

ECE3411 – Fall 2016

Lab 5c.

Debugging using Atmel Studio

Marten van Dijk, Chenglu Jin

Department of Electrical & Computer Engineering
University of Connecticut

Email: {marten.van_dijk, chenglu.jin}@uconn.edu

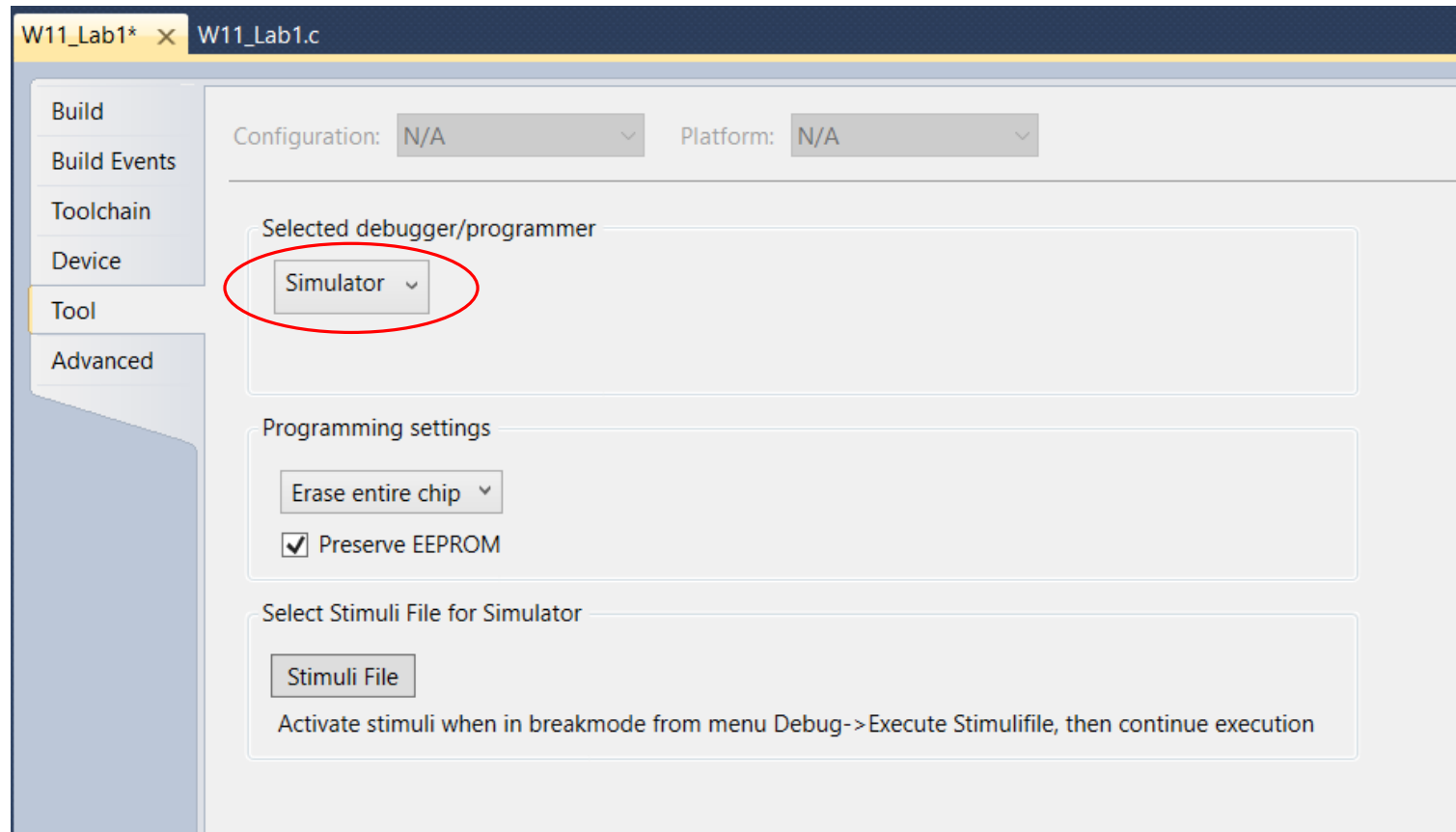
Copied from Lab 5c, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

UConn



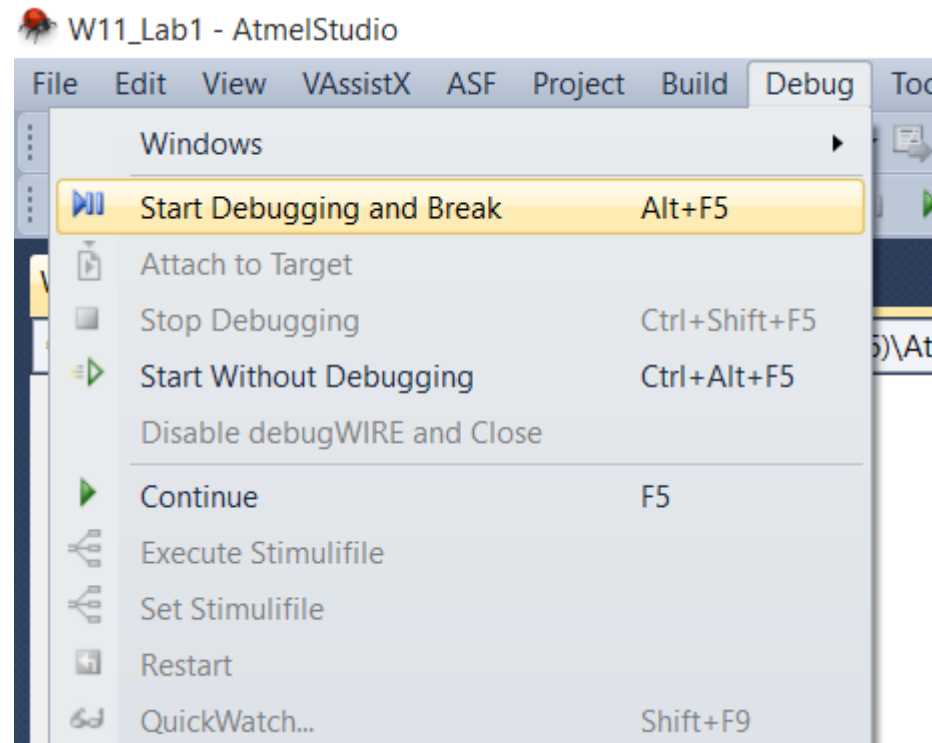
Starting a Debugging Session

- Create a new Atmel Studio project
- Select “Simulator” from the Tool Selection tab



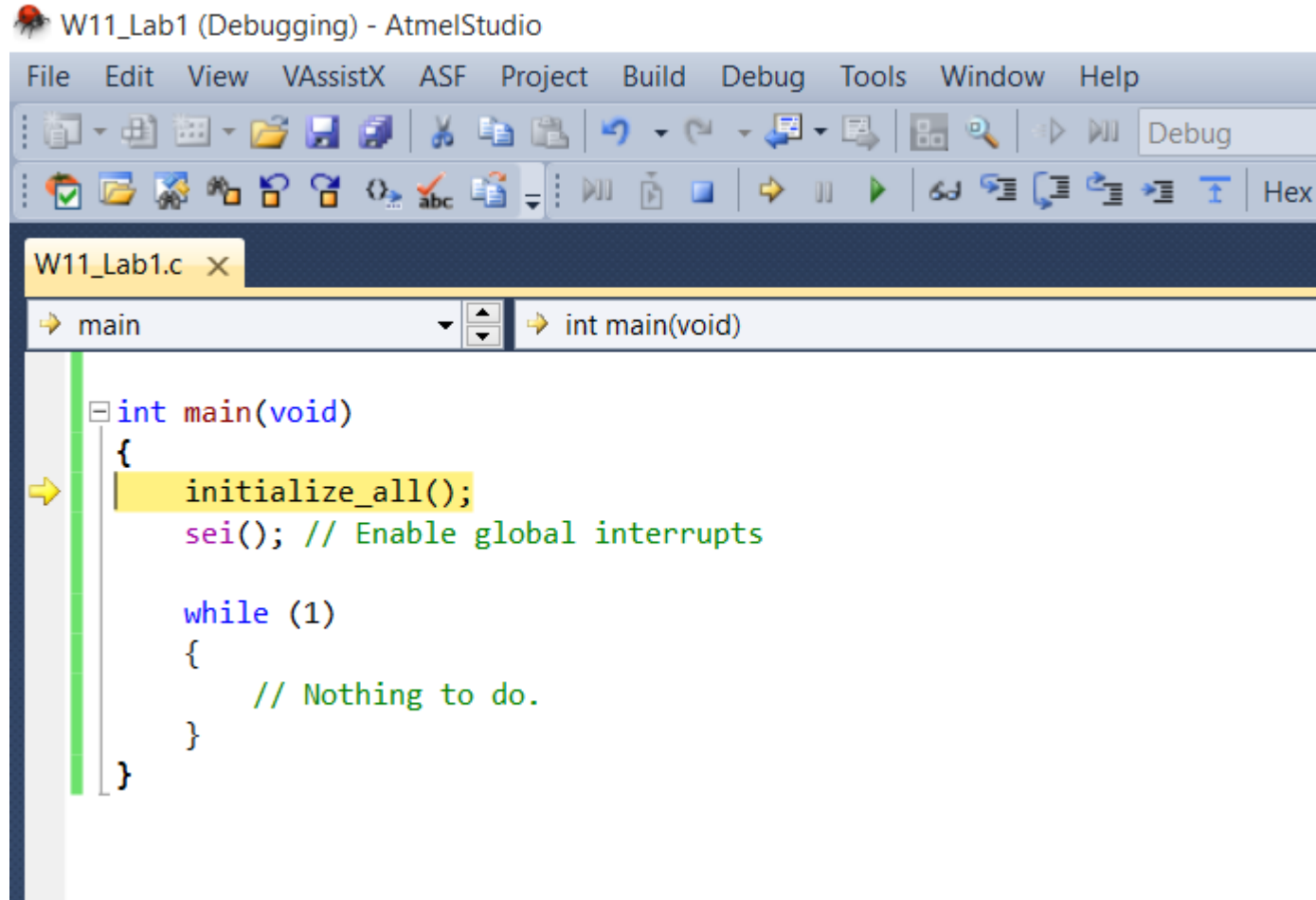
Starting a Debugging Session

- Build the project. (Hit F7)
- From Debug tab, select “Start Debugging and Break”
- The debugger pauses at the start of main.



Start of Debugging Session

- The debugger pauses at the start of main.



The screenshot shows the AtmelStudio IDE interface during a debugging session. The title bar reads "W11_Lab1 (Debugging) - AtmelStudio". The menu bar includes File, Edit, View, VAssistX, ASF, Project, Build, Debug, Tools, Window, and Help. The toolbar contains various icons for file operations, navigation, and debugging. The main window displays the source code for "W11_Lab1.c". The current function being debugged is "main", and the current line of code is "int main(void)". The code is as follows:

```
int main(void)
{
    initialize_all();
    sei(); // Enable global interrupts

    while (1)
    {
        // Nothing to do.
    }
}
```

A yellow arrow points to the first line of the function, indicating the debugger has paused at the start of main. The line "initialize_all();" is highlighted in yellow.

Peripheral Registers in Debugging Session

- Click on I/O view button to see all peripheral registers in an I/O Window

The screenshot shows the IDE interface with the following components:

- Code Editor:** Displays the C code for `main` in `W11_Lab1.c`. The function `initialize_all()` is highlighted.
- Toolbar:** The `I/O View` button is circled in red and labeled "I/O View Button".
- I/O View Window:** A window titled "I/O View" is open on the right, showing a list of peripheral registers. A blue callout points to this window, labeled "I/O Registers Window".

Name	Value
EEPROM	
EXTERNAL_INTERRUPT	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
TWI	
USART0	
WATCHDOG	

Name	Address	Value
------	---------	-------

Adding a Breakpoint in Debugging Session

- Select any instruction in the code
- Right Click and insert a Breakpoint as follows

```
// Timer 1 Compare Match A ISR (TCNT1 = OCR1A)
ISR (TIMER1_COMPA_vect)
{
    // Load new Time period
    OCR1A = time_period;

    // Load new duty cycle
    OCR1B = duty_cycle;
}

ISR(TIMER0_COMPA_vect)
{
    time++;
    a = time*0.001; //convert to seconds
    duty_cycle = sin(2*M_PI*a);
}

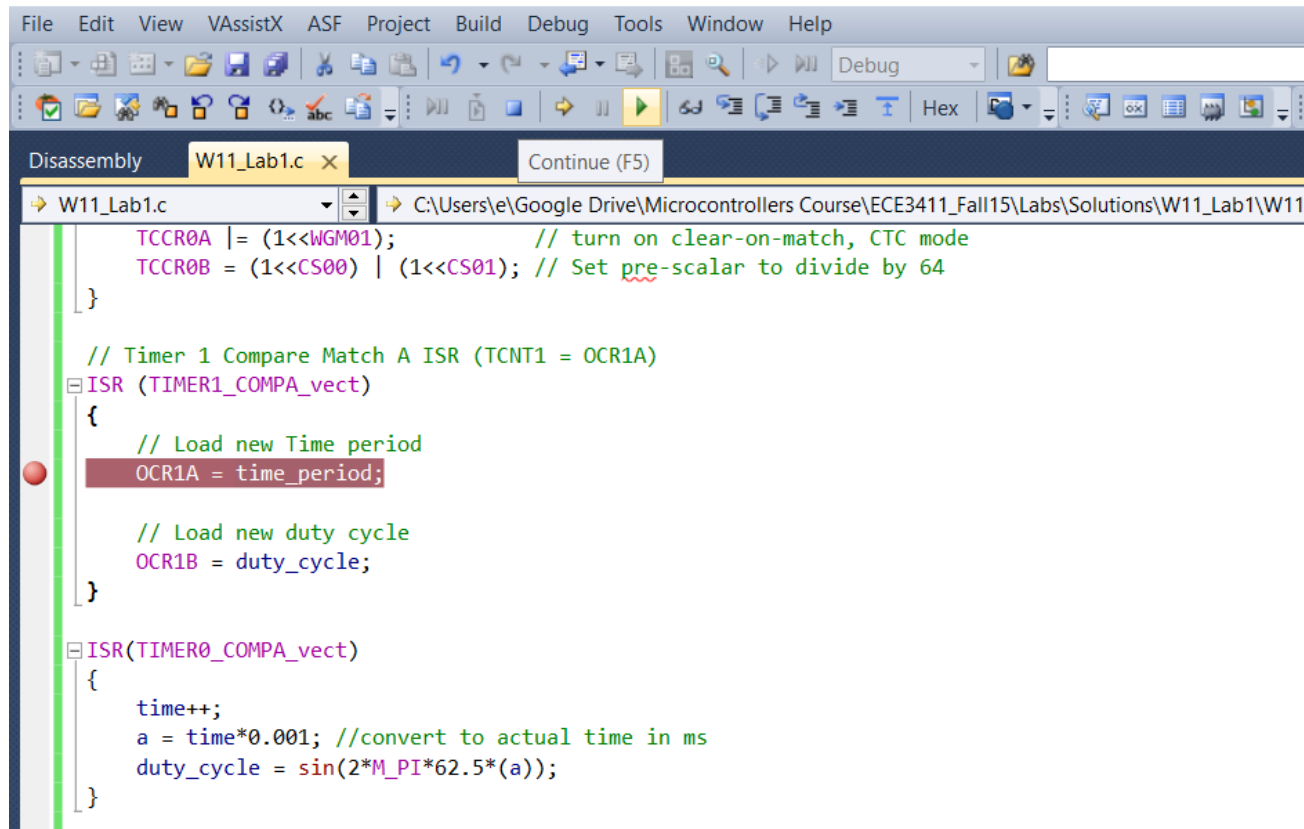
int main(void)
{
```

The screenshot shows a code editor with a context menu open over the line `OCR1A = time_period;`. The menu items are:

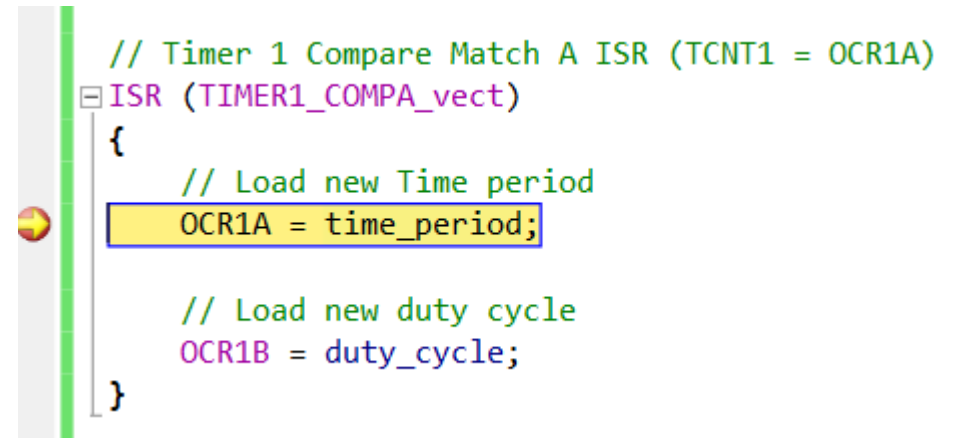
- Goto Implementation (Alt+G)
- Refactor (VA)
- Surround With (VA)
- Insert Snippet... (Ctrl+K, Ctrl+X)
- Surround With... (Ctrl+K, Ctrl+S)
- Breakpoint** (expanded)
 - Add Databreakpoint (Ctrl+Shift+R)
 - Insert Breakpoint**
 - Insert Tracepoint
- Add Watch
- QuickWatch... (Shift+F9)
- Pin To Source
- Show Next Statement (Alt+Num *)

Continue to the next Breakpoint

- After inserting a breakpoint, click Continue (F5)
- The program will stop at Breakpoint as shown in the right window.



```
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Debug
Disassembly W11_Lab1.c x Continue (F5)
W11_Lab1.c C:\Users\...
TCCR0A |= (1<<WGM01); // turn on clear-on-match, CTC mode
TCCR0B = (1<<CS00) | (1<<CS01); // Set pre-scaler to divide by 64
}
// Timer 1 Compare Match A ISR (TCNT1 = OCR1A)
ISR (TIMER1_COMPA_vect)
{
// Load new Time period
OCR1A = time_period;
// Load new duty cycle
OCR1B = duty_cycle;
}
ISR(TIMER0_COMPA_vect)
{
time++;
a = time*0.001; //convert to actual time in ms
duty_cycle = sin(2*M_PI*62.5*(a));
}
```



```
// Timer 1 Compare Match A ISR (TCNT1 = OCR1A)
ISR (TIMER1_COMPA_vect)
{
// Load new Time period
OCR1A = time_period;
// Load new duty cycle
OCR1B = duty_cycle;
}
```

Observing Register/Variable Values at a Breakpoint

- Select particular peripheral and then the register to observe the value. (shown on left)
- Type variable names from your code in Watch Window to monitor their values. (shown on right)

IO View

Filter: []

Name	Value
EEPROM	
EXTERNAL_INTERRUPT	
PORTB	
PORTC	
PORTD	
SPI	
TIMER_COUNTER_0	
TIMER_COUNTER_1	
TIMER_COUNTER_2	
TWI	
USART0	
WATCHDOG	

Name	Address	Value	Bits
TIFR1	0x36	5	□ □ □ □ □ □ □ □
GTCCR	0x43	0	□ □ □ □ □ □ □ □
TIMSK1	0x6F	2	□ □ □ □ □ □ □ □
TCCR1A	0x80	35	□ □ □ □ □ □ □ □
TCCR1B	0x81	25	□ □ □ □ □ □ □ □
TCCR1C	0x82	0	□ □ □ □ □ □ □ □
TCNT1	0x84	19	□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
ICR1	0x86	0	□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
OCR1A	0x88	249	□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
OCR1B	0x8A	0	□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

Watch 1

Name	Value	Type
time_period	249	uint16_t{data}@0x0103
duty_cycle	0	uint16_t{data}@0x010d

Autos Locals Watch 1 Watch 2

Files for today's Lab Tasks

- Download the zipped file from the link below.

http://www.piazza.com/class_profile/get_resource/idhg4rqfhcm1uh/igsgo1qx1j86ok

- This file contains three C code files.
 - Task1.c
 - Task2.c
 - Task3.c

Task1,2: Debugging a buggy PWM

The codes in Task1.c and Task2.c generate a 'rectified' 62.5Hz sine waveform using a 64kHz PWM.

The PWM signal is generated at PB2 using Timer1 such that the duty cycle of the PWM is a function of a 62.5Hz sine wave. I.e. for $f = 62.5$

$$duty\ cycle \propto |\sin(2\pi ft)|$$

- There are some bugs in these codes. Your task is to use Atmel Studio debugger to find the bugs in these codes.

Task3: Debugging a buggy Stopwatch

The code in Task3.c is a buggy implementation of a Stopwatch (1ms resolution) for measuring the total time and the individual lap times of a car racer. The detailed functionality is as follows:

- When SW1 is pressed (i.e. start of the race), Timer1 starts counting the number of milliseconds.
- If SW2 is pressed while the stopwatch is counting (i.e. during the race), it records the current time and prints the time elapsed between this and the previous most recent push. This shows the lap time.
- Finally when SW1 is pressed again (i.e. at the end of the race), the total time and the best (i.e. shortest) lap times are printed on the LCD.
- Timer0 is used to count a debounce delay of 16ms for SW1 and SW2.

Your task is to find the bugs in this code and make it run on your MCU boards!