The pins on the Arduino can be accessed using the `digitalWrite` and `digitalRead` functions however these functions have a lot of overhead in them, taking a lot of time away from doing the task that is needed. The function is included at the end of this write up and can be seen to do a lot of error checking and extra operations. The error checking helps make the program safer and less error prone, but requires time and on an embedded processor we will eventually run out of processing time.

A more direct route has been provided for changing the pins on the Arduino and is known as Port Manipulation (a google search for Arduino Port Manipulation, should find a good reference for you). This consists of a set of variables, the bits of which represent the state of the pins.

Variable Name	Function	Pins	Bits
PORTB	Sets the output values for pins	13,12,11,10,9,8	5,4,3,2,1,0
PINB	Reads the input values for pins	13,12,11,10,9,8	5,4,3,2,1,0
DDRB	Data Direction Register for B (1 in a bit indicates pin is an output)	13,12,11,10,9,8	5,4,3,2,1,0
PORTD	Sets the output values for pins	7,6,5,4,3,2,1,0	7,6,5,4,3,2,1,0
PIND	Reads the input values for pins	7,6,5,4,3,2,1,0	7,6,5,4,3,2,1,0
DDRD	Data Direction Register for B (1 in bit indicates pin is an output)	7,6,5,4,3,2,1,0	7,6,5,4,3,2,1,0
PORTC	Sets the output values for pins	A5 to A0	5,4,3,2,1,0
PINC	Reads the input values for pins	A5 to A0	5,4,3,2,1,0
DDRC	Data Direction Register for A5-A0 (1 in a bit indicates pin is an output)	A5 to A0	5,4,3,2,1,0

As a way of interpreting this, an example might be the most illuminating. Consider the case of setting up pin 13, which is connected to the LED on the Arduino board, as an output and then turning it on and off as quickly as possible. The code would look like this

```
DDRB  |= 0x20;  // Set bit 5 high, making pin 13 an output
PORTB |= 0x20;  // Set bit 5 in PORTB high, making pin 13 high.
PORTB &= ~(0x20); // Set bit 5 low, making pin 13 low.
```

On the other hand if you were to test the input on a pin, you would need to test the variable PINB as in the following

```
DDRB  |= ~0x10;  // Set bit 4 low, making pin 12 an input
if( PINB & 0x10 ) // Test if bit 4 is high
```

Appendix A. Code for digitalWrite

```
void digitalWrite(uint8_t pin, uint8_t val)
{
	uint8_t timer = digitalPinToTimer(pin);
	uint8_t bit = digitalPinToBitMask(pin);
	uint8_t port = digitalPinToPort(pin);
	volatile uint8_t *out;
	if (port == NOT_A_PIN) return;

	// If the pin that support PWM output, we need to turn it off
	// before doing a digital write.
	if (timer != NOT_ON_TIMER) turnOffPWM(timer);

	out = portOutputRegister(port);

	uint8_t oldSREG = SREG;

	cli();

	if (val == LOW) {
	   *out &= ~bit;
	}
	else {
	   *out |= bit;
	}

	SREG = oldSREG;
}
```