Document Name	**External Interrupts** Driver Documentation (SRS &SWS)
Document Author	HEX CLAN
Document Status	Published

Software Specification for External Interrupts

Table of Contents

1. Introduction:

The Atmega32 microcontroller is equipped with several external interrupt pins that allow the system to respond to external events promptly. This software specification document outlines the implementation and usage of the external interrupt feature on the Atmega32 microcontroller.

2. Purpose:

The purpose of this document is to provide a comprehensive understanding of the software implementation for external interrupts on the Atmega32

microcontroller. It serves as a guide for developers, outlining the necessary steps to utilize the external interrupt functionality efficiently.

3. Scope:
This document covers the software implementation for handling external interrupts on the Atmega32 microcontroller. It includes the initialization process, interrupt service routine handling, integration with other modules, and performance considerations.

4. Definitions, Acronyms, and Abbreviations:
ISR: Interrupt Service Routine
MCUCR: MCU Control Register
GICR: General Interrupt Control Register

5. System Overview:
The Atmega32 microcontroller is a high-performance, low-power Atmel 8-bit AVR RISC-based microcontroller. It features 32KB of in-system programmable Flash memory, 2KB of SRAM, and 1KB of EEPROM. The external interrupts are triggered by external events on the INT0 and INT1 pins.

6. External Interrupt Description:
The Atmega32 microcontroller has two external interrupt pins, INT0 and INT1. These pins can be configured to trigger an interrupt on the rising or falling edge of the signal. The external interrupts can be enabled or disabled using the MCUCR and GICR registers.

7. Implementation:
7.1 Initialization of External Interrupts:
To initialize the external interrupts, the following steps should be taken:
a. Configure the INT0 and INT1 pins as inputs.
b. Set the interrupt trigger conditions using the MCUCR register.
c. Enable the external interrupts by setting the corresponding bits in the GICR register.

7.2 Interrupt Service Routine (ISR) Handling:
When an external interrupt is triggered, the MCU jumps to the corresponding ISR. The ISR should be implemented to handle the specific actions required when the external event occurs. The ISR should be kept as short as possible to ensure timely response to the external event.

8. Integration with Other Modules:
The external interrupts can be integrated with other modules, such as timers or communication modules, to create complex event-driven systems. Care should be taken to prioritize and handle the interrupts efficiently to prevent conflicts and ensure the proper functioning of the system.

9. Performance Considerations:
Efficient handling of external interrupts is crucial for the overall performance of the system. Minimizing the execution time of the ISR and optimizing the interrupt handling process can help improve the responsiveness and reliability of the system.

10. Testing:
Comprehensive testing should be conducted to ensure the proper functioning of the external interrupt feature. Test cases should cover various scenarios, including different trigger conditions and interrupt handling routines, to verify the robustness and reliability of the implementation.

11. Conclusion:
The software specification for external interrupts on the Atmega32 microcontroller provides a detailed overview of the implementation process and considerations for efficient usage. By following the guidelines outlined in this document, developers can effectively utilize the external interrupt feature to create responsive and reliable embedded systems.

Contents

1. Scope of Document

This document specifies requirements on the module ADC EXTERNAL_INTERRUPT Driver. The EXTERNAL_INTERRUPT driver
is targeting Successive Approximation EXTERNAL_INTERRUPT Hardware.

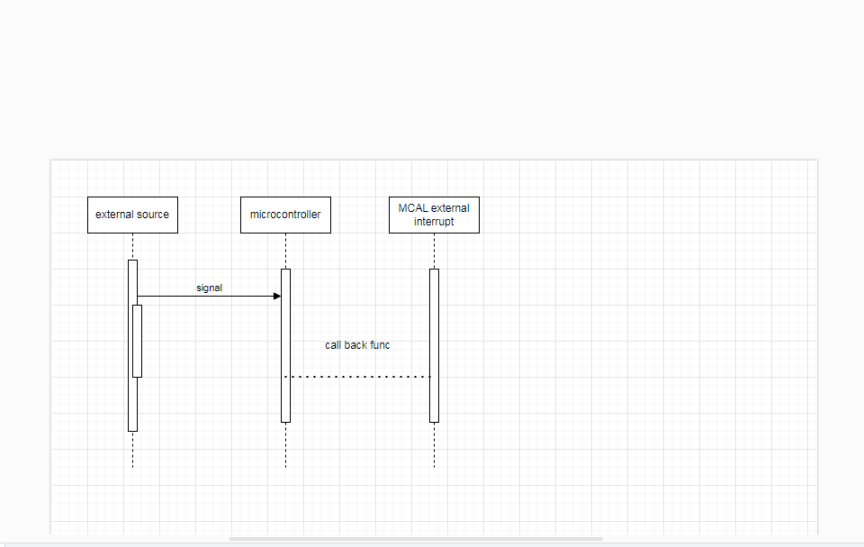
- **API specification**

-Imported types

In this chapter all types included from the following modules are listed:

Module	Header File	Imported Type
lib	STD_TYPES.h	U8 (typedef)
	STD_TYPES.h	OK (Error State)
	STD_TYPES.h	E_NOK (Error State)

- **SequenceDiagrams**



1.1. Functions definitions

1.1.1. void MCAL_EXTINT_voidEnable

Service Name	void MCAL_EXTINT_voidEnable	
Syntax	void MCAL_EXTINT_voidEnable(u8 Copy_u8ExtIntNum , u8 Copy_u8EdgeIntSource)	
Sync/Async	Async	
Reentrancy	Non-reentrant	
Parameters (in)	U8 Copy_u8ExtIntNum, u8 Copy_u8EdgeIntSource	
Parameters (inout)	none	
Parameters (out)	none	
Return value	U8	OK: service is done NOK: service is rejected
Description	This API initialize the EXTINT_voidEnable take ExtIntNum(INT0,INT1,INT2)and EdgeIntSource(ANY_LOGICAL_CHANGE, FALLING_EDGE, RAISING_EDGE, LOW_LEVEL) file and returns error state	
Available via	EXTINT_interface.h	

1.1.2. void MCAL_EXTINT_voidDisable

Service Name	void MCAL_EXTINT_voidDisable	
Syntax	void MCAL_EXTINT_voidDisable(u8 Copy_u8ExtIntNum)	
Sync/Async	Async	
Reentrancy	Non-reentrant	
Parameters (in)	u8 Copy_u8ExtIntNum	
Parameters (inout)	none	
Parameters (out)	none	
Return value	U8	OK: service is done NOK: service is rejected
Description	This API initialize the EXTINT_voidEnable take ExtIntNum(INT0,INT1,INT2)and CLR_BIT after being and returns error state	
Available via	EXTINT_interface.h	

1.1.3. void MCAL_EXTINT_voidSetCallBack

Service Name	void MCAL_EXTINT_voidSetCallBack	
Syntax	void MCAL_EXTINT_voidSetCallBack(void(*Copy_pfun)(void),u8 Copy_u8ExtIntIndex)	
Sync/Async	Async	
Reentrancy	non-reentrant	
Parameters (in)	Copy_pfun)(void),u8 Copy_u8ExtIntIndex	
Parameters (inout)	none	
Parameters (out)	none	
Return value	U8	OK: service is done NOK: service is rejected
Description	This API starts the EXTINT_voidSetCallBack Copy_u8ExtIntIndex reading after beaing done to the pointer Copy_ pfun and returns error state	
Available via	EXTINT_interface.h	