EE2016 Microprocessor Lab & Theory Jul - Nov 2021

EE Department, IIT, Madras.

Experiment 2: Interrupts and Timers in Atmel AVR Atmega

Contents

1 Aim

Using Atmel AVR assembly language programming, implement interrupts and timers in Atmel Atmega microprocessor. The main constraint is that, it should be emulation only (due to ongoing pandemic). Aims of this experiment are

(i) Generate an external (logical) hardware interrupt using an emulation of a push button switch.

(ii) Write an ISR to switch ON an LED for a few seconds (10 secs) and then switch OFF. (The lighting of the LED could be verified by monitoring the signal to switch it ON).

(iii) If there is time, you could try this also: Use the 16 bit timer to make an LED blink with a duration of 1 second.

Also, one needs to implement all of the above using C-interface.

2 Equipment Required

Since this is an emulation based experiment, we need only a PC with the following software : Microchip studio simulation software.

3 Background Material

In order to interface a peripheral, there are two steps: (i) pick or select or address a particular peripheral out of 'n' peripherals and (ii) after choosing the intended one, the operational procedure by which the peripheral is controlled or read or written into or to execute a command specific to the peripheral.

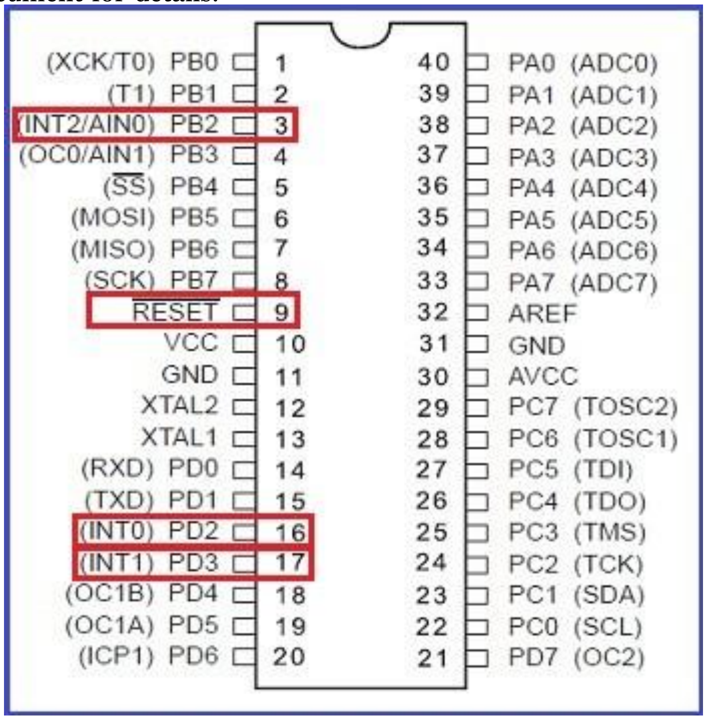
The interrupt is a solution to the 2nd step [For first step, the solution is memory mapped I/O or port mapped I/O]. In 'interrupt', scheme, peripheral initiates an 'interrupt' signal to the CPU, which forces the

PC to point to a specific pre-designated address in program memory. The list of such 'fixed' pre-designated addresses from where the Interrupt Service Routine (ISR) would start is called interrupt vector table.

3.1 Interrupts in Atmega8 AVR Processor

The Atmel ® AVR ® provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate Program Vector in the Program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock Bits BLB02 or BLB12 are programmed. This feature improves software security. See the section Memory Programming on page 215 of Atmel AVR Data sheet document for details.

The lowest addresses in the Program memory space are by default defined as the Reset and Interrupt Vectors. The complete list of Vectors is shown in Interrupts in Section 3.1.2. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 - the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the boot Flash section by setting the Interrupt Vector Select (IVSEL) bit in the General Interrupt Control Register (GICR). Refer to Interrupts in Section 3.1.2 for more information. The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST. Fuse, see “Boot Loader Support Read- While-Write Self-Programming” on page 202 of Atmel AVR data sheet document for details.



When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are dis- abled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction= RETI is executed.

There are basically two types of interrupts. The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the global interrupt enable bit is cleared, the corresponding Interrupt

Flag(s) will be set and remembered until the global interrupt enable bit is set, and will then be executed by order of priority.

The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered.

When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.

Note that the Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be handled by software.

When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction. The following example shows how this can be used to avoid interrupts during the timed EEPROM write sequence.

3.1.1 Interrupt Response Time

The interrupt execution response for all the enabled Atmel ® AVR ® interrupts is four clock cycles minimum. After four clock cycles, the Program Vector address for the actual interrupt handling routine is executed. During this 4-clock cycle period, the Program Counter is pushed onto the Stack. The Vector is normally a jump to the interrupt routine, and this jump takes three clock cycles. If an interrupt occurs during execution of a multi-cycle instruction, this instruction is completed before the interrupt is served. If an interrupt occurs when the MCU is in sleep mode, the interrupt execution response time is increased by four clock cycles. This increase comes in addition to the start-up time from the selected sleep mode. A return from an interrupt handling routine takes four clock cycles. During these four clock cycles, the Program Counter (2 bytes) is popped back from the Stack, the Stack Pointer is incremented by 2, and the I-bit in SREG is set.

3.1.2 Atmel AVR Instructions Related to Interrupts

ATMega8515 Interrupt Vector Table

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address[2]	Source	Interrupt Definition
1	0x000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

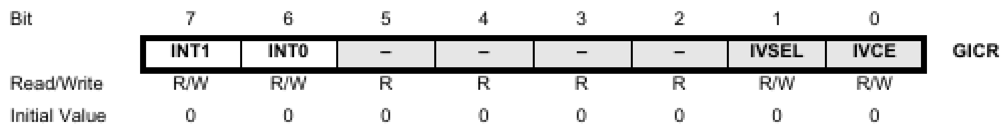
BOOTRST[1]	IVSEL	Reset Address	Interrupt Vectors Start Address
1	0	0x000	0x001
1	1	0x000	Boot Reset Address + 0x001
0	0	Boot Reset Address	0x001
0	1	Boot Reset Address	Boot Reset Address + 0x001

MCU Control Register

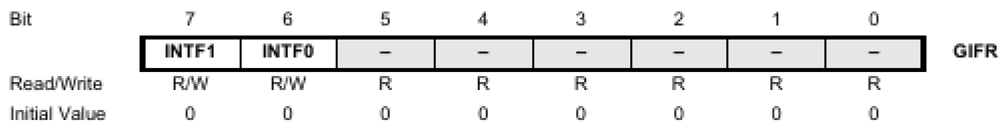
Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

General Interrupt Controller

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	–	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	–	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	–	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

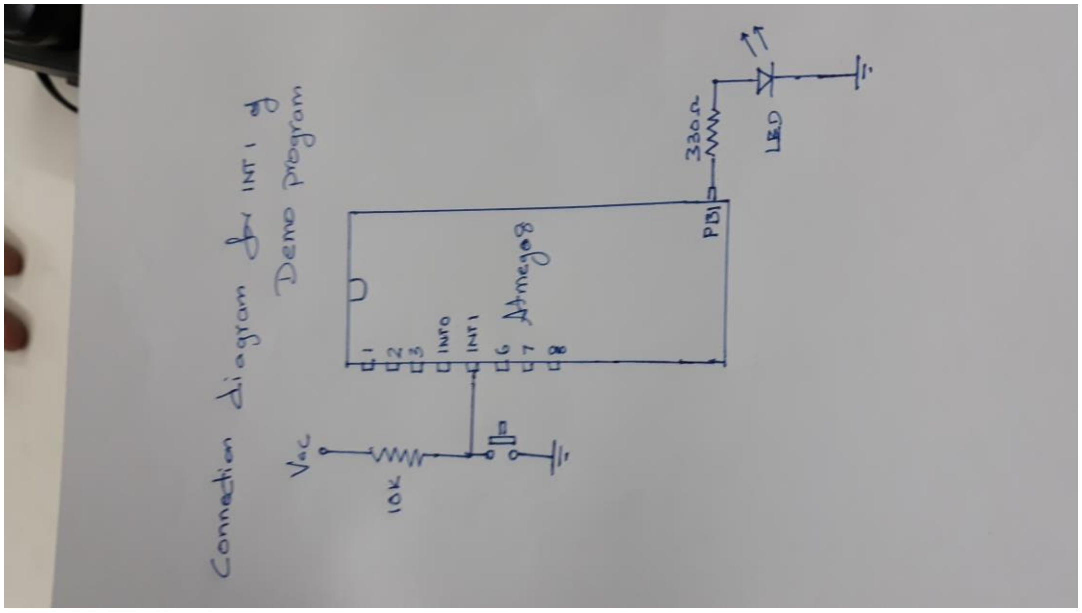
3.1.3 External Interrupts

The external interrupts are triggered by the INT0, and INT1 pins. Observe that, if enabled, the interrupts will trigger even if the INT0..1 pins are configured as outputs. This feature provides a way of generating a software interrupt. The external interrupts can be triggered by a falling or rising edge or a low level. This is set up as indicated in the specification for the MCU Control Register MCUCR. When the external interrupt is enabled and is configured as level triggered, the interrupt will trigger as long as the pin is held low. Note that recognition of falling or rising edge interrupts on INT0 and INT1 requires the presence of an I/O clock, described in "Clock Systems and their Distribution" on page 25 in the AVR (Atmega8L) manual uploaded in moodle. Low level interrupts on INT0/INT1 are detected asynchronously. This implies that these interrupts can be used for waking the part also from sleep modes other than Idle mode. The I/O clockis halted in all sleep modes except Idle mode.

Note that if a level triggered interrupt is used for wake-up from power-down mode, the changed level must be held for some time to wake up the MCU. This makes the MCU less sensitive to noise. The changed level is sampled twice by the Watchdog Oscillator clock. The period of the Watchdog Oscillator is 1 ms (nominal) at 5.0V and 25°C. The frequency of the Watchdog Oscillator is voltage dependent as shown in "Electrical Characteristics TA = -40^{0}C to 85^{0}C" on page 235. The MCU will wake up if the input has the required level during this sampling or if it is held until the end of the start-up time. The start-up time is defined by the SUT Fuses as described in " System Clock and Clock Options" on page 25. If the level is sampled twice by the Watchdog Oscillator clock but disappears before the end of the start-up time, the MCU will still wake up, but no interrupt will be generated. The required level must be held long enough for the MCU to complete the wake up to trigger the level interrupt.

3.1.4 Demo Programs

Interrupt demo program (in both assembly & C) has been uploaded in moodle. There are blanks to be filled in. These are to ensure understanding by the student. You may refer to the manual and then fill them in. There, some lines (in assembly code) are commented for easy understanding. This program implements int1in AVR Atmega8. The corresponding connection diagram is also given below



4 Your Tasks

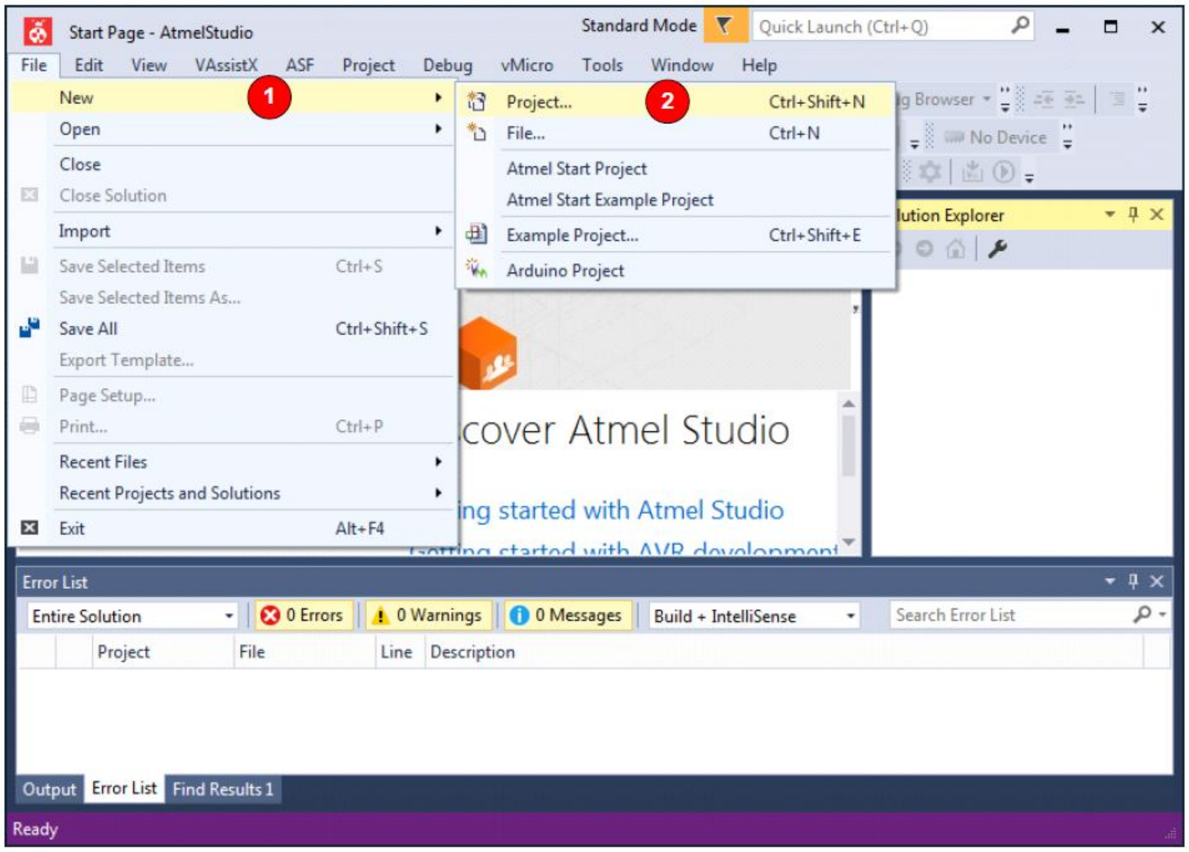
You are given two files (apart from this handout in the directory EE2016F21Exprmnt2AVR.IntrrptC-Intrfcng in moodle): EE2016F21Exp2int1.asm and EE2016F21Exp2.int1.c The former one is an assembly program which implements interrupt using int1, while the later one is a 'C' program using int1.c

1. Fill in the blanks in the assembly code.
2. Use int0 to redo the same in the demo program (duely filled in). Once the switch is pressed the LEDshould blink 10 times (ON (or OFF) - 1 sec, duty cycle could be 50 %). Demonstrate both the cases.
3. Rewrite the program in 'C' (int1). Rewrite the C program for int0.
4. Demonstrate both the cases (of assembly and C).

4.1 Procedure

For assembly programming, follow the procedure of previous experiment. For creating C

1. Go to the ***File*** menu. Choose ***New*** and then ***Project***

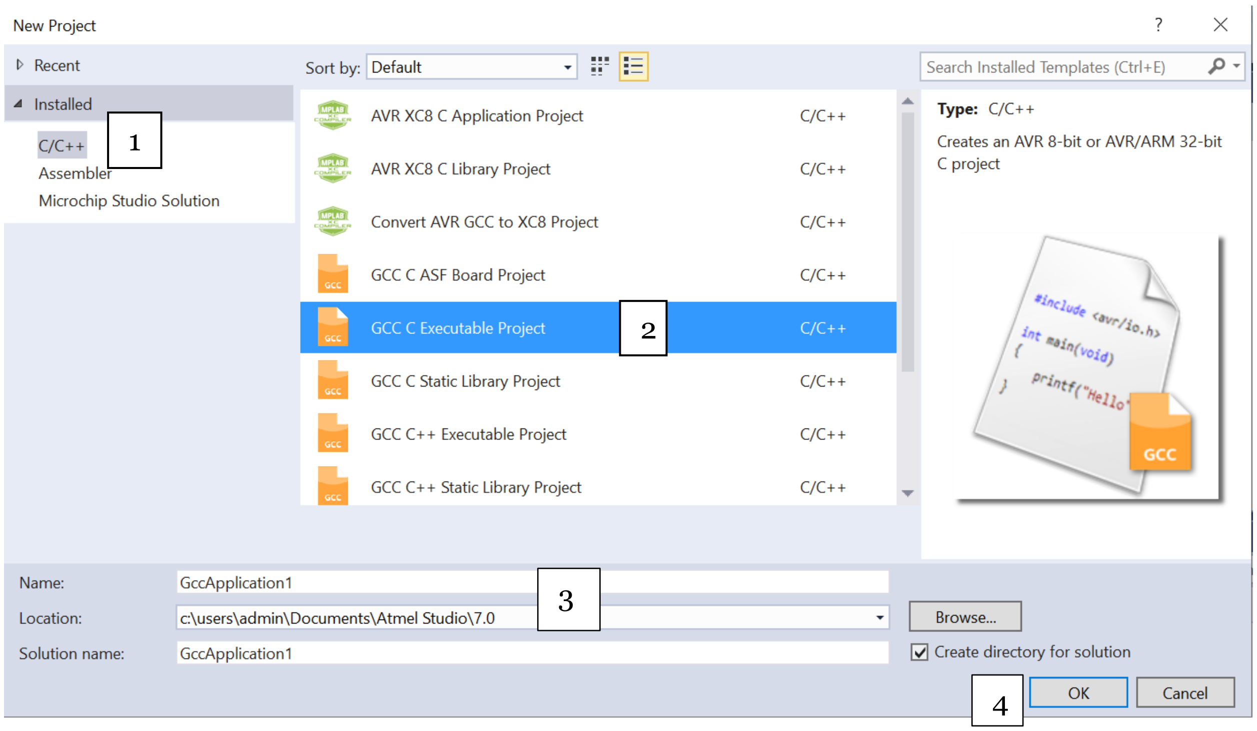


2. In the opened dialog,

a)Choose C/C++

b)Now select GCC C Executable project

c)Rename the project and choose project path using Browse and press Ok


New Project

Recent

Installed

C/C++

Assembler

Microchip Studio Solution

Sort by: Default

Search Installed Templates (Ctrl+E)

AVR XC8 C Application Project	C/C++
AVR XC8 C Library Project	C/C++
Convert AVR GCC to XC8 Project	C/C++
GCC C ASF Board Project	C/C++
GCC C Executable Project	C/C++
GCC C Static Library Project	C/C++
GCC C++ Executable Project	C/C++
GCC C++ Static Library Project	C/C++

Type: C/C++

Creates an AVR 8-bit or AVR/ARM 32-bit C project

Name: GccApplication1

Location: c:\users\admin\Documents\Atmel Studio\7.0

Solution name: GccApplication1

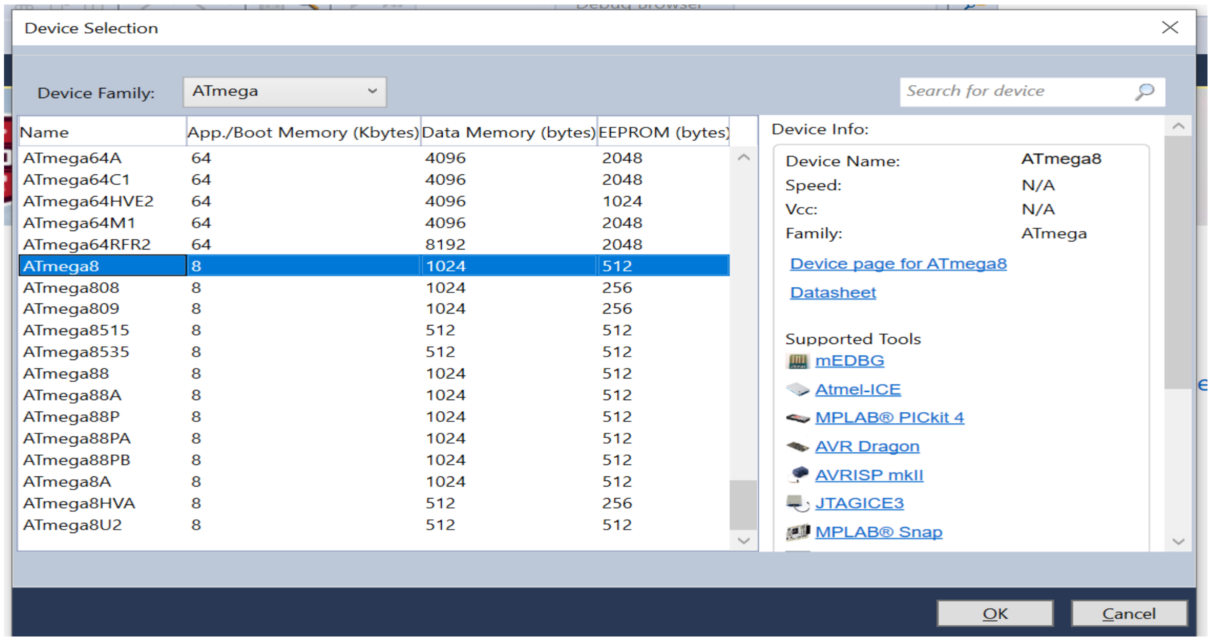
Create directory for solution

3. In the ***Device Selection*** dialog

a. Select ***megaAVR*** as the ***Device family***.

b. Choose ***ATmega8*** (or any other Chips you want to use)

c. Select OK.


Device Selection

Device Family: ATmega

Name	App./Boot Memory (Kbytes)	Data Memory (bytes)	EEPROM (bytes)
ATmega64A	64	4096	2048
ATmega64C1	64	4096	2048
ATmega64HVE2	64	4096	1024
ATmega64M1	64	4096	2048
ATmega64RFR2	64	8192	2048
ATmega8	8	1024	512
ATmega808	8	1024	256
ATmega809	8	1024	256
ATmega8515	8	512	512
ATmega8535	8	512	512
ATmega88	8	1024	512
ATmega88A	8	1024	512
ATmega88P	8	1024	512
ATmega88PA	8	1024	512
ATmega88PB	8	1024	512
ATmega8A	8	1024	512
ATmega8HVA	8	512	256
ATmega8U2	8	512	512

Device Info:

Device Name:	ATmega8
Speed:	N/A
Vcc:	N/A
Family:	ATmega

Device page for ATmega8

Datasheet

Supported Tools

mEDBG

Atmel-ICE

MPLAB® PICkit 4

AVR Dragon

AVRISP mkII

JTAGICE3

MPLAB® Snap

The compiler automatically makes the ***example1*** and adds an assembly file to it.

5 Results

Demonstrate it to TAs before you leave the lab.

6 Debugging issues

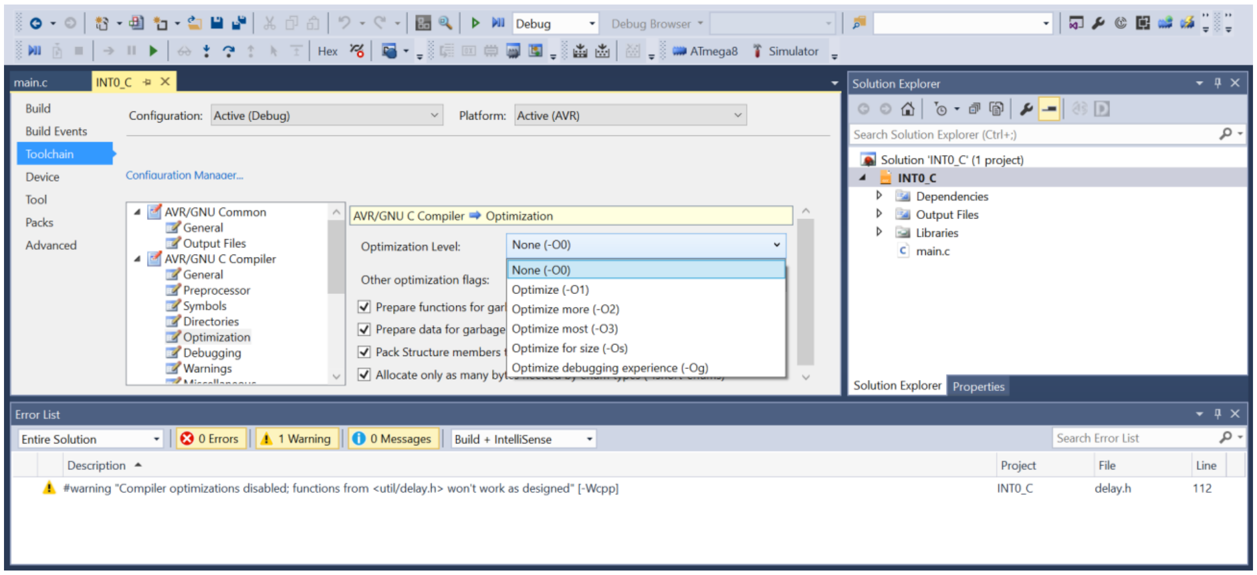
While in debugging mode, before you enter main loop make PIND bits corresponding to INT0 or INT1 to 1, similar to INT1 pin in image from section 3, page 6 connected to VDD (No interrupt initially). While in main loop, make PIND bit 0,emulating the push of a button switch.

6.1 Going from main loop to ISR

We can’t use step over while in debugging mode for ISR(Microchip studio designed it to be that way).Hence we have to set breakpoint in main loop and another breakpoint just after we enter ISR. After interrupt emulation occurs, use continue until next breakpoint to enter ISR, in both assembly in C.

6.2 Setting breakpoint in while loop in C program

We can’t directly set breakpoint in while loop of C program due to optimization by Microchip studio. To disable this optimization, debugger->Toolchain->AVR/GNU complier->set optimization to None.



Configuration: Active (Debug) Platform: Active (AVR)

AVR/GNU C Compiler ➡ Optimization

Optimization Level: None (-O0)

- None (-O0)
- Optimize (-O1)
- Optimize more (-O2)
- Optimize most (-O3)
- Optimize for size (-Os)
- Optimize debugging experience (-Og)

Other optimization flags:

- [✓] Prepare functions for gar
- [✓] Prepare data for garbage
- [✓] Pack Structure members t
- [✓] Allocate only as many by

Description	Project	File	Line
#warning "Compiler optimizations disabled; functions from <util/delay.h> won't work as designed" [-Wcpp]	INT0_C	delay.h	112