Timing of Events

Suppose that we want produce a pulse on a digital line that was exactly 500 ms in length?

• What would the code look like?
Timing of Events

// Assume it is pin 0 of port B

PORTB = PORTB | 1;
delay_ms(500);
PORTB = PORTB & ~1;
Timing of Events

// Assume it is pin 0 of port B

PORTB = PORTB | 1;
delay_ms(500);
PORTB = PORTB & ~1;

This will work, but why is it undesirable?
Timing of Events

This will work, but why is it undesirable?

delay_ms() is implemented by using a for() loop

• The microcontroller can’t do anything else while it is looping
• Have to loop a precise number of times (not always easy to do)
Timing of Events: Another Example

Suppose we would want to measure the width of a pulse. How would we implement this?
Timing of Events: Another Example

How would we implement this?

// Wait for pin to go high
while (PINB & 0x1 == 0) {};

// Now count until it goes low
for (counter = 0; PINB & 0x1; ++counter) {
    delay_ms(1);
}

// Now: counter is the width of
// of the pulse in ms
Timing of Events: Another Example

Again: the program cannot be doing anything else while it is waiting
Counter/Timers in the Mega2560

The mega2560 includes six counter/timer devices in hardware.

These can:

- Be used to count the number of events that have occurred (either external or internal)
- Act as a clock
Timer 0

- Two possible input sources:
  - Pin T0 (PD4)
  - System clock
    - Potentially divided by a “prescaler”
- 8-bit counter
- When the counter turns over from 0xFF to 0x0, an interrupt (an event) can be generated (more on this later)
**Generic Timer Implementation**

- **Prescaler:** divides clock frequency

![Diagram of Generic Timer Implementation](image)

- System Clock $f$ (ticks/sec)
- Prescaler (ticks/tick)
- $f$, $f/2$, $f/(2^K)$
- External input (some pin)
- Signal Selector (multiplexer)
- Counter Configuration
- Counter
- Data bus
Generic Timer Implementation

- **Prescaler**: divides clock frequency
- **Multiplexer**: selects one of the inputs to drive the counter

![Diagram of a generic timer implementation]

- System Clock $f$ (ticks/sec) → Prescaler (ticks/tick) → Signal Selector (multiplexer) → Counter Configuration → Counter
- Prescaler: $f$, $f/2$, $f/(2^k)$, external input (some pin)
- Counting up/down via data bus
Generic Timer Implementation

- **Prescaler:** divides clock frequency
- **Multiplexer:** selects one of the inputs to drive the counter
- **Counter:** increment on low-to-high transition of its input
Timer 0 (and Timer 1)

Possible prescalers:
- 8
- 64
- 256
- 1024
Timing Example

Suppose:
• $f=16\text{MHz}$ clock
• Prescaler of 1024
• We wait for the timer to count from 0 to 156

How long does this take?
Timer 0 Example

\[
delay = \frac{1024 \times 156}{16,000,000} = 9948 \ \mu s \approx 10 \ ms
\]
Timer 0 Code Example

timer0_config(TIMER0_PRE_1024);   // Init: Prescale by 1024

timer0_set(0);       // Set the counter to 0

<Do something else for a while>
while(timer0_read() < 156) {
    <Do something while waiting>
};

// Break out of while loop after ~10 ms

See Atmel HOWTO for example code (timer_demo2.c)
Timer 0 Example

Advantage over delay_ms():

• Can do other things while waiting
• Timing is much more precise
  – We no longer rely on a specific number of instructions to be executed
Timer 0 Example

One caution:
• “something else” cannot take very much time

(we have a solution for this – coming soon!)
Next Example

How do we time a delay of 100 usecs?
Next Example

How do we time a delay of 100 usecs?

\[
\text{counts} \times \text{prescale} = .0001 \times \text{clock freq}
\]

\[
= .0001 \times 16000000
\]

\[
= 1600
\]
Next Example

How do we time a delay of 100 usecs?

\[ \text{counts} \times \text{prescale} = 0.0001 \times \text{clock \_ freq} \]

\[ = 0.0001 \times 160000000 \]

\[ = 1600 \]

\[ 200 \times 8 = 1600 \]

OR

\[ 25 \times 64 = 1600 \]
Timer 0 Code Example

timer0_config(TIMER0_PRE_8); // Init: Prescale by 1024

timer0_set(0); // Set the timer to 0

<Do something else for a while>
while(timer0_read() < 200) {
    <Do something while waiting>
};

// Break out of while loop after ~100 us
Example 3: Timing the Width of a Pulse

- Input: port B, pin 1
- How long is the pin high?
Timing a Pulse Width: Our Original Implementation

// Wait for pin to go high
while(PINB & 0x1 == 0) {};

// Now count until it goes low
for(counter = 0; PINB & 0x1; ++counter) {
    delay_ms(1);
}

// Now: counter is the width of
//   of the pulse in ms
Example: Timing a Pulse Width

// Init: Prescale by 1024
timer0_config(TIMER0_PRE_1024);

// Wait for pin to go high
while(PINB & 0x2 == 0) {
    <Do something while waiting>
};
timer0_set(0);       // Set the timer to 0

while((PINB & 0x2) != 0) {
    <Do something while waiting>
};
pulse_width = timer0_read();
Example: Timing a Pulse Width

What is the “resolution” of pulse_width?
Example: Timing a Pulse Width

What is the “resolution” of pulse_width?

• Each “tock” is:

\[
\text{delay} = \frac{1024}{16,000,000} = 64 \ \mu s
\]
Example: Timing a Pulse Width

So, with pulse_width tocks:

\[
delay = \frac{1024 \times pulse\_width}{16,000,000} = 64 \times pulse\_width \, \mu s
\]
Example: Timing a Pulse Width

// Init: Prescale by 1024
timer0_config(TIMER0_PRE_1024);

// Wait for pin to go high
while(PINB & 0x2 == 0){
  <Do something while waiting>
};
timer0_set(0); // Set the timer to 0

while((PINB & 0x2) != 0) {
  <Do something while waiting>
};
pulse_width = read_timer0();

Note: the longer “something” takes, the larger the possible error in timing
Other Timers Besides Timer 0

Timers 1, 3, 4, 5:
- 16 bit counter
- Prescalers: 1, 8, 64, 256, 1024

Timer 2:
- 8 bit counter
- Prescalers: 1, 8, 32, 64, 128, 256, 1024
Note

See oulib documentation for the list of possible prescalers for the timers
Pulse-Width Modulation in Hardware

- The Atmel Mega processors will perform a wide-range of timing functions in hardware
- This includes the generation of pulse-width modulated signals
- Once configured, your main program need only to set the duty cycle of the PWM signal
Pulse-Width Modulation in Hardware

• Configuration includes:
  – Signal frequency (through the prescalers)
  – Signal polarity (high then low or vice-versa)
  – Resolution for specifying the duty cycle

• Once configured:
  – You need only specify changes to the duty cycle
PWM on the Atmel Mega2560s

Timers 1, 3, 4, 5: each have 3 PWM output channels associated with them (known as A, B, and C)

For our example here:

• Use 10 bits of the 16 available with the counter
• Counter counts from 0 to 1023, and then back to 0
• Output goes high at 0
• Output goes low at specified count
  – Specified by the “output compare” register
Example

For our example, we will use:

• Timer 4, channel A
  – I/O Port H, pin 3
• 10-bit resolution
• Prescaler of 8
Initialization Example (Timer 4)

```c
int main(void) {
    // The timer 4 channel A pin is labeled “OC4A” on the Arduino circuit diagram
    DDRH = 0x8;

    // tocks/sec = 2,000,000/sec (with a 16,000,000 ticks/sec clock)
    timer4_config(TIMER4_PRE_8);

    // Configure for 10-bit PWM
    timer4_output_compare_config(TIMER4_OUTPUTCOMPARE_CONFIG_PWM_F_10);

    // Configure timer 4, channel A for PWM: high then low
    timer4_compare_output_A_mode_set(TIMER16B_COMPARE_OUTPUT_MODE_CLEAR);
    :
    :
```
Use Example

```c
int16_t i;

// Loop forever
while(1) {

    // Slowly increase the duty cycle on channel A
    for(i=0; i < 1024; ++i) {
        timer4_output_compare_A_set(i);
        delay_ms(1);
    }

    // Slowly bring the duty cycle back to zero
    for(i=1023; i > 0; --i) {
        timer4_output_compare_A_set(i);
        delay_ms(1);
    }
}
```
See examples_2560/pwm for more details