External Interrupt Driver for ATmega32 - Explanation

7. Line-by-Line Code Explanation

File: EXI.c

1- static void (*ApfuncEXTINT[3])(void);

=> Declares an array of function pointers to store callbacks for INT0, INT1, and INT2.

2- void EXI_Enable(uint8 EXINum, uint8 EdgeIntSource)

=> Enables a specific external interrupt and sets the edge trigger.

3- switch (EXINum) {...}

=> Selects which interrupt (INT0, INT1, INT2) to enable and configures the trigger edge.

4- case EXI_INT0: SET_BIT(GICR, INT0); ...

=> Enables INT0 and sets bits 0 and 1 of MCUCR according to EdgeIntSource.

5- case EXI_INT1: SET_BIT(GICR, INT1); ...

=> Enables INT1 and sets bits 2 and 3 of MCUCR according to EdgeIntSource.

6- case EXI_INT2: SET_BIT(GICR, INT2); ...

=> Enables INT2 and sets bit 6 of MCUCSR for trigger configuration.

7- sei();

=> Enables global interrupts.

8- void EXI_Disable(uint8 EXINum)

=> Disables a specific external interrupt by clearing its bit in GICR.

9- void EXI_SetCallBack(void (*pfun)(void), uint8 EXINum)

=> Sets a user-defined callback function for the given interrupt number.

External Interrupt Driver for ATmega32 - Explanation

10- ISR(INT0_vect) { ApfuncEXTINT[EXI_INT0](); }

=> ISR for INT0, calls the registered callback.

11- ISR(INT1_vect) { ApfuncEXTINT[EXI_INT1](); }

=> ISR for INT1, calls the registered callback.

12- ISR(INT2_vect) { ApfuncEXTINT[EXI_INT2](); }

=> ISR for INT2, calls the registered callback.

File: EXI_interface.h

13- #define EXI_INT0 0 ... etc.

=> Macros defining symbolic names for INT0, INT1, INT2 and edge types.

14- Function declarations for enabling, disabling, and setting callbacks.

File: EXI_private.h

15- #define cli() and sei()

=> Inline assembly macros to disable/enable global interrupts.

16- #define INT0_vect ... etc.

=> Interrupt vector names for use with ISR macro.

17- #define ISR(vector) ...

=> Macro to simplify ISR declarations.

18- Bit definitions for GICR, MCUCR, and MCUCSR registers.