bytes of emulated E2PROM, 8K bytes of FLASH Program Memory are partitioned and reserved. For more information, please refer to the E2PROM Read/Write Access section on page 20.

## MemCon Data Register

The MemCon has an internal 4-byte data buffer (E2PD0~E2PD3). This 4-byte buffer forms the 32-bit data interface for accessing FLASH. This cache buffer has an internal multiplexer the EEDR. It is used to transition between 32-bit Word and 8-bit Byte access modes. When the memory controller is in 8-bit Read/Write Mode, the EEDR controls reading and writing byte data to the buffer. The MemCon's lower data register EEARL[1:0] contains the buffer address information in the two least significant bits. These two bits are used to load the 8-bit Byte data into the correct data buffer location. The starting address is based on the current FLASH target address. After the MemCon fills in the buffer's other three bytes of data, it updates all 32-bits in FLASH.

When the MemCon is in 32-bit Read/Write Mode, the EEDR (MUX) register can still be used as a data interface. The internal data cache is still addressed by EEARL[1:0] for access to the complete 32-bit word. In addition, you can use the data cache to directly access the registers of the IO space (E0~3).

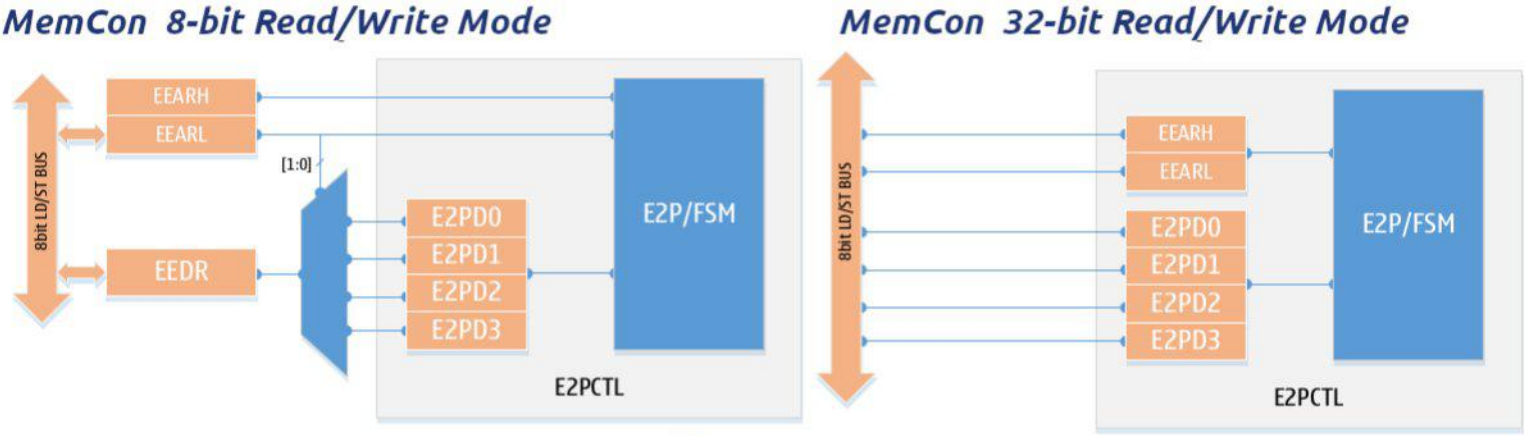



*Figure 9.*

Byte mode is used for backward compatibility with the byte read and write mode of the LGT8FX8D. The LGT8FX8P's built-in FLASH is 32-bits in width. Using 32-bit read/write mode will improve read/write efficiency and is recommended.

## E2PROM Read/Write Access

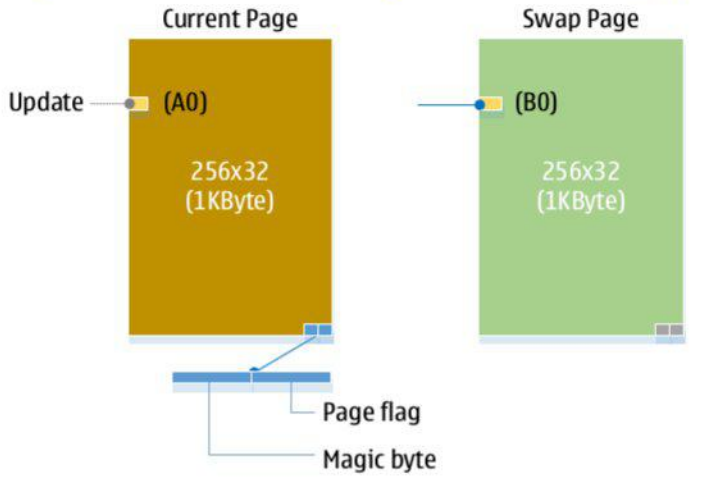
As mentioned previously FLASH Memory must be erased before writing, and the erase operation is performed in page size blocks. Each Page of Flash Memory inside the LGT8FX8P is 1024 bytes. Updating a Byte of data within a single Page first requires erasing the data of the entire Page. Then the target address data can be updated while also restoring the other unchanged bytes from the page. The whole operation is not only time consuming, but also brings the risk of data loss if power is disconnected during the operation.

The MemCon has an integrated page exchange system specificly designed for emulated E2PROM. The page exchange system is designed to mitigate data loss during the page erase operation due to unexpected situations like user initiated system power loss. The page exchange system requires two pages of space in the exchange. While this data protection system comes at a managed memory cost, the system should also increase the service life of the emulated E2PROM.

## E2PROM Read/Write Access (Cont.)

The MemCon always manages each Page of EFLASH separately. When working in the emulated E2PROM mode, the last 2 bytes of each page are reserved for storing information about the status of the page. These two Bytes are the Page Flag and Magic Byte pictued below. Normally the user can not read or write to these last two Bytes. If the user selects an emulated E2PROM partition larger than one Page, the user must account for these two special Bytes in software.

### Figure 10. E2PROM Page Based Exchange




As pictured above, the MemCon requires two 1024-Byte (aka 256-Word) Pages of EFLASH to emulate each Page of E2PROM Memory. One of these two pages is always set as the Current Page, and the other is set as the Swap Page. The MemCon uses the last 2 bytes of each Page to store Page information like the Current and Swap Page status.

When it is time to update the Page, such as the A0 Byte in the illustration above, there are several steps involved. First, the Swap Page is erased. Next, the Current Page's data addresses from before and after the changing data are defined as CP0 and CP1 respectively. After this, the MemCon will copy the data from CP0 on the Current Page into the corresponding addresses of the Swap Page. Once this is complete the Memcon writes the changed data that triggered the entire process (A0 in the illustration). This is written to the Swap Page at the designated corresponding address of the data on the Current Page. The last step is to begin copying CP1 from the Current page to the Swap Page.

At this point in the operation the Current Page is completely unchanged. If a power failure occurs, the old data's integrity is secure. The last step of the emulated E2PROM data operation is triggered when the MemCon controller reaches the very end of CP1 on the Swap Page. This final trigger sets the Swap Page as the Current Page.

## E2PROM Read/Write Access (Cont.)

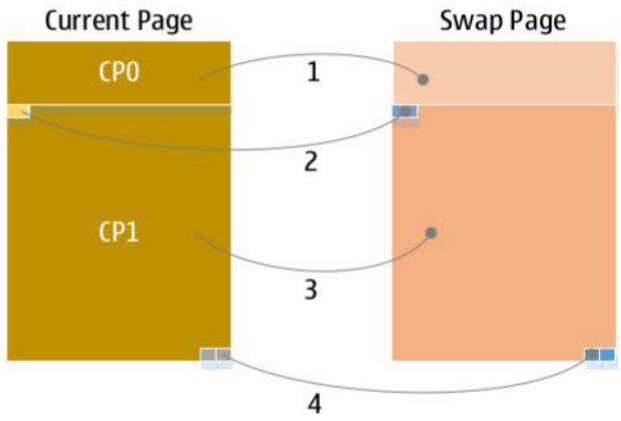


**Figure 11. E2PROM Swap Addressing**

When the emulated E2PROM is configured as greater than 1K, the E2PCTL (MemCon) still operates within the E2PROM partition with the Page as the minimum unit. For example, if the user configures a 2K E2PROM partition, the MemCon requires 4 pages (4K) of space to operate. Two pages are always required for implementing each block of emulated E2PROM memory that the user configures as E2PROM.

It should be noted that the 2K byte E2PROM space configured by the user is not continuous, because the last 2 bytes of each page will be used to save the Page status information.

## E2PROM Continuous Programming Mode

Since the E2PROM update will cause page swapping, the old Current Page will be erased during the swap. Page erasure is not only time consuming, but it also effects the cycle life of the FLASH. Therefore the MemCon includes a Continuous Write Mode. In the Continuous Write Mode, the user can continuously update the emulated E2PROM. In this mode, the page update and exchange operation is only performed at the end of continuous addressing. For applications that need to write several bytes of data, the continuous mode is more efficient.

Continuous Programming Mode is set by enabling the MemCon's SWM bit in the ECCR control register[bit 4]. After this mode is enabled, the subsequent write operations will directly write data to the Swap Page at the corresponding address of the Current Page. In SWM mode, the write operation will not perform the CP0/1 data copy operation. After writing the last byte, the software disables the SWM/continuous mode bit and triggers the final CP0/1 routine which then triggers the page swap.