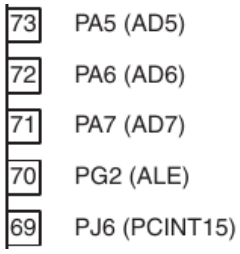
Arduino MEGA Hardware Guide ECE474

This document will help you find your way around the hardware details of the Arduino MEGA board.

Basic Digital I/O

Arduino MEGA has a huge number of I/O pins for your use. The simplest case is to allow your code to set or clear individual bits one at a time. Pins have multiple uses so before your code starts its real work, it has to initialize all I/O pins you intend to use. Specifically let's look at board pins 30-37 ("Port C," see below). Each pin is a pretty complex circuit (see Fig 13-2) but basically, they can be either an output (puts out a high or low voltage) or an input (senses a logic level). One other thing you can configure is to optionally have a pull-up resistor for inputs between the pin and Vcc. You can use this in combination with, for example, an external switch. Connect the switch between the pin and ground and then the internal pull-up resistor supplies current which generates a logic 1 when the switch is open. The five modes each pin can be in are listed in Table 13-1 (page 69).

Setting the pin mode: The chip pin numbers are labeled two ways. At the most concrete level, hardware package pins are sequentially numbered. For the 2560, they are pin1 - pin100. Since there are different packing styles, there are abstract pin numbers Pxn which divide the pins into 8-bit "Ports" labeled with capital letters (see pages 7-8). So for example, (see schematic below) hardware package pin 71 is called PA7 meaning it's the 7th bit of port A. Finally, each pin has a logical name that links it to its function. In the case of pin 71/PA7, this is AD7, the 7th Analog/Digital Conversion input.



Each port has two key registers, the PORT register and the Data Direction Register (DDR). DDR has a bit for each port pin which determines if it is an input (0) or an output (1). When the pin is an output, setting PORT register bits sets the output voltage as high or low. When the pin is set to input (via the DDR), the PORT bit turns on or off the pull-up.

Coding for Direct Register Access [additional background from WPI]
Registers such as the PORT and DDR registers are mapped into the memory space of the microcontroller so we can think of them as memory locations. Like any memory locations, they have an address. So, we can use C pointers to read and write to the registers. For example, For port A, the DDR is DDRA (see 13.4.3 in the Datasheet (page 96)). The address for each register is shown in "Register Description for IO Ports" (page 96) but even simpler, the Arduino IDE pre-defines them for us using the same abbreviations (like DDRA!). For example, a simple app can tell us that DDRA comes from
#define DDRA 0x21

This matches the documentation (13.4.3, p96) (the value in parens at least) and I confirmed it with a little sketch as follows:

```
...
  int value = (int)&DDRA;  // get the address of DDRA, cast to int
  Serial.println(value, HEX);   // prints out "21"
...
```

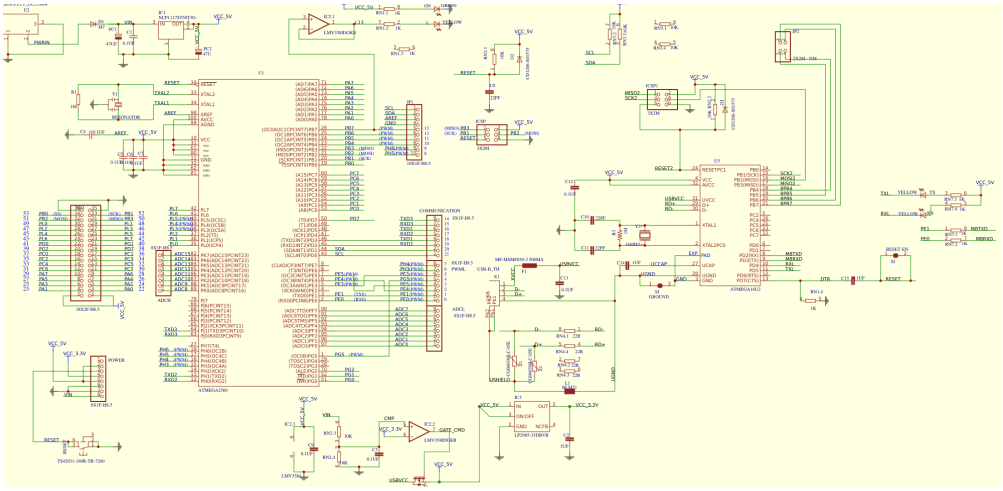
So, now we can use the register names like 16-bit integer variables!
Example 1,
Set bit 5 of Port C to Write (output), and write a 1 to it, delay 10ms and write a 0.

```
...
#define  BIT_5      1<<5;  //  shift "1" 5 places over
…
DDRC |= BIT_5;  // bit-wise OR 1<<5 to set pin to output mode
…
PORTC |= BIT_5; // set bit 5 to "1"
delay(10);
PORTC &= !BIT_5;  // clear bit 5
...
```

Advanced Chip Function Pinouts

Reference links:
[High Res Arduino Mega Board Schematic] [ATmega2560 Datasheet]

Board Schematic:



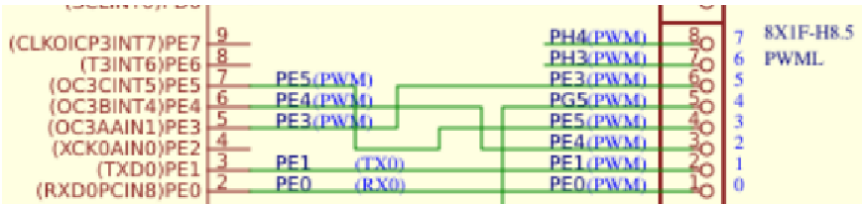
The Arduino MEGA board uses the Atmel ATMega2560 chip at its core. This powerful chip is documented in detail in the [Data Sheet]. However, it can be complex to trace a specific input or output, as described in the document all the way to the end-user pins on the Arduino board.

It's important to realize that each pin has multiple possible functions which your software has to select or configure. But you can only use one function of each pin at a time so watch for conflicts! For example, consider Port E. Table 13-15 (page 82) shows you what the allowed functions for each bit of port E can be but only one of the listed functions for each pin can be selected.

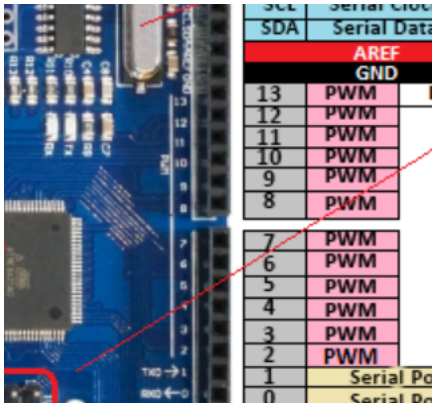
Pinout Example: Timers

We'll study an example focusing on the 16-bit hardware Timer/Counters inside the 2560 (covered in more detail below). We're trying to work our way out from signals described inside the 2560 chip out to the pin numbers on the Arduino MEGA board. To find an internal signal on the Arduino board edge connectors:

1) Identify the pin name in the documentation chapter for your internal device. The 16-bit Timer/Counters are described in Chapter 17 (pages 133-163). For example, (Fig 17-1, p 134) there are three Waveform Generators that drive three pins called OCnA, OCnB, OCnC.
2) Because there are several 16-bit Timer/Counters, we need to identify a specific timer/counter so we replace "n" with the unit number. Let's go with the 3rd Timer/Counter, n=3.
3) Going back to the 2560 pinout on page 2, we can find the abstract pins and physical pins. For this example: OC3A = PE3/pin 5, OC3B = PE4/pin 6, OC3C = PE5/pin7.
4) Now we look at the board schematic to get to the Arduino I/O connector pins. Here it helps that they have organized the pins into the 8-bit logical groups, so it's easy to find PE3-PE5 (lower middle right side of chip U1).



5) Our 3 signals go over to **Arduino pins 2,3,5** in the connector labeled 8X1F-H8.5 (the PWM section of the board edge. Note that for some reason, the schematic shows that Arduino Mega board designers scrambled the wires a bit so we have:



Name	Abstract pin	HW pin	Arduino Pin
OC3A	PE3	5	5
OC3B	PE4	6	2
OC3C	PE5	7	3

Programming the Timers

Timers inside the ATmega2560 are powerful digital instruments. You get a whole set of them:

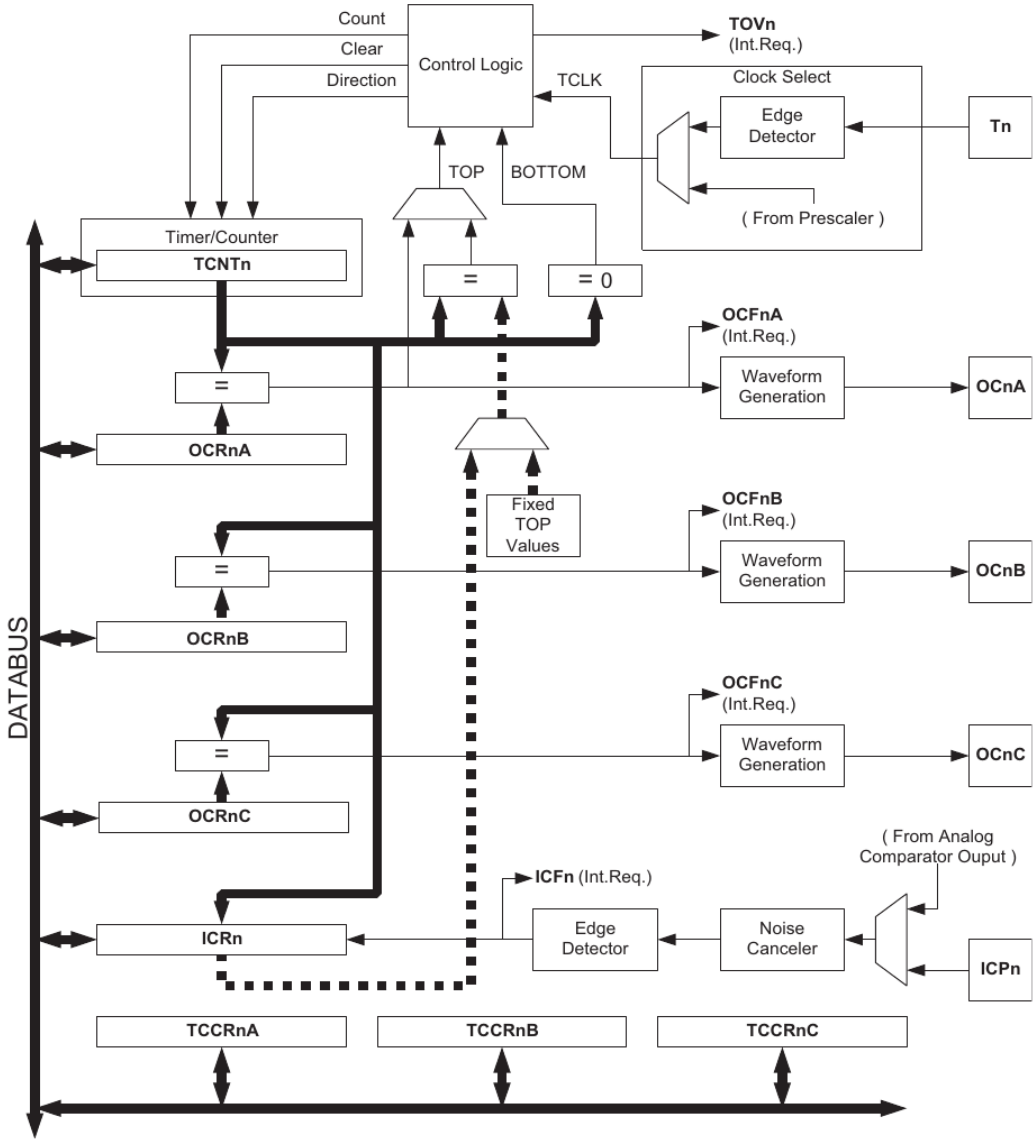
Timer #	# of bits	Special Functions	Page ref.
0	8	Basic counter with two PWM outputs (Carrier For T1)	115
1	16	High res (16 bit) PWM, three outputs, Frequency Generator, Multiple interrupts per timer, event time-stamper. (Modulates T0)	133
2	8	Same as 0, but can use/sync with external clock signal.	169
3	16	Same as 2 (no modulation)	133
4	16	Same as 2 (no modulation)	133
5	16	Same as 2 (no modulation)	133

(color indicates timers with exact same features. All have many overlapping features.)

Considering the 16 bit timers, the skinny horizontal rectangles in their block diagram (Fig 17-1 on page 134) are registers that hold 16 bits each, connected by a data-bus (heavy black double-arrows)[1].

[1] The data-bus is only 8 bits so there is a two-step process for reading or writing all 16 bits (sec 17.3) but good news, the C-compiler takes care of the two-step process automatically so we don't have to worry about it.

Figure 17-1. 16-bit Timer/Counter Block Diagram[1]



The core of the timer/counter is the 16 bit counter itself (TCNTn) which counts either CPU clock pulses (measuring time) or external pulses (counting random external events). The “Clock Select” block chooses between those options. Then there are several comparators (“=” blocks) that can detect when the counter reaches some value. That value is specified by bits in the OCRnA-C registers. Another nifty feature is to be able to capture the clock’s value in a register when some external event takes place. The ICRn register handles that task. All these options and features are selected and controlled by bits in the control registers TCCRnA - TCCRnC.