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

- MOSI1, RX1, GND
- MOSI0, TX1, PW2
- SCL2, CAN0TX, PW3
- SDA2, CAN0RX, PW4
- miso1, tx1, PW5
- scl0, mosi0, PW7
- sda0, miso0, PW8
- CS0, RX2, PW9
- CS0, TX2, PW10
- MOSI0, PW11
- MIS00, PW12
- 3.3V, PW13 (LED)
- GND
- Vin (3.6 to 6.0 volts)
- Analog GND
- 3.3V (250 mA max)
- 23 A9, PWM
- 22 A8, PWM
- 21 A7, PWM
- 20 A6, PWM
- 19 A5, PWM
- 18 A4, PWM
- 17 A3, PWM
- 16 A2, PWM
- 15 A1, PW14
- 14 A0, PWM
- 13 SCK0
- 12 CS0
- 11 mosi1
- 10 sck1
- 9 sda0
- 8 sda0
- 7 scl0
- 6 CS0
- 5 PW6
- 4 PW5
- 3 PW4
- 2 PW3
- 1 PW2
- 0 PW1

- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
- Touch, Touch
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

MK64FX512 (Teensy 3.5)
MK66FX1M0 (Teensy 3.6)
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
Configuring a Pin for Digital Output

• Initialization, step 2:
  
  // 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
Setting to Pin into the High State

```c
// 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:

```cpp
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);
}
```
Arduino Environment

The environment automatically includes the following function:

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

    while(1)
    {
        loop();
    }
}
```
void loop() {
    //
    GPIOC_PDOR ^= 0x20;

    // Wait for 0.1 second
    delay(100);
}
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. 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!
Initialization

```c
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;
}
```
What does this program do?

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

Flashes LED on PD6 at 2 Hz
on PD5: 1 Hz

Duty Cycle for each: 50%
… go to Bit Manipulation
Teensy 3.5 Schematic

- 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
Initialization

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