Manipulating Pins on the Teensy 3.5
Data Types

• short, int, long: size depends on the particular microprocessor

• In order to be clear about sizes, gcc (our compiler) provides a set of types, including:
  • int8_t 8-bit signed
  • uint16_t 16-bit unsigned
  • uint32_t 32-bit unsigned

• Use these for our projects – not short, int, long
Teensy 3.5
Teensy 3.5

- Floating Point Unit (FPU): high-speed math
- Serial I/O: RS232, I2C, SPI, CAN, Ethernet
- Digital I/O
- Pulse Width Modulation (PWM)
- Multiple timers
- Digital-to-analog converter channels (2)
- Analog-to-digital converter channels (25)
Digital Input/Output

The Teensy encodes a digital value using 0V (low) and 3.3V (high)

• If a pin is an input:
  • We can ask the pin what its voltage state is
  • Possible answers: 0 or 1 (low or high)

• If a pin is an output:
  • We can drive the pin to be 0V or 3.3V
  • Again, these are encoded digitally as 0 or 1
Digital Input/Output

• Pins are organized into groups, called PORTS
• Each port can be composed of up to 32 pins
  • In practice, this number is generally much smaller
• The ports are named A … E
Teensy 3.5 Schematic

Key take-away: shows us the connection between the Teensy pin numbers and the Arm Cortex M4 I/O ports
Teensy 3.5
Schematic
Teensy 3.5 Schematic

- Port C, bit 2
Teensy 3.5 Schematic

- Port C, bit 2
- Teensy pin 23
  - Also analog pin 9
Pins in the Arm Cortex M4

- Most pins have multiple possible functions
  - Can be a digital input or output
  - Can generate a continuous voltage (analog output)
  - Can read a continuous voltage (analog input)
Configuring a Pin for Digital Output

There is an on-board LED connected to PORT C, bit 5: let’s write code to blink the LED

• Initialization:

```c
// Initialize PORT C, bit 5 to be a digital I/O bit
PORTC_PCR5 = PORT_PCR_MUX(0x1);
```

• PORTC_PCR5 is a special-purpose register (32 bits) that controls what this specific pin does

• PCR = Port Configuration Register

Andrew H. Fagg: Embedded Real-Time Systems: Digital IO
Configuring a Pin for Digital Output

• Initialization, step 2:

```c
// Configure bit 5 to be an output (and all others to be inputs)
GPIOC_PDDR = 0x20;
```

• GPIO = General Purpose Input/Output

• PDDR = Port Data Direction Register

• On boot: all pins are configured as analog inputs

Andrew H. Fagg: Embedded Real-Time Systems: Digital IO
Setting to Pin into the High State

// Turn on the bit (and all others off)
GPIOC_PDOR = 0x20;

• The pin is now in a high state
• PDOR = Port Data Output Register
Putting it Together in the Arduino Environment

This function is called when the processor first boots:

```c
void setup() {
    // Configure PORTC, bit 5 to be a digital I/O bit
    PORTC_PCR5 = PORT_PCR_MUX(0x1);

    // Configure bit 5 to be an output (and all others to be inputs)
    GPIOC_PDDR = 0x20;
}
```
Putting it Together in the Arduino Environment

And this function is called repeatedly thereafter:

```c
void loop() {
    // Turn on the bit (and all others off)
    GPIOC_PDOR = 0x20;

    // Wait for 0.1 second
    delay(100);

    // Turn off the bit (and all others)
    GPIOC_PDOR = 0;

    // Wait for 0.1 second
    delay(100);
}
```

Andrew H. Fagg: Embedded Real-Time Systems: Digital IO
Arduino Environment

The environment automatically includes the following function:

```c
void main() {
    setup();

    while(1) {
        loop();
    }
}
```
An Alternative Implementation

void loop() {

    //
    GPIOC_PDOR ^= 0x20;

    // Wait for 0.1 second
    delay(100);
}

Andrew H. Fagg: Embedded Real-Time Systems: Digital IO
PORTS A .. E

- PORTx_PCRy = each bit has one register
- GPIOx_PDDR, GPIOx_PDOR: each port has one register

- Note: the Arduino environment provides other ways to manipulate these pins (don’t use these alternatives!)
  - For digital I/O, we will use these registers. We get:
    - Efficiency
    - Simultaneous state change of multiple pins
Teensy 3.5 Schematic

- Let’s connect LEDs to PTD5 & 6
- Don’t forget the resistor!
void setup() {
    // Configure PORTD, pins 5 & 6 as digital I/O
    PORTD_PCR5 = PORT_PCR_MUX(0x1);
    PORTD_PCR6 = PORT_PCR_MUX(0x1);

    // Configure bit 5 & 6 to be outputs
    GPIOD_PDDR = 0x60;
}

Initialization
What does this program do?

```c
void loop() {
    GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x60;
    delay(250);
    GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x20;
    delay(250);
    GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x40;
    delay(250);
    GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x0;
    delay(250);
}
```
What does this program do?

```c
void loop() {
  GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x60;
delay(250);
  GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x20;
delay(250);
  GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x40;
delay(250);
  GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x0;
delay(250);
}
```

Flashes LED on PD6 at 2 Hz on PD5: 1 Hz
Duty Cycle for each: 50%
... go to Bit Manipulation
Let’s connect a switch to PTC2

Don’t forget the pull-up resistor!

If switch reads zero, turn PTD6 on and PTD5 off

Otherwise, turn PTD6 off and PTD5 on
void setup() {
    // Configure PORTD, pins 5 & 6 as digital I/O
    PORTD_PCR5 = PORT_PCR_MUX(0x1);
    PORTD_PCR6 = PORT_PCR_MUX(0x1);

    // Configure PORTC, pin 2 as digital I/O
    PORTC_PCR2 = PORT_PCR_MUX(0x1);

    // Configure bit 5 & 6 to be outputs
    GPIOD_PDDR = 0x60;
}
Loop Implementation

```c
void loop() {
  if(GPIOC_PDIR & 0x4) {
    // Switch open
    GPIOD_PDOR = ...
  } else {
    // Switch closed
    GPIOD_PDOR = ...
  }
}
```
```c
void loop() {
    if(GPIOC_PDIR & 0x4)
    {
        // Switch open
        GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x40;
    }
    else{
        // Switch closed
        GPIOD_PDOR = (GPIOD_PDOR & ~0x60) | 0x20;
    }
}
}```