ECE3411 – Fall 2017
Lecture 3a.

Debugging Techniques

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Debugging.pdf, ECE3411 – Fall 2015,
by Marten van Dijk and Syed Kamran Haider

UCONN

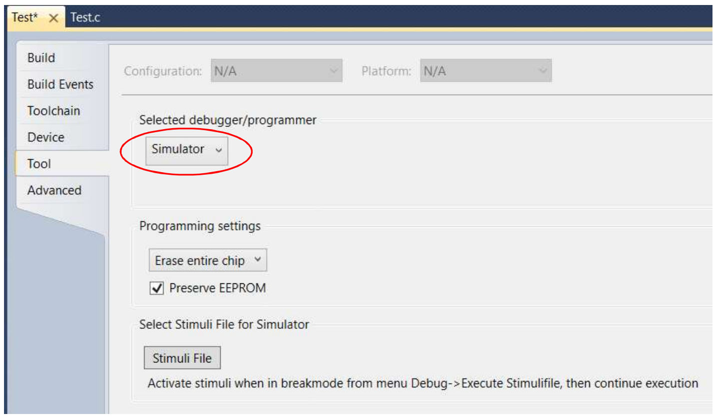
Debugging Techniques

- Debugging in Atmel Studio
  - Simulator mode
  - On-chip debugging using debugWire interface for Xplained Mini kits
- Debugging using Assert library
- Debugging using Hardware Peripherals
  - LEDs, LCD
  - Observing output signals using Oscilloscope

2

Debugging in Atmel Studio (Simulator Mode)

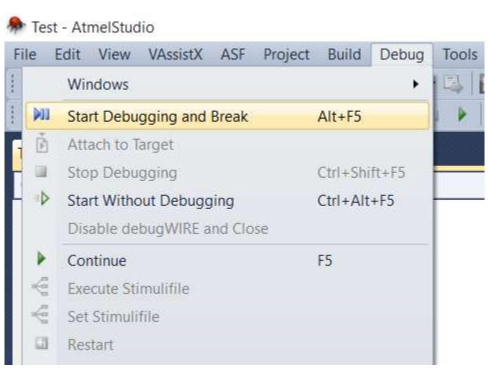
- Create a new Atmel Studio project
- Select “Simulator” from the Tool Selection tab



3

Starting a Debugging Session

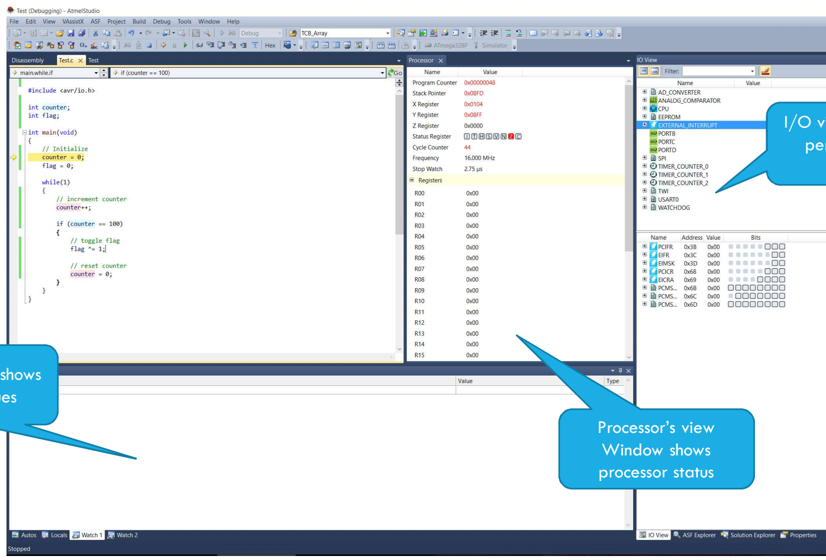
- Build the project. (Hit F7)
- From Debug tab, select “Start Debugging and Break”
- The debugger pauses at the start of main.




```c
#include <avr/io.h>

volatile int counter;
volatile int flag;

int main(void)
{
    // Initialize
    counter = 0;
    flag = 0;

    while(1)
    {
        // increment counter
        counter++;

        if (counter == 100)
        {
            // toggle flag
            flag ^= 1;

            // reset counter
            counter = 0;
        }
    }
}
```

4

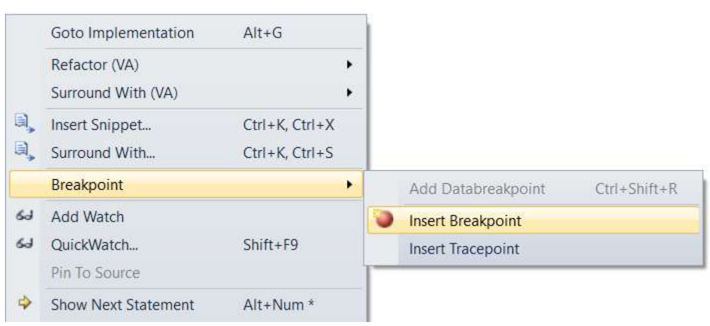
Various Windows in Debugging Session



5

Adding a Breakpoint in Debugging Session

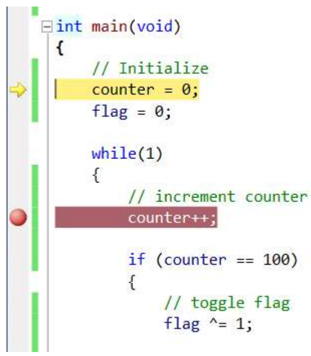
- Select any instruction in the code
- Right Click and insert a Breakpoint as follows

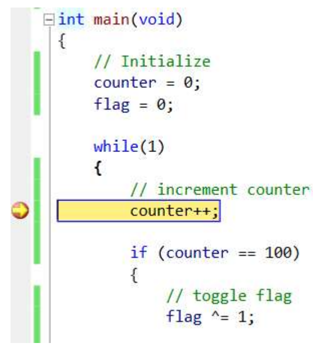


6

Continue to the next Breakpoint

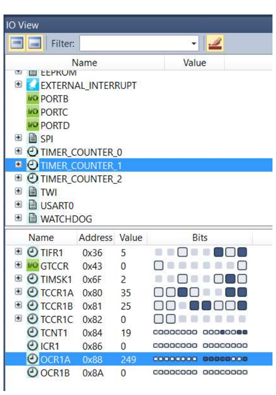
- After inserting a breakpoint, click Continue (F5)
- The program will stop at Breakpoint as shown in the right window.

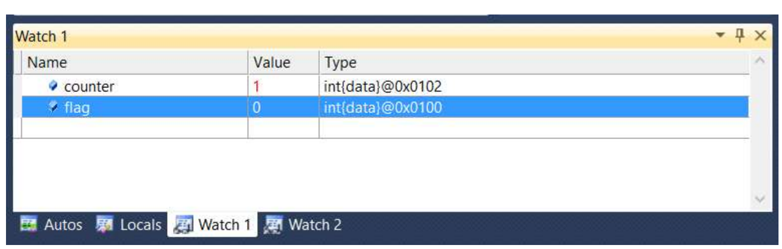




Observing Register/Variable Values at a Breakpoint

- 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)
  - Notice that I have ran through the loop once → counter = 1

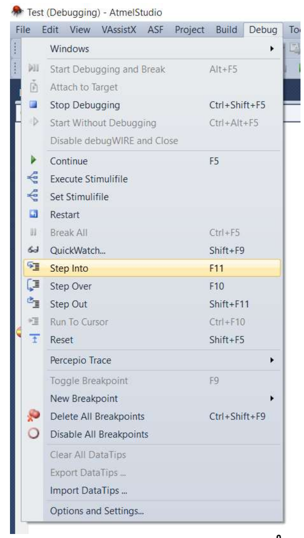




Other Commands in Debugging Session

Inside ‘Debug’ tab, you’ll see various useful debugging commands.

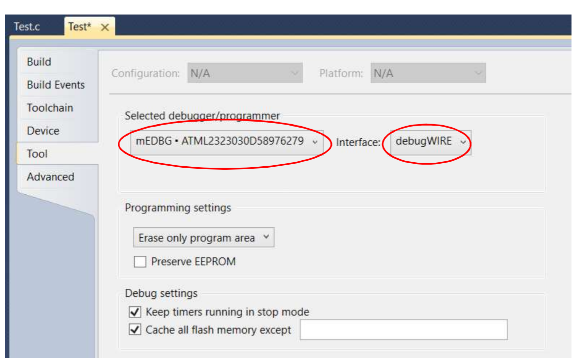
- ‘Stop Debugging’ exists the debugging session.
- ‘Continue’ run the code until the next breakpoint.
- ‘Restart’ restarts the debugging session and runs the code.
- ‘Step Into’ steps through the code line by line.
- ‘Step Over’ jumps over a function and stops after executing it.
- ‘Step Out’ returns from the current function and stops.
- ‘Run to cursor’ runs down to where the cursor is.
- ‘Reset’ command resets the current debugging session.



9

Debugging in Atmel Studio (debugWire Mode)

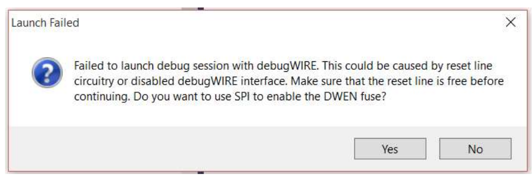
- On-chip debugging for Xplained Mini kits using debugWire interface is also quite similar to the simulator mode.
  - Simulator mode simulates the code as if it is running on the actual microcontroller
  - debugWire allows you to actually run the code on the microcontroller while you debug it step by step.
- Connect the Xplained Mini board with your computer
- Go to the Tool tab and select mEDBG with debugWire interface.



10

Starting a Debugging Session (debugWire Mode)

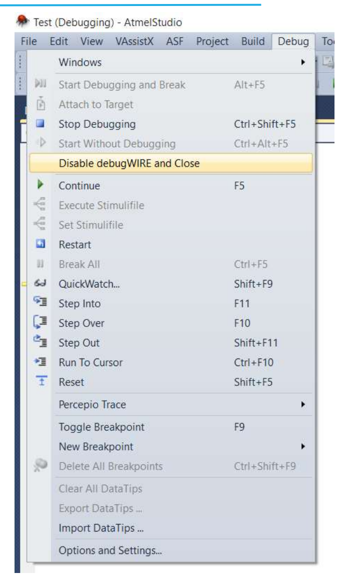
- Build the project (hit F7) and from Debug tab, select "Start Debugging and Break"
- Most likely you'll see an error message asking you to enable DWEN fuse (as shown below).
  - DWEN fuse (debugWire Enable fuse) enables the debugWire interface on your microcontroller.
  - Click 'Yes' on the error message window and enable DWEN fuse.
- The debugger will pause at the start of main, just like simulator mode.
- Now you may use similar debugging techniques as done in Simulator mode
  - Use breakpoints to stop at a particular instruction.
  - Use Watch windows to observe/set program variables.
  - Use I/O view to observe/set the peripheral registers.



11

Exiting a Debugging Session (debugWire Mode)

- It is really important to exit the debugWire debugging session in a proper way!
- To exit the debugging session, click on "Disable debugWire and Close".
  - This will first disable the DWEN fuse in the microcontroller.
  - Then it will close the debugging session.
- If DWEN fuse is not disabled, you'll not be able to program the microcontroller in ISP mode (which we want to use most frequently).



Debugging using Assert library

- http://people.ece.cornell.edu/land/courses/ece4760/Debugging/index.htm has many great suggestions
- One can use the assert library (http://en.wikipedia.org/wiki/Assert.h) to test assertions in code
- Example:

```
//set up the debugging utility ASSERT
#define __ASSERT_USE_STDERR
#include <assert.h>

//test assertion - will print message if argument is NOT true;
assert(time<10);
```

13

Debugging using Hardware Peripherals

- Debugging can also be performed by hardware peripherals.
  - By setting GPIO pins, for example, one can test the frequency of ISRs or certain program conditions (i.e. PORTD |= 0x01; when something happens) and measure results with an oscilloscope.
- Once the LCD lab has been done, one can also display variables and conditions on the screen as code is executed, if there are problems.

14

ECE3411 – Fall 2017
Lab3a.

Debugging using Atmel Studio, Measuring Human Reaction Time, Timer 1 Capture Interrupt

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lab 5c and Lab 4a, ECE3411 – Fall 2015, by Marten van Dijk and Syed Kamran Haider

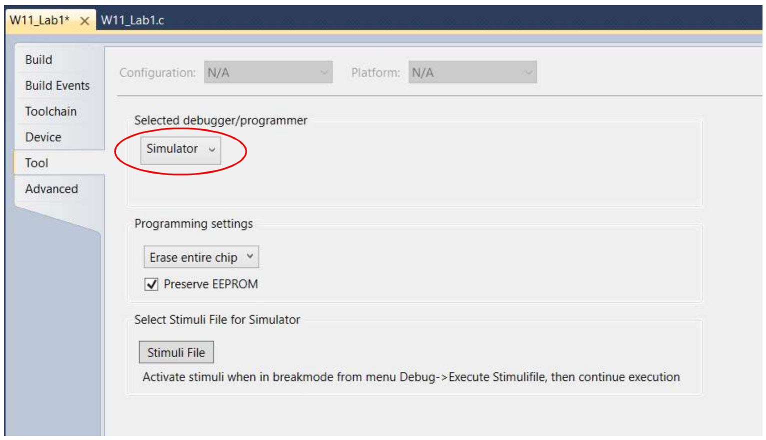
Task 1: Debugging

1. Download the buggy code (Lab3aBuggy.c) from Piazza under resources.
   - Correct the syntax errors in it.

2. Read the slide deck about debugging techniques.
   - The spec of the buggy code is that we want to use eight LEDs to show the number of button presses, but if you program your board with this buggy code, you will find that the count keeps incrementing.
   - Use simulator or DebugWire to help you fix this code.

Starting a Debugging Session

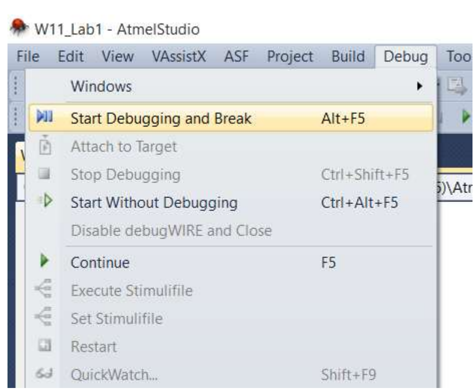
- Create a new Atmel Studio project
- Select "Simulator" from the Tool Selection tab



3

Starting a Debugging Session

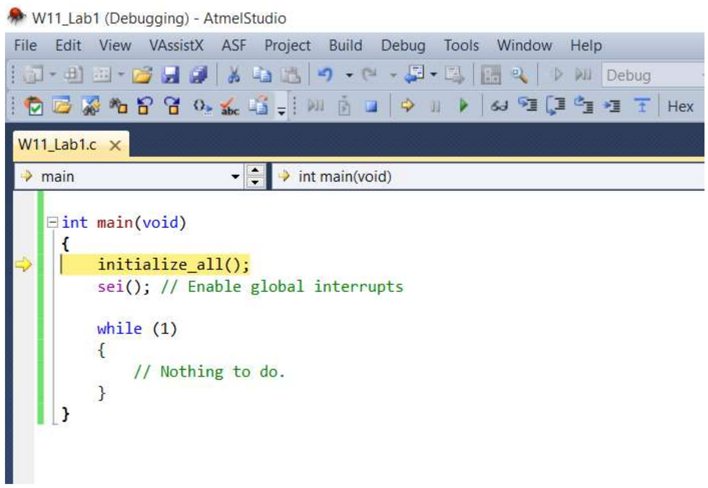
- Build the project. (Hit F7)
- From Debug tab, select "Start Debugging and Break"
- The debugger pauses at the start of main.



4

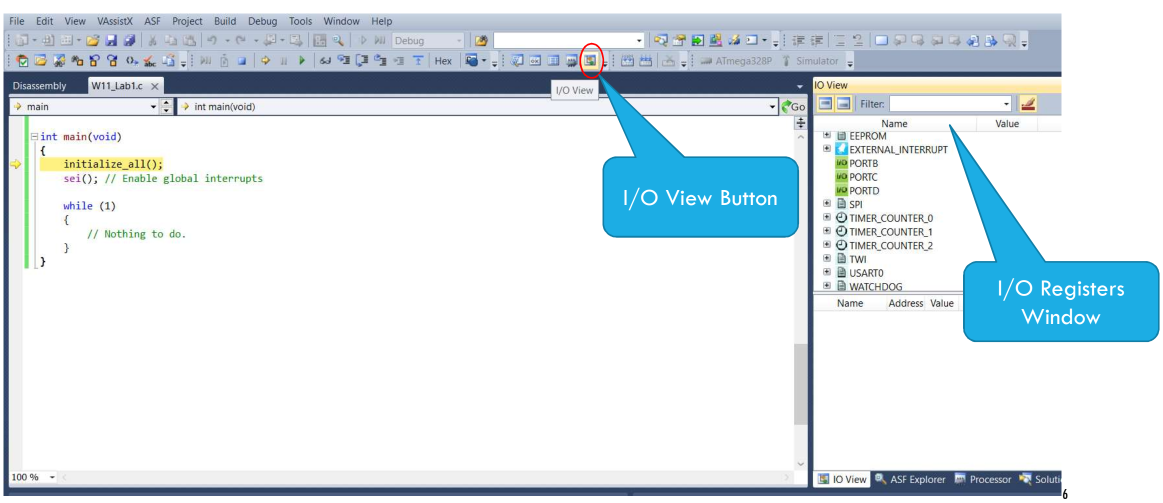
Start of Debugging Session

- The debugger pauses at the start of main.



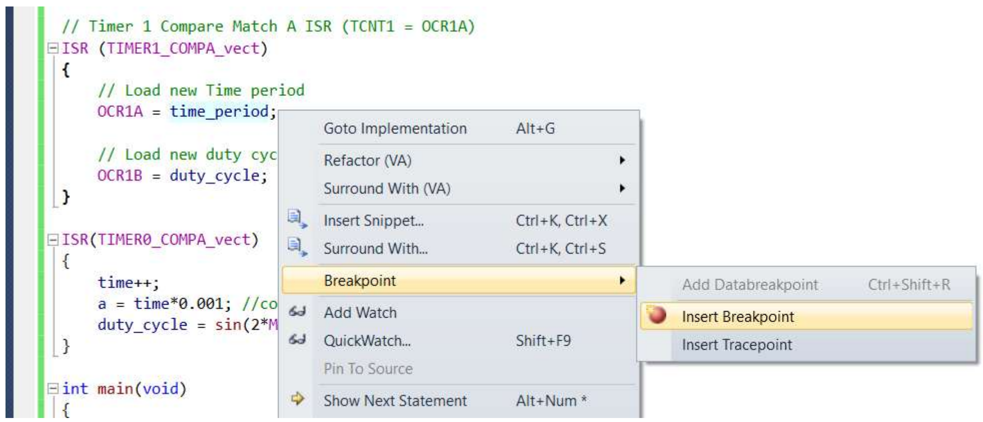
Peripheral Registers in Debugging Session

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



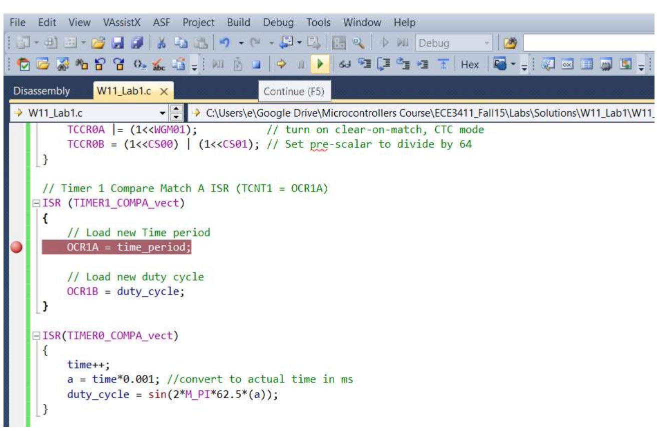
Adding a Breakpoint in Debugging Session

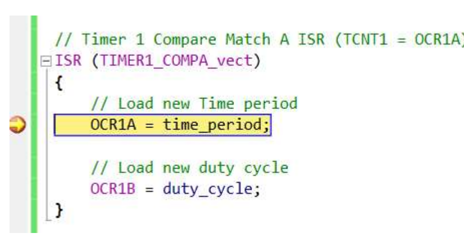
- Select any instruction in the code
- Right Click and insert a Breakpoint as follows



Continue to the next Breakpoint

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

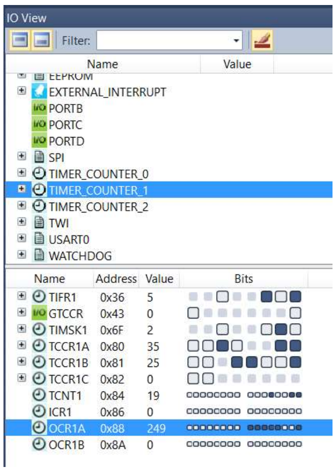


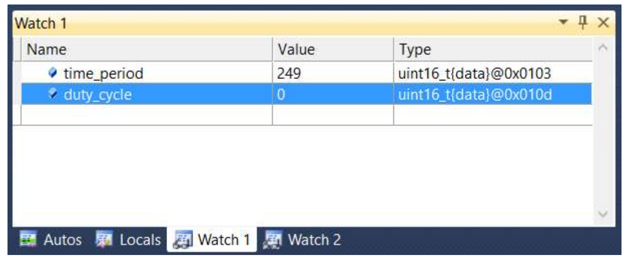


8

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)





9

Files for today's Lab Tasks

- Download the zipped file **Lab3a.zip** from piazza under resources section.
- This file contains three C code files.
  - Fix_it.c

10

Task2: Measuring the Human Reaction Time

Implement a system to measure the Human Reaction Time down to a resolution of 1ms.

In particular:

1. Print a message on UART for the user to get ready
2. Wait for some random amount of time, e.g. between 2 to 5 seconds
3. Turn on a LED & start Timer1
4. The user is supposed to push a button as soon as the LED turns on
5. Read Timer1 to measure the time between the two events, i.e. tuning on the LED and detecting a button push
6. Print the reaction time in milliseconds on UART

Task3: Experimenting with Capture Interrupt

Run the sample code demonstrating “Timer1 Capture Interrupt” provided in Lec2c.

- Connect PB3 (OC2A) to PD7 (AIN1)
- This program uses Timer1 Capture Interrupt to accurately measure Polling time for Task1().
- It then prints the actual time (200 cycles) measured by Timer1 and the time observed by polling mechanism.
- Your task is to vary the time “t1” that controls the printing rate.
- Why does the observed polling time vary with “t1”?

ECE3411 – Fall 2017
Lecture 3b.

External Interrupts
Pin Change Interrupts
Task Based Programming

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lecture 3c and Lecture 4a, ECE3411 – Fall 2015,
by Marten van Dijk and Syed Kamran Haider
Based on the Atmega328P datasheet

UCONN

External Interrupts

- Chapter 12 datasheet
- INT0 & INT1
  - Can be triggered by a falling or rising edge or a low level → EICRA (External Interrupt Control Register A)
  - Low level interrupt is detected asynchronously → can be used to wake from idle mode as well as sleep modes (will see one such example in a forthcoming lecture)
    - If used for wake-up from power-down, the required level must be held long enough for the MCU to complete the wake-up to trigger the level interrupt. (Start-up time defined by SUT and CKSEL fuses, chapter 8)
- PCINT23..0
  - The pin change interrupt PCI0 will trigger if any enabled PCINT7..0 pin toggles
  - The pin change interrupt PCI1 will trigger if any enabled PCINT14..8 pin toggles
  - The pin change interrupt PCI2 will trigger if any enabled PCINT23..16 pin toggles

Interrupt Vectors

- http://www.atmel.com/webdoc/AVRLibcReferenceManual/group__avr__interrupts.html

INT0_vect	SIG_INTERRUPT0	External Interrupt 0	AT90S1200, AT90S2313, AT90S2323, AT90S2333, AT90S2343, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega32, ATmega323, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega406, ATmega48P, ATmega64, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega8, ATmega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny11, ATtiny12, ATtiny13, ATtiny15, ATtiny22, ATtiny2313, ATtiny26, ATtiny28, ATtiny43U, ATtiny48, ATtiny45, ATtiny25, ATtiny85, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
INT1_vect	SIG_INTERRUPT1	External Interrupt Request 1	AT90S2313, AT90S2333, AT90S4414, AT90S4433, AT90S4434, AT90S8515, AT90S8535, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM3, AT90PWM2, AT90PWM1, AT90CAN128, AT90CAN32, AT90CAN64, ATmega103, ATmega128, ATmega1284P, ATmega16, ATmega161, ATmega162, ATmega163, ATmega168P, ATmega32, ATmega323, ATmega328P, ATmega32HVB, ATmega406, ATmega48P, ATmega64, ATmega8, ATmega8515, ATmega8535, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATmega16HVA, ATtiny2313, ATtiny28, ATtiny48, ATtiny261, ATtiny461, ATtiny861, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
PCINT0_vect	SIG_PIN_CHANGE0	Pin Change Interrupt Request 0	ATmega162, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega406, ATmega48P, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny13, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, ATtiny45, ATtiny25, ATtiny85, AT90USB162, AT90USB82, AT90USB1287, AT90USB1286, AT90USB647, AT90USB646
PCINT1_vect	SIG_PIN_CHANGE1	Pin Change Interrupt Request 1	ATmega162, ATmega165, ATmega165P, ATmega168P, ATmega169, ATmega169P, ATmega325, ATmega3250, ATmega3250P, ATmega328P, ATmega329, ATmega3290, ATmega3290P, ATmega32HVB, ATmega406, ATmega48P, ATmega645, ATmega6450, ATmega649, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny43U, ATtiny48, ATtiny24, ATtiny44, ATtiny84, AT90USB162, AT90USB82
PCINT2_vect	SIG_PIN_CHANGE2	Pin Change Interrupt Request 2	ATmega3250, ATmega3250P, ATmega328P, ATmega3290, ATmega3290P, ATmega48P, ATmega6450, ATmega6490, ATmega88P, ATmega168, ATmega48, ATmega88, ATmega640, ATmega1280, ATmega1281, ATmega2560, ATmega2561, ATmega324P, ATmega164P, ATmega644P, ATmega644, ATtiny48

Interrupt Vector Table

Table 11-6. Reset and Interrupt Vectors in ATmega328P

VectorNo.	Program Address[(2)]	Source	Interrupt Definition
1	0x0000[(1)]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt

- Notice that the external interrupts and pin interrupt are at the top of the table
- They will be the first to be checked after an ISR finishes → They have priority
- Usage: Program a SW interrupt for executing an *atomic* piece of code
  - A pin is set as an output
  - Main code toggles the pin
  - This creates a PCINT HW event and sets a corresponding flag
  - Interrupt unit will scan this flag first and prioritizes the corresponding PCINT ISR (i.e., if during toggling another ISR is called due to some other HW event, then once this ISR is finished the PCINT ISR will be called next)
  - The PCINT ISR will be fully executed without interruption → an atomic execution

Example PCINT21 = PD5

DDRD |= (1<<DDD5); //**PD5=PCINT21 is output**

12.2.6 PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

PCMSK2 = (1<<PCINT21); //**toggling PD5 sets flag**

12.2.5 PCIFR – Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

When PD5 toggles, flag PCIFR & (1<<PCIF2) changes from 0 (0 as an integer represents 0x00) to (1<<PCIF2) (which, represented as an integer, equals 4)

12.2.4 PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCICR |= (1<<PCIE2); //**Enable interrupt for**
//**PCIFR.PCIF2**

Write
- ISR(PCINT2_vect){ Atomic code;}
- If the atomic code needs to be executed in the main program, just toggle PORTD ^= (1<<PORTD5);

Sequence of Events

1. In main program toggle PORTD ^= (1<<PORTD5);
2. PCIFR.PCIF2 is set to 1
3. PCICR |= (1<<PCIE2); → Interrupt unit checks PCIFR.PCIF2
4. If currently an ISR is executing, finish its execution and start the next instruction in the main program
5. As soon as the current instruction in the main program is finished, the interrupt unit checks for flags with enabled interrupts
6. The interrupt unit does this in round robin fashion but starts at the top of the interrupt vector table after an ISR is finished → prioritizes RESET over external interrupts over pin interrupts over the rest
7. Looks up address corresponding to ISR(PCINT2_vect), saves register state, puts PC on stack, etc.
8. Execute without any interruption ISR(PCINT2_vect){ Atomic code;}
9. During RETI state is restored, flag PCIFR.PCIF2 is cleared, and PC points to the next instruction in the main program

NOTE: Instead of PORTD ^= (1<<PORTD5); the main code can also directly set PCIFR |= (1<<PCIF2);

INT1

- Programming external interrupt INT1 = PD3 on falling edge
  - Switch connected to PD3 (set to PD3 to input): DDRD &= ~(1<<DDD3);
  - #define SW_PRESSED !(PIND & (1<<PIND3))
  - If SW_PRESSED {…} checks whether PIND & (1<<PIND3) == 0
  - PD3 low means pressed and PD3 high means not pressed: Want to detect falling edge
- EICRA |= (1<<ISC11);

12.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

- EIMSK |= (1<<INT1);

Need to write ISR and implement a debounce state machine …

12.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

INT1

```
#define SW_PRESSED !(PIND & (1<<PIND3))

void Initialize(void)
{
   DDRD &= ~(1<<DDD3);
   EICRA |= (1<<ISC11);
   EIMSK |= (1<<INT1);
   …. Timer 0 ….
   poll_time = POLLING_DELAY;
   DebounceFlag = 0;
}

void ISR(TIMER0_COMPA_vect)
{
   if ((poll_time>0) && (DebounceFlag==1)) --poll_time;
   …
}

ISR(INT1_vect)
{
   EIMSK &= ~(1<<INT1); // Disable interrupt
   … record this event …
   DebounceFlag = 1;
}
```

```
void PollButton(void)
{
  if SW_PRESSED { … latest recorded event is for a button push …}
  DebounceFlag = 0;
  poll_time = POLLING_DELAY;
  EIMSK |= (1<<INT1);
}

int main(void)
{
   Initialize();
   sei();

   while(1)
   {
      if (poll_time == 0) {PollButton();}
      …
   }
}
```

Debouncing with a Pin Interrupt

- Instead of using INT1 we can use a pin interrupt
- The pin toggles:
  - Wrap all ISR code in an extra if statement
  - If SW_PRESSED { .. Code .. }
  - Now we will only execute Code if the button transitions from not-pressed to pressed.

9

Stop Watch

- The ISR records the moment of the falling edge
- Represented by a SW counter maintained in ISR(TIMER0_COMPA_vect)
- Only if the button is really pressed, PollButton() will set a flag telling the main program that the recorded event is valid.
- The main while loop polls the flag and as soon as it is set it e.g. prints the recorded time after which the flag is set back to invalid.
- All kinds of variations possible

10

Example Problem 1

Consider the following code:

```
ISR(TIMER0_COMPA_vect)
{
   if (flag_timer > 0) {flag_timer--;}
   if (flag_timer == 0) {flag = ??; }
}

ISR(INT1_vect)
{
   flag = ??;
   flag_timer = ??;
}
```

Assume ISR(TIMER0_COMPA_vect) is triggered every 1ms. How should the question marks be filled in such that

- as soon as ISR(INT1_vect) is triggered, then flag is set to 1, and
- as soon as 1 second (with approx 1ms precision) has passed since the last time ISR(INT1_vect) was triggered, flag is set to 0.

Example Problem 1

Solution:

```
ISR(TIMER0_COMPA_vect)
{
   if (flag_timer > 0) {flag_timer--;}
   if (flag_timer == 0) {flag = 0; }
}

ISR(INT1_vect)
{
   flag = 1;
   flag_timer = 1000;
}
```

Assume ISR(TIMER0_COMPA_vect) is triggered every 1ms. How should the question marks be filled in such that

- as soon as ISR(INT1_vect) is triggered, then flag is set to 1, and
- as soon as 1 second (with approx 1ms precision) has passed since the last time ISR(INT1_vect) was triggered, flag is set to 0.

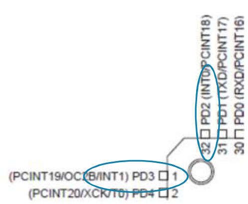
Example Problem 2

- One of your colleagues has already written the code for a task() (which takes about 100 micro seconds) and asks you to write:
- The main code int main(void), which
  - starts by setting registers, enabling interrupts, and executing task() for a first time, and
  - concludes with a while loop which starts executing task() as soon as within a 1 second (with approx 1ms precision) time frame each of the two pins 1 and 32 have signaled a falling edge after the last time task() finished executing
- **Example**: Consider the moment when task() finished executing at for example t=10.0001 seconds.
- Suppose that pin 1 signals falling edges at times t1=13, t1=13.7, and t1=14.3 seconds, pin 32 signals falling edges at times t0 = 12, t0= 13.5, and t_0=14.2 seconds. Assume for the purpose of this example that no other falling edges happen.
- After time t=10.0001 when task() finished executing, the first moment each of the two pins signal a falling edge within a 1 second time frame happens at t=13.5. So, task() should start executing at time t=13.5.
- Suppose it finishes at t=13.5001. After time t=13.5001, the first moment each of the two pins all signal a falling edge within a 1 second time frame happens at t=14.2. So, task() should again start executing at time t=14.2.

13

Example Problem 2

- Besides the main code you are also required to write the appropriate ISRs and declare variables. Assume the MCU runs at 20MHz. You can use the next two pages to write your code.
- **Hint**: As in problem 1, program a flag0 and a flag1 for each of the two pins: As soon as they sum up to 2, each pin triggered an ISR within the last 1 second timeframe.



14

Example Problem 2

```
// Put the declaration of your global variables here:
#define t_flag 1000 // 1000ms = 1 second

volatile int flag0, flag1;
volatile int flag0_timer, flag1_timer;

ISR(TIMER0_COMPA_vect)
{
   // Put your code here:
   if (flag0_timer > 0) {flag0_timer--;}
   if (flag0_timer == 0) {flag0 = 0; }

   if (flag1_timer > 0) {flag1_timer--;}
   if (flag1_timer == 0) {flag1 = 0; }
}
```

```
ISR(INT0_vect)
{
   // Put the code of your second ISR here:
   flag0 = 1;
   flag0_timer = t_flag;
}

ISR(INT1_vect)
{
   // Put the code of your third ISR here:
   flag1 = 1;
   flag1_timer = t_flag;
}
```

15

Example Problem 2

```
int main(void)
{
  // Put your code of the main body (including initializations) here:

  // An accurate 1ms timer (as explained in class):
  TIMSK0 = 2; // enable interrupt
  TCCR0A = 0x02; // return on clear-on-match
  TCCR0B = 0x02; // prescalar @ 8
  OCR0A = 249; // each time tick is 8(OCR0A+1)/20MHz = 1ms exactly
  // An accurate enough timer is needed otherwise the flag_timers drift with respect
  // to real time and we may not meet the specification of ~1ms precision.

  // Initialize external interrupts INT0 (= pin 32) and INT1 (= pin 1) on falling edges
  DDRD = 0x00; // D.2 = pin 32 and D.3 = pin 1 are inputs
  EICRA = (1<<ISC01) | (1<<ISC11);
  EIMSK = (1<<INT0) | (1<<INT1);

  task_timer = t_task;
  flag0 = 0;
  flag1 = 0;

  // Globally enable interrupts
  sei();
```

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	

16

Example Problem 2

```
// Execute task() before entering the while loop
// This allows us to formally meet the specifications of the program
task();

while (1)
{
  if (flag0 + flag1 == 2) // See hint
  {
    task();
    // All flags should be reset, since we just finished executing task()
    flag0 = 0;
    flag1 = 0;
  }
}

return 0;
}
```


Example Problem 3

- One of your colleagues has already written the code for two tasks task1() and task2() (each taking only about 100 micro seconds) and asks you to write: The main code int main(void), which
  - starts by setting registers and enabling interrupts, and
  - concludes with a while loop which
    - starts executing task1() every 1 millisecond (as accurate as possible), and
    - starts executing task2() as soon as
      1. a rising edge is received over pin 1 since the last time task2() finished executing and
      2. at least 1 second has passed since the last time task2() finished executing.

    Example: task2() finished executing at time t=0.

    (a) If the next rising edge after t=0 is detected at e.g. time t=300 milliseconds, then the while loop waits another 700 milliseconds (such that 1 full second has passed) before it starts executing task2().

    (b) If the next rising edge after t=0 is detected at e.g. time t=1100 milliseconds, then the while loop immediately starts executing task2() (since 1 full second has already passed).

- Besides the main code you are also required to write the appropriate ISRs. Assume the MCU runs at 20MHz.

Example Problem 3

```
// Put the declaration of your global variables here:
// define t1 as 1ms and t2 as 1000ms = 1 second
#define t1 1
#define t2 1000

volatile int task1_timer;
volatile int task2_timer;
volatile int flag;

ISR(TIMER0_COMPA_vect)
{
  // Put the code of your first ISR here:
  // Set up virtual timers
  if (task1_timer > 0) {task1_timer--;}
  if (task2_timer > 0) {task2_timer--;}
}

ISR(INT1_vect)
{
  // Put the code of your second ISR here:
  // Set flag
  flag = 1;
}
```


Example Problem 3

```
int main(void)
{
  // Put the code the main body here:

  // An accurate 1ms timer (as explained in class):
  TIMSK0 = 2; // enable interrupt
  TCCR0A = 0x02; // return on clear-on-match
  TCCR0B = 0x02; // prescalar @ 8
  OCR0A = 249; // each time tick is 8(OCR0A+1)/20MHz = 1ms exactly

  // Initialize external interrupt INT1 (= pin 1) on rising edge
  DDRD = 0x00; // D.3 = pin 1 is an input
  EICRA = (1<<ISC11) | (1<<ISC10);
  EIMSK = (1<<INT1);

  task1_timer = t1;
  task2_timer = t2;
  flag = 0;

  // Globally enable interrupts
  sei();
```

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	

Example Problem 3

```
while (1)
  {
    if (task1_timer == 0)
    {
      task1_timer = t1; // Before calling task1(), otherwise task1() is called
                        // every 1.1 ms where the 100 micro second delay comes
                        // from the execution of task1()
      task1();
    }

    if (task2_timer == 0) && (flag == 1)
    {
      task2();
      flag = 0;   // As soon as task2() is finished we want to be able
                  // to detect external interrupts
      task2_timer == t2; // After calling task2() since we measure the time that
                         // passes since task2() has finished executing for the
                         // last time
    }
  }

  return 0;
}
```

21

ECE3411 – Fall 2017
Lab3b.

Implementing a Stopwatch

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lab 4b, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

UCONN

Task 1: Simple Stopwatch

Implement a Stopwatch using Timer0 that measures the time down to 1ms resolution.

- Connect a switch to External Interrupt INT1 (PD3)
- Pushing the switch should start the Stopwatch.
- The same switch pushed once again should show the elapsed time on LCD.
- Another button push resets the Stopwatch and makes it ready for another measurement.
- Make sure you debounce the button pushes.

2

Task 2: Improved Stopwatch

Implement a Stopwatch by reading TCNT1 of Timer1 to measure the time down to 1ms resolution. Use Timer0 to introduce Polling Delay for Switch Debouncing.

- Connect a switch to External Interrupt INT1 (PD3)
- Pushing the switch should start the Stopwatch.
- The same switch pushed once again should show the elapsed time on LCD.
- Another button push resets the Stopwatch and makes it ready for another measurement.
- Make sure you debounce the button pushes.

ECE3411 – Fall 2017
Lecture 3c.

Review Session

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lecture 4b, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

UCONN

Hardware Registers of a Port

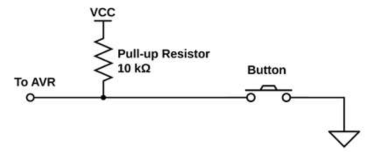
Each Port on the Mega AVRs has three hardware registers associated to it:

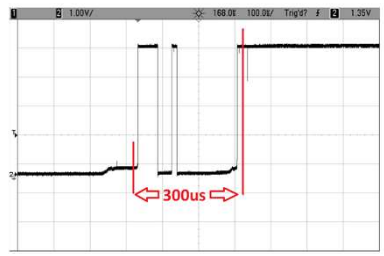
- **DDRx** : *Data-Direction Register for Port x*
  - Controls whether each pin is configured for input (0) or output (1).
  - To enable a pin as output, a '1' is written to that bit in DDRx.
  - By default, all pins are initialized as inputs (DDRx = 0x00).
- **PORTx** : *Port x Data Register*
  - Sets an output pin to logic HIGH (1) or LOW (0).
  - E.g. writing a '1' to a bit position in PORT register will produce logic HIGH at that pin & vice versa.
- **PINx** : *Port x Input Pins Address*
  - Used to read the logic values of each pin that's configured as input.
  - E.g. a value '0' of a bit of PIN register indicates a low voltage at that pin & vice versa.

2

Debouncing of Bouncing Signals

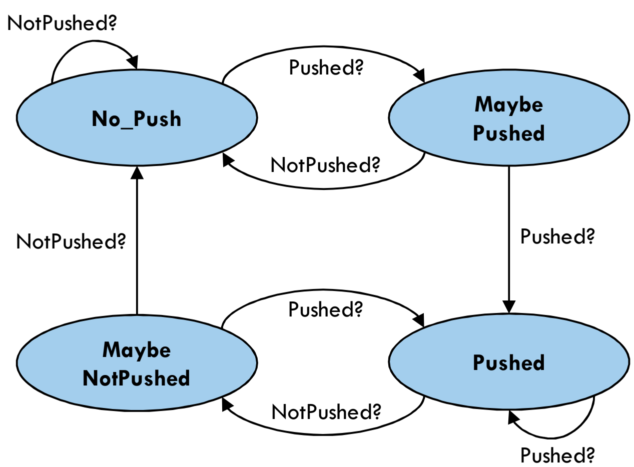
- A button push results in a bouncy transition
  - Due to physical limitations of the contact surfaces
  - Bouncing is often very fast → orders of few us to ms
- Debouncing in software
  - Key idea: Read → Wait → Verify
  - Wait time needs to be carefully controlled
  - E.g. wait time should be at least 300us for this example.





3

Software Debouncing State Machine



4

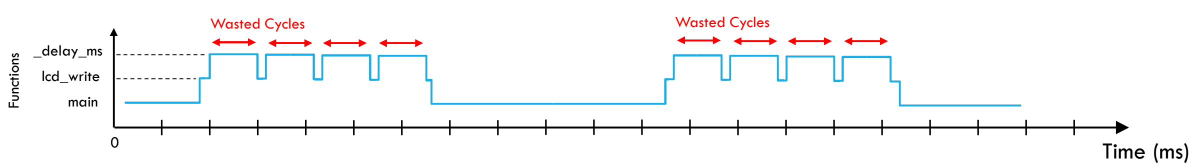
LCD Data Write (4-bit Mode)

```
void LcdDataWrite(uint8_t da)
{
        // First send higher 4-bits
        DATA_PORT = (DATA_PORT & 0xf0) | (da >> 4);         //give the higher half of cm to DATA_PORT
 ①      CTRL_PORT |= (1<<RS);                               //setting RS=1 to choose the data register
        CTRL_PORT |= (1<<ENABLE);                           //setting ENABLE=1

        _delay_ms(1);                                       // allow the LCD controller to successfully read command in
 ②      CTRL_PORT &= ~(1<<ENABLE);                          // Setting ENABLE=0
        _delay_ms(1);                                       // allow long enough delay

        // Send lower 4-bits
        DATA_PORT = (DATA_PORT & 0xf0) | (da & 0x0f);       //give the lower half of cm to DATA_PORT
 ③      CTRL_PORT |= (1<<RS);                               //setting RS=1 to choose the data register
        CTRL_PORT |= (1<<ENABLE);                           //setting ENABLE=1

        _delay_ms(1);                                       // allow the LCD controller to successfully read command in
 ④      CTRL_PORT &= ~(1<<ENABLE);                          // Setting ENABLE=0
        _delay_ms(1);                                       // allow long enough delay
}
```

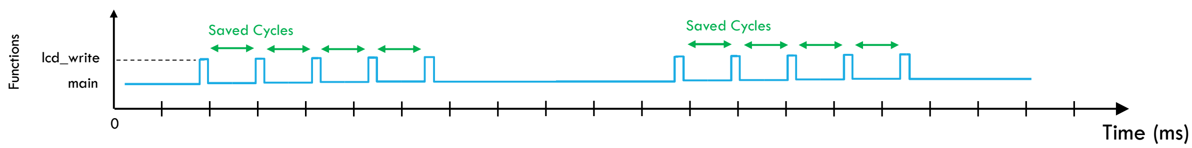
5

Blocking vs. Non Blocking LCD Write Timing

Blocking Writes:



Non-Blocking Writes:



Interrupts & ISRs

A few questions:

- Who calls the ISR?
- Can you "pass" a variable to an ISR?
- What is the return value of an ISR?
- How does the AVR know where to find the code for the corresponding ISR?

Interrupts & ISRs

- Who calls the ISR?
  - The hardware!
- Can you "pass" a variable to an ISR?
  - No! The variable must be globally defined.
- What is the return value of an ISR?
  - Nothing! However, it can store some value in a global variable.
- How does the AVR know where to find the code for the corresponding ISR?
  - Through the Interrupt Vector Table.

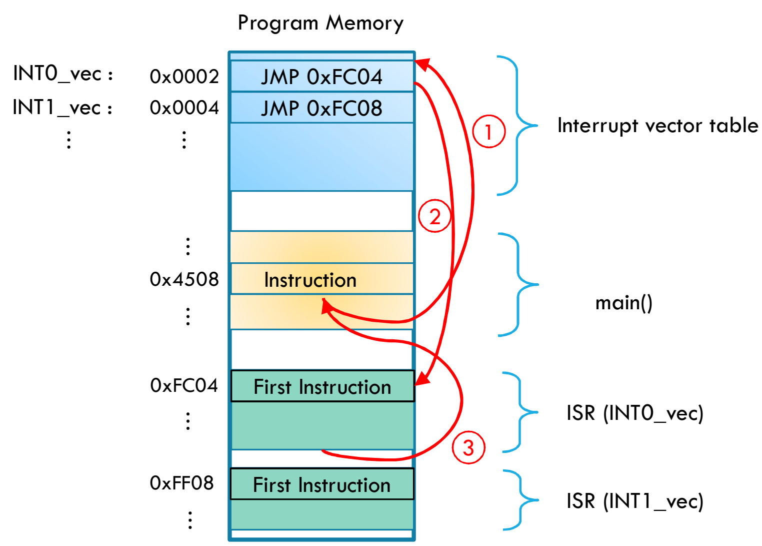
ATmega328P Interrupt Vector Table

- The AVR knows what type of interrupt has occurred.
- It jumps to the program address specified in Interrupt Vector Table.
  - E.g. Address 0x0002 for INT0
- There it sees another Jump instruction which takes it to the ISR code.

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete

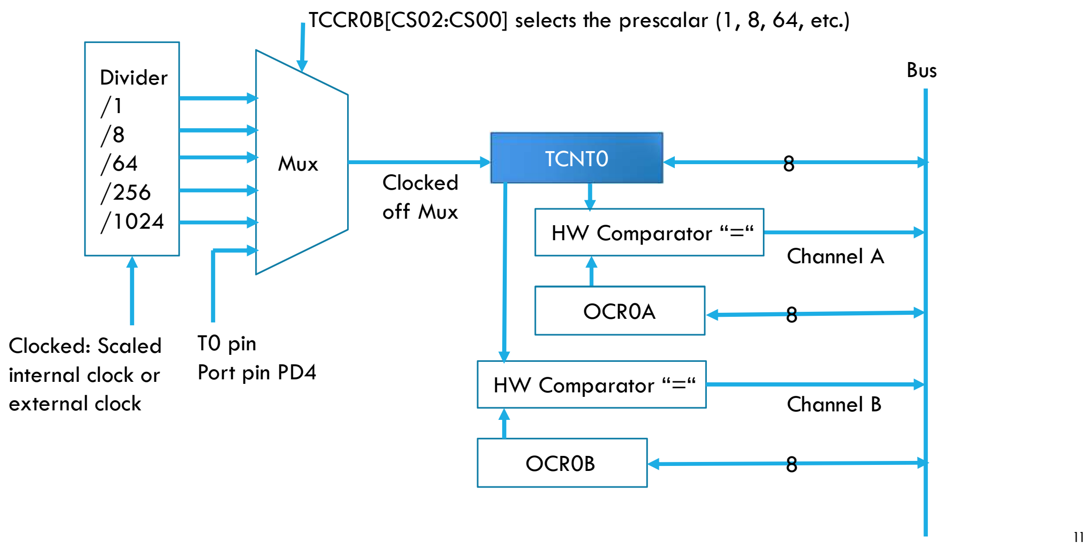
9

Execution of an ISR



10

Timer 0



11

Timer 0 Modes of Operation

- Normal Mode
  - Timer counts up from 0
  - Timer overflows at 0xFF (i.e. 255)
  - Interrupt can be generated upon Overflow
- CTC Mode
  - OCR0A is loaded with some value between 0 to 255
  - Timer counts up from 0
  - A compare match (kind of an overflow) occurs when TCNT0 = OCR0A
  - Interrupt can be generated upon Compare Match

12

Timer 0 Mode Selection

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[(1)(2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

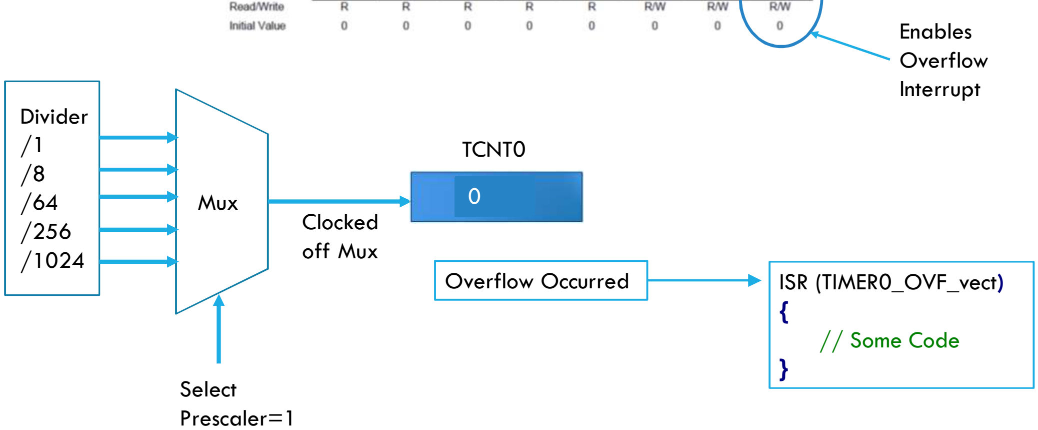
Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

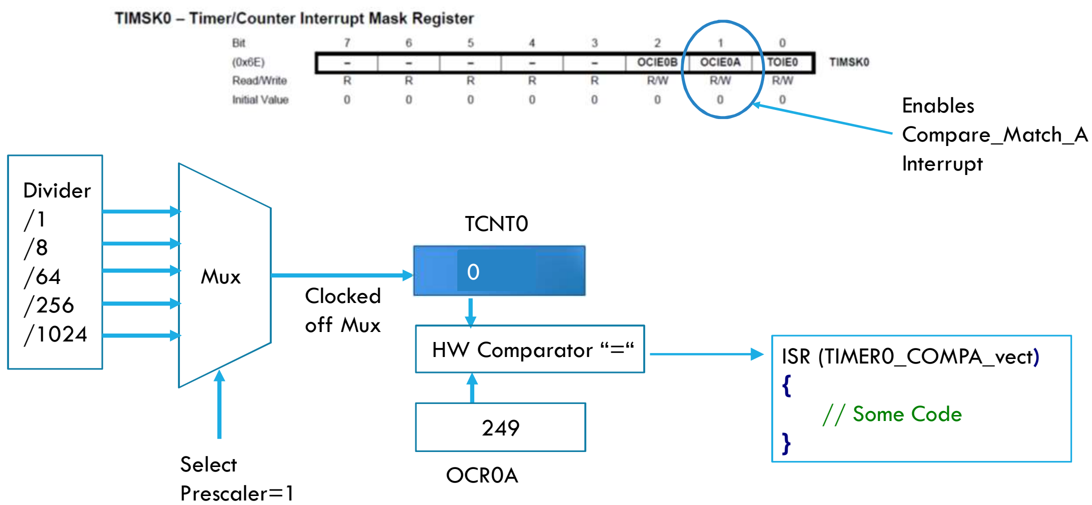
Timer 0 Overflow Interrupt

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Timer 0 Compare Match Interrupt



Timer 1 Modes of Operation

- Normal Mode
  - Timer counts up from 0
  - Timer overflows at 0xFFFF (i.e. 65535)
  - Interrupt can be generated upon Overflow
- CTC Mode
  - OCR1A is loaded with some value between 0 to 65535
  - Timer counts up from 0
  - A compare match (kind of an overflow) occurs when TCNT1 = OCR1A
  - Interrupt can be generated upon Compare Match

Timer 1 Mode Selection

Table 15-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

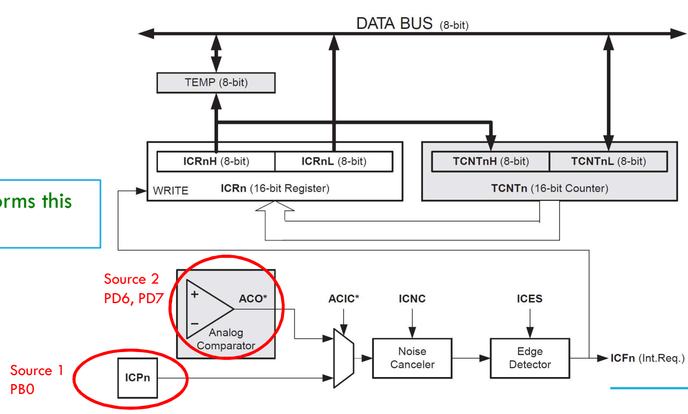
Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

17

Timer 1 Input Capture Interrupt

// Hardware performs this
ICR1 = TCNT1;




```
ISR (TIMER1_CAPT_vect)
{
    // Some Code
}
```

ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3	2	1	0	
0x30 (0x50)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

18

External Interrupts

- External Interrupts INT0 & INT1
  - Can detect any logic change in input pins PD2 and PD3 respectively
  - Can also be configured to trigger by a falling or rising edge
  - INT0 has the highest priority among all interrupts, then INT1and so on…
- Pin Change Interrupts PCINT23..0
  - The pin change interrupt PCI0 will trigger if any enabled PCINT7..0 pin toggles
  - The pin change interrupt PCI1 will trigger if any enabled PCINT14..8 pin toggles
  - The pin change interrupt PCI2 will trigger if any enabled PCINT23..16 pin toggles

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2

Configuring INT1

EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Configuring Pin Change Interrupts

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

Corresponding Pins:
PB0, PB1, PB2

PCMSK1 – Pin Change Mask Register 1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

ECE3411 – Fall 2017

Lab3c (= Lab3b continued).

Implementing a Stopwatch

Marten van Dijk
Department of Electrical & Computer Engineering
University of Connecticut
Email: marten.van_dijk@uconn.edu

Copied from Lab 4c, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

UCONN

Task: Accurate Stopwatch

Implement the Stopwatch using Timer1 and Capture Interrupt to measure the time accurately down to 1ms resolution. Use Timer0 to introduce Polling Delay for Switch Debouncing.

- Connect a switch to External Interrupt INT1 (PD3)
- Pushing the switch should start the Stopwatch.
- The same switch pushed once again should show the elapsed time on LCD.
- Another button push resets the Stopwatch and makes it ready for another measurement.
- Make sure you debounce the button pushes.

2

Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2017

Problem Set P3

There are 3 questions in this problem set. Answer each question according to the instructions given in at least 3 sentences on own words.

If you find a question ambiguous, be sure to write down any assumptions you make.
Be neat and legible. If we can't understand your answer, we can't give you credit!

Any form of communication with other students is considered cheating and will merit an F as final grade in the course.

SUBMIT YOUR ANSWERS IN A HARDCOPY FORMAT.

Do not write in the box below

1 (x/10)	2 (x/30)	3 (x/50)	3 (x/10)	Total (xx/100)

Name:

Student ID:

1. [10 points]: Answer the following questions:
(Encircle the correct answer for Multiple Choice Questions)

A. What register stores the values of external interrupt flags and will trigger an external interrupt if the I-bit in SREG is set?

(a) EICRA
(b) EIMSK
(c) EIFR
(d) PCIFR

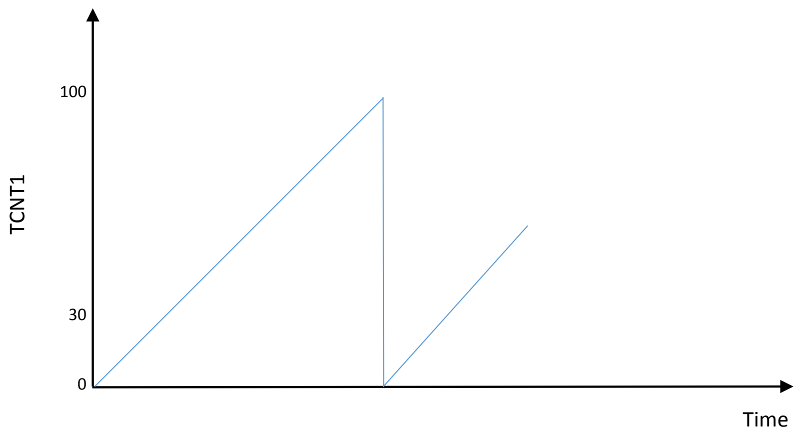
B. Given below is an ISR for external interrupt INT1 that toggles a LED whenever a switch connected to INT1 pin is pushed.

```
/* External Interrupt INT1 ISR. Interrupt triggered at Falling Edge */
ISR(INT1_vect)
{
    EIMSK &= ˜(1<<INT1);     // Disable External Interrupt INT1
    PORTB ˆ= (1<<PORTB5);    // Toggle a LED

    /* Enable External Interrupt INT1 again later in main() code */
}
```

What is the purpose of disabling INT1 in the ISR? What could go wrong if INT1 is not disabled immediately?

2. [30 points]: Given below is an Interrupt Service Routine called for Timer1 Compare Match A. Assume no prescalar is set for Timer 1 and `OCR1A` is initially set to 100.

```
ISR(TIMER1_COMPA_vect)
{
   OCR1A = 30;
}
```



a. Indicate in the time axis given above where the ISR is being called?

b. Indicate in the time axis when `OCR1A = 30` in the ISR will be executed?

c. For what value of `TCNT1` will the ISR be executed next?

Initials:

3. [50 points]: Assume a clock frequency of $f_{clk} = 20\text{MHz}$ and the following initialization:

```
DDRD   = 0x10;
OCR1A  = 39062;
OCR1B  = 13020;
TCCR1A = 0b00110011;
TCCR1B = 0b00011101;
```

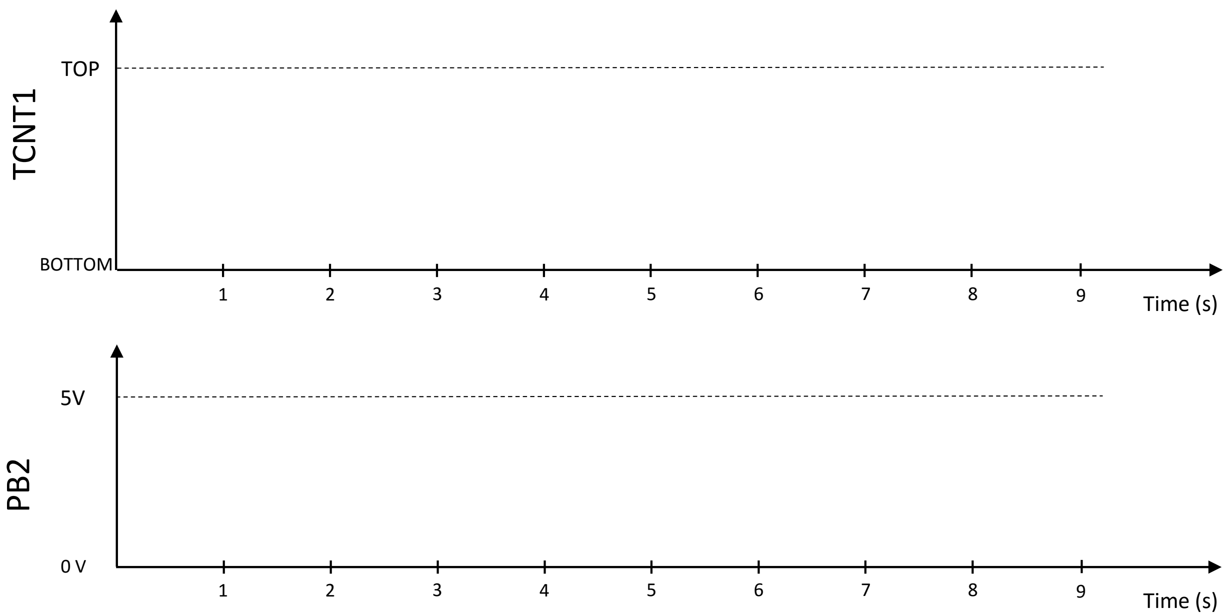
Answer the following questions:

a. In which mode is `Timer1` running?

b. What is the numerical value of 'TOP' for `Timer1` in this mode?

c. How much time (in seconds) does it take for `Timer1` to complete one full cycle, i.e. going from BOTTOM → TOP → BOTTOM? Be as accurate as possible in your calculations.

d. Starting from the moment of `Timer1`'s initialization, draw the waveforms of the `TCNT1` register value and the pin PB2 value w.r.t. time. Please draw the waveform strictly according to the timing scale shown on X-axis, otherwise no credit will be given.



e. In each full cycle of `Timer1`:

- For how much time (in seconds) is PB2 low? Be as accurate as possible in your calculations.

- For how much time (in seconds) is PB2 high? Be as accurate as possible in your calculations.

Initials:

4. [10 points]: Can you shortly describe what you have learned and feel confident about using in the future?

Initials:

End of Problem Set

Please double check that you wrote your name on the front of the quiz.

Initials:

Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2017

Problem Set A3

There are 3 questions in this problem set. Answer each question according to the instructions given in at least 3 sentences on own words.

If you find a question ambiguous, be sure to write down any assumptions you make.
Be neat and legible. If we can't understand your answer, we can't give you credit!

Any form of communication with other students is considered cheating and will merit an F as final grade in the course.

SUBMIT YOUR ANSWERS IN A HARDCOPY FORMAT.

Do not write in the box below

1-A (x/10)	1-B (x/30)	1-C (x/10)	1-D (x/10)	2 (x/20)	3 (x/20)	Total (xx/100)

Name:

Student ID:

1. [60 points]: A colleague wants your help in executing a particular task for which you need to write a code such that a task() is executed as soon as the following events occur (See Figure 1).

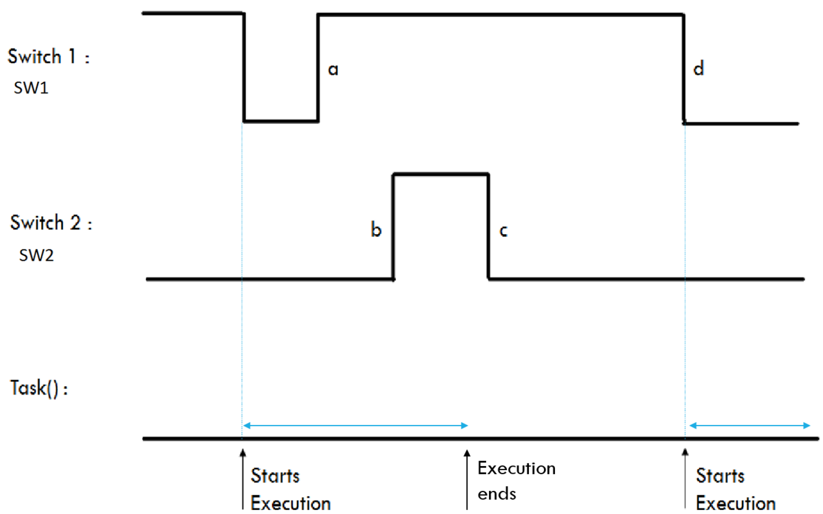


Figure 1: Timing Diagram.

a - There is a rising edge at SW1

b, c - SW2 toggles twice after event (a)

d - After events b and c occur there is a falling edge at SW1 and task() is not running at that very moment. Note that if task() is executing at that moment then the MCU needs to wait for event (a) to occur again.

The switches SW1 and SW2 are connected to PB1 and PD3 of ATmega328P respectively, as shown in the Figure 3. The clock frequency ($clk_{I/O}$) is 16MHz.

Implement this system by answering the short questions and filling in the gaps in the code layout given below. Notice that you are **not allowed** to use any software counter or `_delay_ms()/_delay_us()` routines.

Initials:

The following code layout needs to be used.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <avr/interrupt.h>
// Flag Variables
volatile uint8_t taskflag;

// Declare more variables as required in sub problem [B]

void initialize_all(void)
{
    //To be filled in sub problem [A]
}

ISR( //To be filled in sub problem [A]  )
{
   //To be filled in sub problem [B]
}


ISR( //To be filled in sub problem [A]  )
{
   //To be filled in sub problem [B]
}

/* Main Function */
int main(void)
{
    initialize_all();     // Initialize everything
    sei();                // Enable Global Interrupts
    taskflag = 0;

    while(1)
    {
        if(taskflag == 1)
        {
           // In sub problem [B] you will need to decide the order of
           execution of the statements 1) taskflag = 0; and 2) task();

        }
    }

} /* End of main() */
```

Initials:

A. Initialization: (10 points)

Complete the function `initialize_all(void)` as instructed below:

```
/* Initialization function */
void initialize_all(void)
{

    // Program only the necessary control register and ports

} /* End of initialize_all() */
```

Give the names of the interrupt vector used for interpreting the input from the 2 connected switches.(see figure 2 provided at the end of the quiz)

(a) ISR(C)

(b) ISR(D)

C -

D -

Initials:

B. Interrupt ISR: (30 points)
Complete the function `ISR( C )` and `ISR( D )` and declare the necessary variables. **Do not** execute the task() in the ISR, instead set the taskflag value accordingly. [Hint : It would be helpful to use a FSM that tracks the event sequence.]

```
  // Declarations
```

```
ISR( C )
{

   // Code

}
```

Initials:

```
ISR( D )
{

    //Code

}
```

Write the code for the while loop in the main function

```
while(1)
{
   if(taskflag == 1)
   {

      //Complete code here

   }
}
```

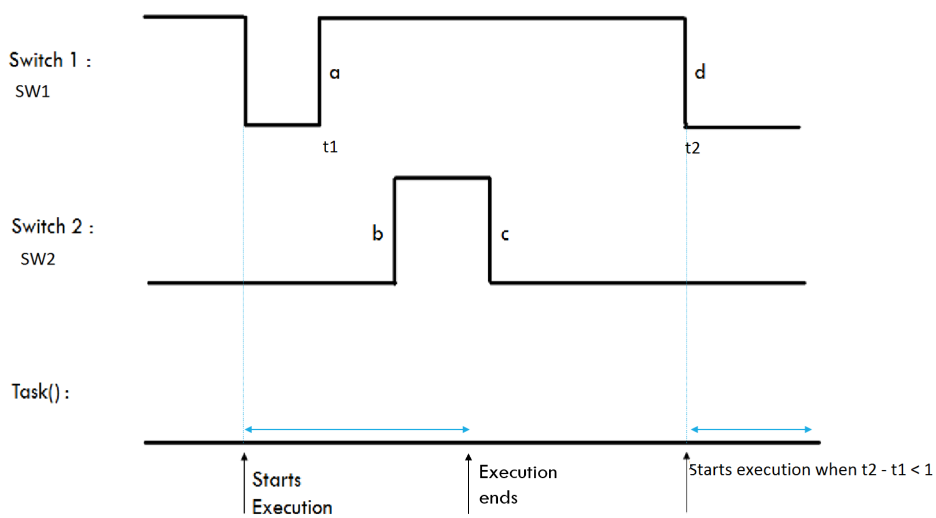
Initials:

C. (10 points) Extend the system to implement an additional requirement

Consider the timing diagram given below. Suppose event (a) occurs at time t1 and event (d) occurs at time t2.

You are asked to change the code such that when event (d) happens you also check whether the time $t2 - t1$ is less than 1 second. If this is not the case the task() will not be executed.

Describe in words what changes need to be included in the code. [Hint : Think about how you can measure the time between the events a and b, the extra declarations required etc.]



Initials:

D. (10 points) Suppose the switch SW2 is connected to PB7. Explain in words what changes you need to incorporate in the code?

2. [20 points]: You need to design a system such that whenever a certain internal condition (checked by the function '`is_condition_true()`') is true, a small function executes **atomically** and with **the highest priority** over any other code in your software. One way to do it is by using External Interrupt INT0 ISR (INT0 is at pin `PD2`).
Complete the "`initialize_all()`" and "`main()`" functions such that INT0 ISR gets triggered everytime the function "`is_condition_true()`" returns true. State clearly if yo need to make any hardware connections between any two pins etc.
You may use External Interrupts data sheet provided at the end of this booklet.

State hardware connections (if any):

```
//------------------------------------------------------------------------
/* Initialization function */
void initialize_all(void)
{
    // Configure INT0 and perform any other initializations here.









    // Enable Global Interrupts here.


} /* End of initialize_all() */
//------------------------------------------------------------------------
/* External Interrupt INT0 ISR */
ISR(INT0_vect)
{
    /* Function that needs to be executed atomically */
    Some_Atomic_Code();

    /* Any other code that you want to include in ISR goes here. */




}
//------------------------------------------------------------------------
```

Initials:

```
/* Main Function */
int main(void)
{
    // Initialize everything
    initialize_all();

    while(1)
    {
        if( is_condition_true() )
        {
            /* Your code to trigger INT0 ISR goes here */
















        }
    }

} /* End of main() */

//----------------------------------------------------------------------
```

Initials:

3. [20 points]: Given that the clock frequency ($clk_{I/O}$) of ATmega328P is 16MHz, implement the finite state machine (FSM) shown in Figure 2. The state transitions are made whenever a button connected to INT0 pin (i.e. PD2) is pushed and a **Falling Edge** is detected at INT0. Each state produces an output signal at PB2, and the output specifications of the states are as follows:

State_A: PB2 stays at logic LOW level.

State_B: A non-inverting 1kHz PWM signal with 30% duty cycle is generated at PB2.

State_C: A non-inverting 2kHz PWM signal with 70% duty cycle is generated at PB2.

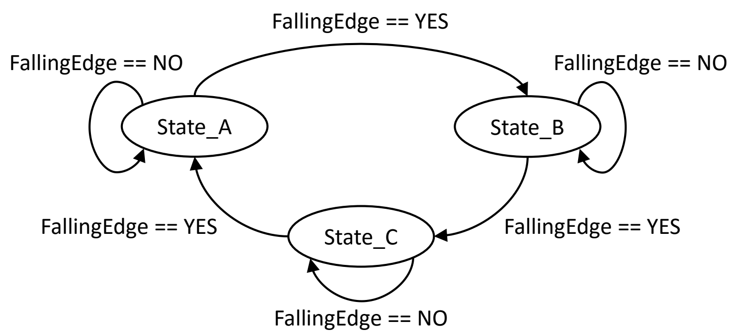


Figure 2: State Transition Diagram of the FSM

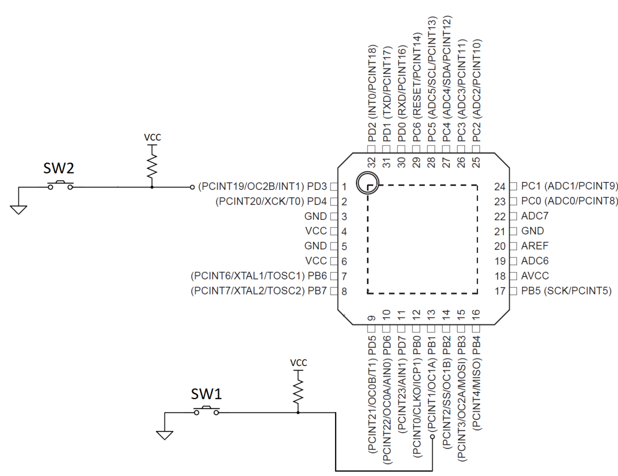


Figure 3: ATmega328P Hardware Configuration.

Assuming that the push button does not need debouncing, complete the following code segments.

Initials:

The following code snippet provides the necessary includes, declarations, definitions and a basic layout.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <inttypes.h>
#include <avr/interrupt.h>

// For State Machine
#define State_A    1
#define State_B    2
#define State_C    3
volatile uint8_t System_State;

// For PWM
volatile uint16_t time_period;
volatile uint16_t duty_cycle;

// Define any other variables here



//-----------------------------------------------------------------------

/* Triggers at Falling Edge on PD2 */
ISR(INT0_vect)
{
    // Calls state transition function
    make_state_transition();
}
//-----------------------------------------------------------------------

// Timer 1 Compare Match A ISR (TCNT1 = OCR1A)
ISR (TIMER1_COMPA_vect)
{
    OCR1A = time_period;   // Update PWM time period
    OCR1B = duty_cycle;    // Update PWM duty cycle
}
//-----------------------------------------------------------------------

/* Main Function */
int main(void)
{
    initialize_all();     // Initialize everything
    sei();                // Enable Global Interrupts
    while(1);             // Nothing to do.

} /* End of main() */
```

Initials:

A. Initialization: (10 points)

Complete the function `initialize_all(void)` as instructed below:

```
/* Initialization function */
void initialize_all(void)
{
    // Initializing the state variable
    System_State = State_A;

    /* Configure PB2 here */



    /* Configure INT0 here */



    /* Configure Timer 1 here */



    /* Any other initializations here if needed */



} /* End of initialize_all() */
```

Initials:

B. State Transition Function Implementation: (10 points)
Write the function `make_state_transition()` to implement the FSM.

```
/* State transition function called by INT0 ISR */
void make_state_transition()
{

} /* end of make_state_transition() */
```

Initials:

End of Problem Set

Please double check that you wrote your name on the front of the quiz.

Initials:

Department of Electrical and Computing Engineering

UNIVERSITY OF CONNECTICUT

ECE 3411 Microprocessor Application Lab: Fall 2017

Independent LAB3

There are 2 independent lab questions in LAB3.

You may not discuss independent labs in any way, shape, or form with anyone else and you are not allowed to lookup solutions from other sources.

Any form of communication with other students or looking up solutions is considered cheating and will merit an F as final grade in the course.

Name:

Student ID:

1. [Pass/Fail points]: In this task, we are going to design a Stopwatch (1ms resolution) for measuring the total time and the individual lap times of a car racer. A detailed breakdown of the task is given below, whereas the detailed timing diagram of the stopwatch is shown in Figure 1.
Notice that for this task, `_delay_ms()`/`_delay_us()` function calls are not allowed (except for the ones already present in `lcd_lib.c`).

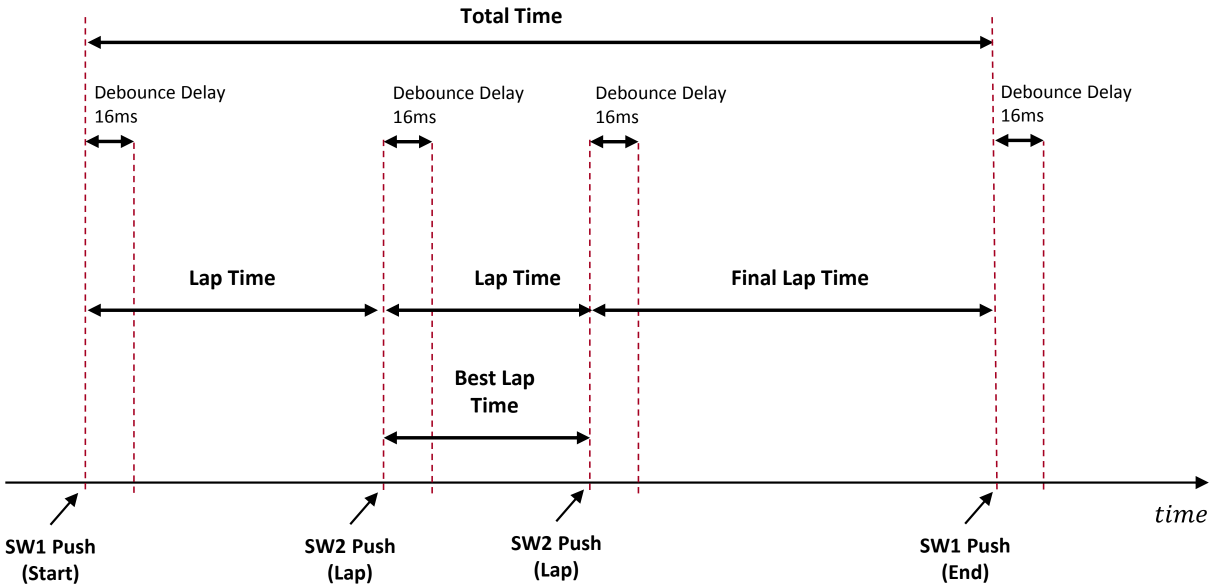


Figure 1: Stopwatch Timing Diagram.

a. Using ***Pin Change Interrupts***, read two push switches SW1 and SW1 connected to PB1 and PB7 respectively, and design a basic system that does the following:

- When SW1 is pressed, LED1 turns on. This shows the start of the race.
- While LED1 is on, if SW2 is pressed then LED2 toggles. This shows completion of a lap.
- Finally if SW1 is pressed again, both LEDs turn off. This shows end of the race.

Use `Timer0` to count a debounce delay of 16ms for SW1 and SW2.

b. Extend Task(a) and use `Timer1` to implement the following basic stopwatch:

- When SW1 is pressed (i.e. start of the race), start `Timer1` to count the number of milliseconds. You may want to use a software counter to keep track of long time intervals.
- When SW1 is pressed again (i.e. end of the race), record the current time. This shows end of the race.
- Print the total elapsed time (in milliseconds) on the first row of the LCD.

Make sure that you debounce SW1 with a 16ms delay using `Timer0`, yet start/capture `Timer1` at the very moment of the button push (as shown in Figure 1) instead of 16ms later.
Hint: You can read the current value of Timer1 by reading `TCNT1` register anywhere in the code.

Initials:

c. Extend Task(b) to implement the following functionality of the stopwatch:

- If SW2 is pressed while the stopwatch is counting (i.e. during the race), record the current time. This shows completion of a lap. Notice that `Timer1` continues to count the total race time.
- Print on the second row of the LCD the time elapsed (in milliseconds) between this SW2 push and the previous most recent button push event. This shows the lap time of the racer. E.g. if SW2 is pressed for the first time after the start of the race then print the total time elapsed since SW1 push. Otherwise print the time elapsed since last SW2 push (as shown in Figure 1).

Make sure that you debounce SW2 with a 16ms delay using `Timer0`, yet capture `Timer1` at the very moment of the button push (as shown in Figure 1) instead of 16ms later.

d. Extend Task(c) to complete the stopwatch implementation as follows:

- Finally when SW1 is pressed again (i.e. at the end of the race), record the final lap time which is the time since the last SW2 push (as shown in Figure 1).
- Print the total race time on first row and the best lap time (i.e. the smallest) among all the recorded lap times on the second row of the LCD.

Hint: Determine and record the smallest lap time at each SW2 push.

Initials:

2. [Pass/Fail points]: In this task, we are going to implement a simplified version of Morse Codes for a few English alphabets shown in Table 1. In order to produce an alphabet, the following two conditions must be met:

(a) A particular sequence of SW1 and SW2 button pushes as shown in Table 1.

(b) The push sequence must be completed within a 2 seconds window (starting from the first push).

Table 1: Simplified Morse Code Table.

Alphabet	**Button Push Sequence within 2 seconds window.**
A	SW1, SW2
B	SW2, SW1, SW1, SW1
C	SW2, SW1, SW2, SW1
D	SW2, SW1, SW1
Invalid	Any other sequence.

The clock frequency ($clk_{I/O}$) is 16MHz.
The switches SW1 and SW2 are connected to PB1 and PB7 of ATmega328P respectively, as shown in the Figure 2. Both SW1 and SW2 need a **debouncing delay** of 4ms.

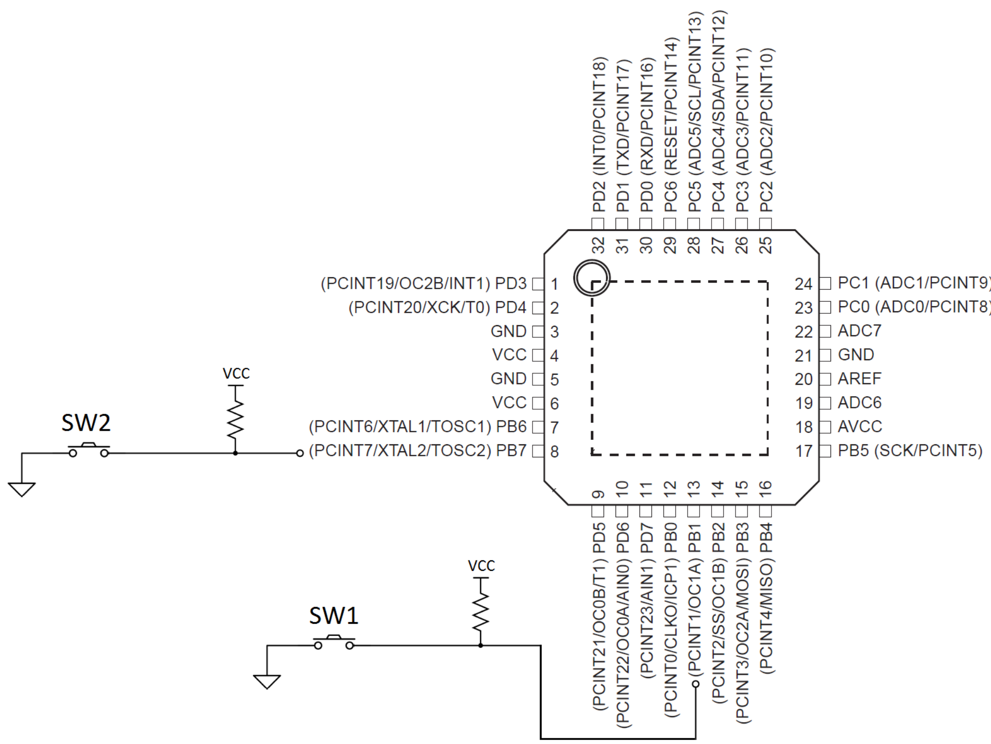


Figure 2: ATmega328P Hardware Configuration.

Implement this system by filling in the gaps in the code layout given below.
Notice that you are **not allowed** to use `_delay_ms()`/`_delay_us()` routines.

The following code snippet provides the necessary includes, declarations and definitions.

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <inttypes.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include <string.h>
#include "lcd_lib.h"

#define SW1_PRESSED (~PINB & (1<<PINB1))
#define SW2_PRESSED (~PINB & (1<<PINB7))

// Flag Variables
volatile uint8_t DebounceFlag1;
volatile uint8_t DebounceFlag2;
volatile uint8_t index1;
volatile uint8_t index2;

// Push Sequence Encoding
volatile uint8_t encodings[4][5] =
    {
        {1, 2, 0, 0, 0},    // A's encoding is accessed as encodings[0]
        {2, 1, 1, 1, 0},    // B's encoding is accessed as encodings[1]
        {2, 1, 2, 1, 0},    // C's encoding is accessed as encodings[2]
        {2, 1, 1, 0, 0}     // D's encoding is accessed as encodings[3]
    };

// Index to Character Mapping
volatile uint8_t mapping[4] = {'A', 'B', 'C', 'D'};

// Save the Button Pushes in this array
volatile uint8_t sequence[5];

//-----------------------------------------------------------------------

/* Main Function */
int main(void)
{
    initialize_all();     // Initialize everything
    sei();                // Enable Global Interrupts
    while(1);             // Nothing to do.

} /* End of main() */
//-----------------------------------------------------------------------
```

Initials:

A. Initialization:

Complete the function `initialize_all(void)` as instructed below:

```
/* Initialization function */
void initialize_all(void)
{
    // Initializing the LCD.
    initialize_LCD();
    LCDcursorOFF();
    LCDclr();

    // Initializing the flag variables
    DebounceFlag1 = DebounceFlag2 = 0;
    index1 = index2 = 0;

    // Enable Pin Change Interrupts for PB1 and PB7 here





    // Setup Timer0 in CTC mode to generate Compare Match Interrupt A every 4ms
    // Set Timer0 Prescaler in 'start_timer0()' function on the next page.









    // Setup Timer1 in CTC mode to generate Compare Match Interrupt A every 2s










} /* End of initialize_all() */
```

Initials:

B. Timer0 Prescaler & Pin Change Interrupt ISR:

Complete the function `start_timer0()` and `ISR(PCINT0_vect)` as instructed below:

```
/* Starts Timer0 */
void start_timer0()
{
    // Select and set appropriate prescaler for Timer0 here



}

//----------------------------------------------------------------------

/* Stops Timer0 */
void stop_timer0()
{
    TCCR0B = 0x00;          // Prescaler = NONE
    TCNT0 = 0;              // Resets the timer
}

//----------------------------------------------------------------------

/* Pin Change Interrupt 0 ISR */
ISR(PCINT0_vect)
{
    // Disable the Pin Change Interrupt 0 here


    // Update any flags etc.











    // Start Timer0 to count Debounce Delay
    start_timer0();

} /* end of ISR(PCINT0_vect) */
```

Initials:

C. Timer0 Compare Match ISR:

Complete `ISR(TIMER0_COMPA_vect)` as instructed below:

```
/* Timer0 Compare Match A ISR */
ISR(TIMER0_COMPA_vect)
{
    // Stopping Timer0
    stop_timer0();

    // Read and record the button push in ‘sequence’ array





    // Re-enable Pin Change Interrupt
    PCICR |= (1<<PCIE0);

} /* end of ISR(TIMER0_COMPA_vect) */
```

Initials:

D. Timer1 Compare Match ISR:

Complete `ISR(TIMER1_COMPA_vect)` as instructed below:

```
/* Timer1 Compare Match A ISR */
ISR (TIMER1_COMPA_vect)
{
    // Print the Alphabet corresponding to the received push sequence on LCD
    // Print 'I' if an invalid sequence is received.




    // Clear the received sequence buffer
    for(index2=0; index2<4; index2++)
        sequence[index2] = 0;

    // Reset index2
    index2 = 0;

} /* end of ISR(TIMER1_COMPA_vect) */
// ------------------------------------------------------------------------ //
```

Initials:

End of Independent LAB3

Initials: