I/O speeds?

This question has been **Answered.**

Web link: https://communities.intel.com/thread/45267

Correct Answer
by deckard026354 on Oct 4, 2013 7:09 AM

Joe

- GPIO
  - Default max throghput is 230 Hz
  - IO2 and IO3 if configured as OUTPUT_FAST - will clock in at 477 kHz - to 2.93 MHz
    - digitalWrite() will achieve 477 kHz
    - fastGpioDigitalWrite() - an Intel extension of the Arduino API will achieve 680 kHz
    - fastGpioDigitalWriteDestructive - can achieve GPIO speeds of about 2.93 MHz
  - Example sketch usage below
- SPI
  - The Arduino API does byte-by-byte SPI transactions - which are not especially fast
  - Intel has provided an API extension to allow transfer of larger blocks of SPI data transferBuffer
  - This extension should allow Arduino sketches to take better advantage of the SPI interface.
  - I don't have the throughput number to hand - will search for it.
- ADC
  - The throughput of the ADC is constrained by the SPI
- Interrupts
  - In theory it would be possible to circumvent the GPIO input mechanism to kick a task in user-space
  - Right now we allow
    - GPIO lib based callbacks
    - HPET driven callbacks with granularity up to 1 kHz
  - As with all systems based on a general purpose operating system and even some systems that claim determinism - your reaction time to any given event is a function of the workload the processors is engaged in. A quiescent system is likely to react very quickly to a GPIO even if its triggering a user-space application. A system under load could have significant jitter. This is down to the choices made by the application engineer re: how much concurrent processing is appropriate.

Galileo - has two pins IO2 and IO3 through which we can drive significant data rates.
By default these two pins are routed to the cypress.
There are three methods to communicate with these pins - which have increasing throughput

1. digitalWrite()
   1. Using this method it is possible to toggle an individual pin in a tight loop @ about 477 kHz
   2. pinMode(2, OUTPUT_FAST);
   3. pinMode(3, OUTPUT_FAST);
   4. This is a read-modify-write
      1. To toggle a bit - first we read
      2. Then we update
      3. Then we write back
2. fastGpioDigitalWrite(register uint8_t gpio, register uint8_t val)
   1. This function actually lets you write directly to the registers - without going through the code around digitalWrite() and consequently has better performance than a straight digitalWrite
   2. Using this method it is possible to toggle an individual pin (GPIO_FAST_IO2, GPIO_FAST_IO3) at about 680 kHz
   3. pinMode(2, OUTPUT_FAST);
   4. pinMode(3, OUTPUT_FAST);
      1. fastGpioDigitalWrite(GPIO_FAST_IO2, 1);
      2. fastGpioDigitalWrite(GPIO_FAST_IO3, 0);
   5. Again this uses read/modify/write - and can toggle one GPIO at a time

3. fastGpioDigitalWriteDestructive(register uint8_t gpio_mask);
    1. Using this method it is possible to achieve 2.93 Mhz data toggle rate on IO2/IO3 individually or simultaneously
    2. pinMode(2, OUTPUT_FAST);
    3. pinMode(3, OUTPUT_FAST);
    4. It is the responsibility of the application to maintain the state of the GPIO registers directly
    5. To enable this a function called fastGpioDigitalLatch() is provided - which allows the calling logic to latch the initial state of the GPIOs - before updating later
    6. This method just writes GPIO bits straight to the GPIO register - i.e. a destructive write - for this reason it is approximately 2 x faster then read/modify/write

Example-1 - outputs 477kHz waveform on IO2:

```
setup(){
  pinMode(2, OUTPUT_FAST);
}
loop()
{
  register int x = 0;
  while(1){
    digitalWrite(2, x);
    x =!x;
  }
}
```

Example-2 - outputs 683kHz waveform on IO3:

```
setup(){
  pinMode(3, OUTPUT_FAST);
}
loop()
{
  register int x = 0;
  while(1){
    fastGpioDigitalWrite(GPIO_FAST_IO3, x);
    x =!x;
  }
```

Example-3 - outputs 2.93MHz waveform on IO3:

```
uint32_t latchValue;
setup(){
  pinMode(3, OUTPUT_FAST);
  latchValue = fastGpioDigitalLatch();
}
loop()
{
  while(1){
    fastGpioDigitalWriteDestructive(latchValue);
    latchValue ^= GPIO_FAST_IO3;
  }
}
```

Example-4 - outputs 2.93MHz waveform on both IO2 and IO3:

```
uint32_t latchValue;
setup(){
  pinMode(2, OUPUT_FASTMODE);
  pinMode(3, OUPUT_FASTMODE);
  latchValue = fastGpioDigitalLatch(); // latch initial state
}
loop()
{
  while(1){
    fastGpioDigitalWriteDestructive(latchValue);
    if(latchValue & GPIO_FAST_IO3){
```

```
        latchValue |= GPIO_FAST_IO2;
        latchValue &= ~ GPIO_FAST_IO3;
      }else{
        latchValue |= GPIO_FAST_IO3;
        latchValue &= GPIO_FAST_IO2;
      }
   }
}
```

In other words the responsibility lies with the application designer in cases 3 and 4 to ensure the GPIO register values are correct - assuming - these values matter to the application use-case