LGT8FX8P Series – Programming Manual 1.0.4J
Errata Page 13
Jake 2019 Oct 9 11:30am
Discrepancy My edited text ATMEL text Original+Google Translate
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.

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after 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, see Figure 5-3 on page 18.

The stack is used to store temporary data, local variables, and the return address of interrupts and subroutine calls. It is important to note that the stack is not designed to grow from a high address to a low address. The Stack Pointer Register (SP) always points to the top of the stack. The stack pointer points to the physical space where the data SRAM is located, where the stack space necessary for the subroutine or interrupt call is stored. The PUSH instruction will decrement the stack pointer.
The location of the stack in the SRAM must be properly set by the software before the subroutine execution or interrupt enable. In general, the stack pointer is initialized to the highest address of the SRAM. The stack pointer must be set to the high SRAM start address. Please refer to the system data storage section for the address of the SRAM mapped in the system data store.

堆栈用于存储临时数据,局部变量以及中断和子程序调用的返回地址。需要特别注意的是,堆栈别设计为从高地址向低地址生长。堆栈指针寄存器(SP)总是指向堆栈的顶部。堆栈指针指向数据 SRAM 所在的物理空间,这里存放子程序或中断调用必须的堆栈空间。PUSH 指令将会使得堆栈指针递减。
堆栈在 SRAM 中的位置必须在子程序执行或者中断使能之前由软件正确的设置。一般情况下是将堆栈指针初始化指向 SRAM 的最高地址处。堆栈指针必须设置为高位 SRAM 开始地址。SRAM 在系统数据存储映射的地址请参考系统数据存储部分。

This may be a translation error, but the google translation says the opposite of the Atmega DS while mirroring it almost exactly otherwise. I'm going with the Atmega version.

Stack Pointer Instructions

The stack pointer consists of two 8-bit registers allocated in the I/O space. The actual length of the stack pointer is related to the system implementation. In some chip implementations of the LGT8XM architecture, the data space is so small that only SPL can satisfy the addressing needs, in which case the SPH register will not appear.

The AVR Stack Pointer is implemented as two 8-bit registers in the I/O space. The number of bits actually used is implementation dependent. Note that the data space in some implementations of the AVR architecture is so small that only SPL is needed. In this case, the SPH Register will not be present.

The stack pointer consists of two 8-bit registers allocated in the I/O space. The actual length of the stack pointer is related to the system implementation. In some chip implementations of the LGT8XM architecture, the data space is so small that only SPL can satisfy the addressing needs, in which case the SPH register will not appear.

堆栈指针由分配在 I/O 空间的两个 8 位的寄存器构成。堆栈指针的实际长度与系统实现相关。在 LGT8XM 构架的有些芯片实现中,数据空间非常小,以至于仅仅 SPL 就能满足寻址需要,这种情况下,SPH 寄存器将不会出现。

This section describes the general access timing concepts for instruction execution. The LGT8XM is driven by the CPU clock (CLKcpu), which comes directly from the clock source selection circuit of the system.
The following figure shows the parallel instruction fetch and execution timing based on the Harvard architecture and the fast access register file concept. This is the basic pipelining concept that enables the core to achieve 1 MIPS/MHz execution efficiency.

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk CPU , directly generated from the selected clock source for the chip. No internal clock division is used.
Figure 4-4 shows the parallel instruction fetches and instruction executions enabled by the Harvard architecture and the fast-access Register File concept. This is the basic pipelining concept to obtain up to 1 MIPS per MHz with the corresponding unique results for functions per cost, functions per clocks, and functions per power-unit.

This section describes the general timing concepts for instruction execution. The LGT8XM core is driven by the core clock (CLKcpu), which comes directly from the clock source selection circuit of the system.
The following figure shows the instruction pipeline execution timing based on the Harvard architecture and the fast access register file concept. This is a physical guarantee that enables the core to achieve 1 MIPS/MHz execution efficiency.

这一章节描述指令执行的一般时序概念。LGT8XM 内核由内核时钟(CLKcpu)驱动,这个时钟直接来自与系统的时钟源选择电路。

下图展示了哈弗构架与快速访问寄存器文件概念基础上的指令流水线执行时序。这是使得内核能够获得1MIPS/MHz 的执行效率的物理保证。