I/O Ports

Here a USB port is illustrated.

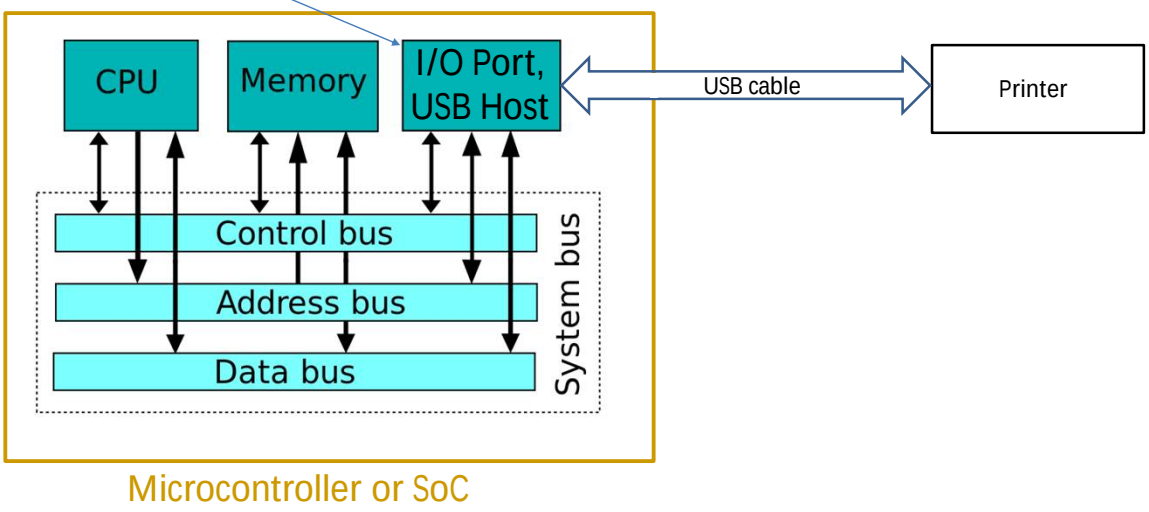


Illustration: https://commons.wikimedia.org/wiki/File:Computer_system_bus.svg

1

I/O Ports

Or it could be an Ethernet port

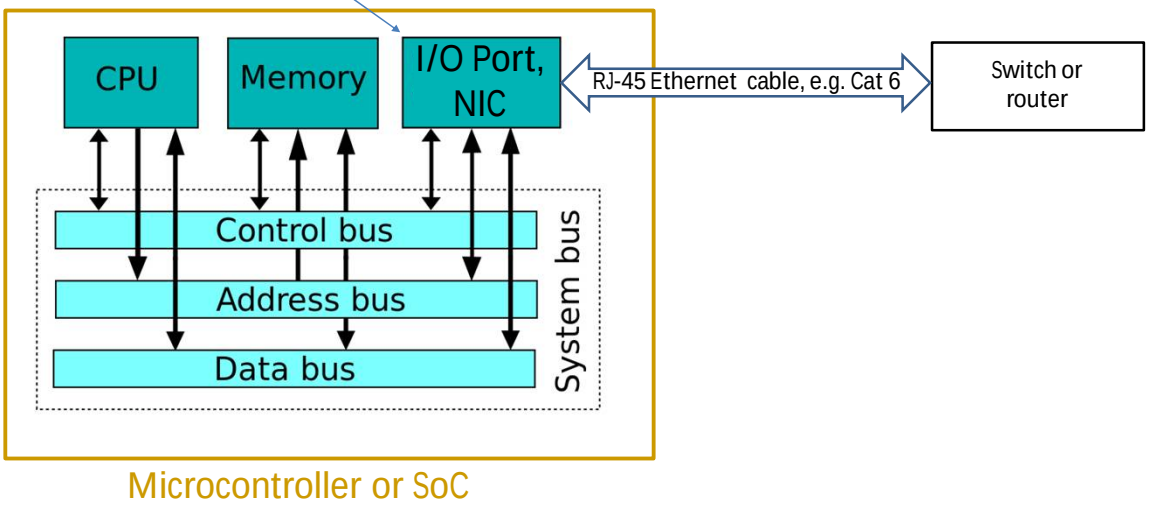


Illustration: https://commons.wikimedia.org/wiki/File:Computer_system_bus.svg

2

I/O Ports

Here a parallel port is illustrated.
(Centronics parallel ports are obsolete. Parallel ports for consumer electronics in general are obsolete.
But. . . For lab work and various applications that will not be exposed to consumers, parallel ports have a place.)

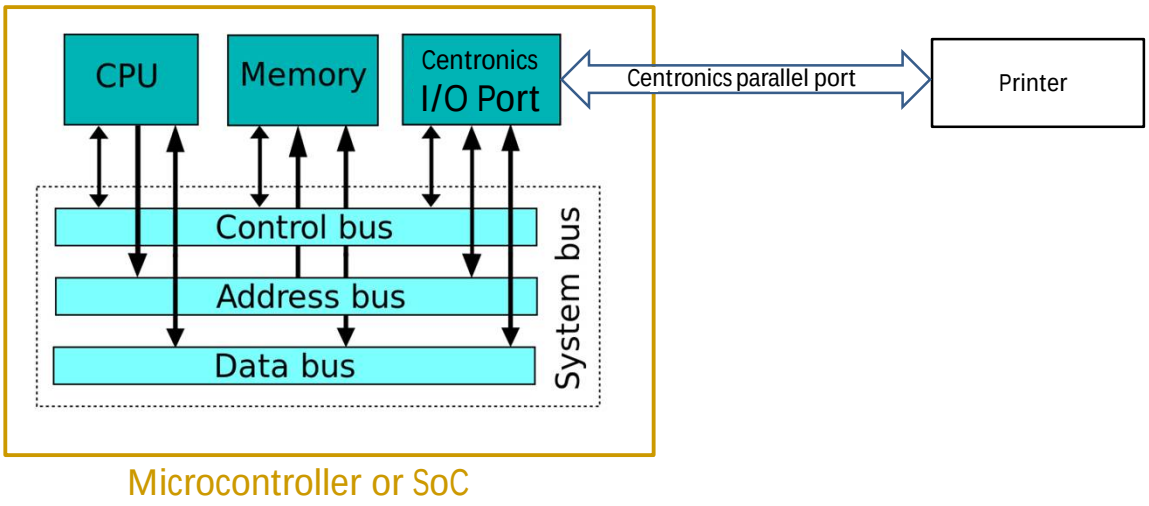
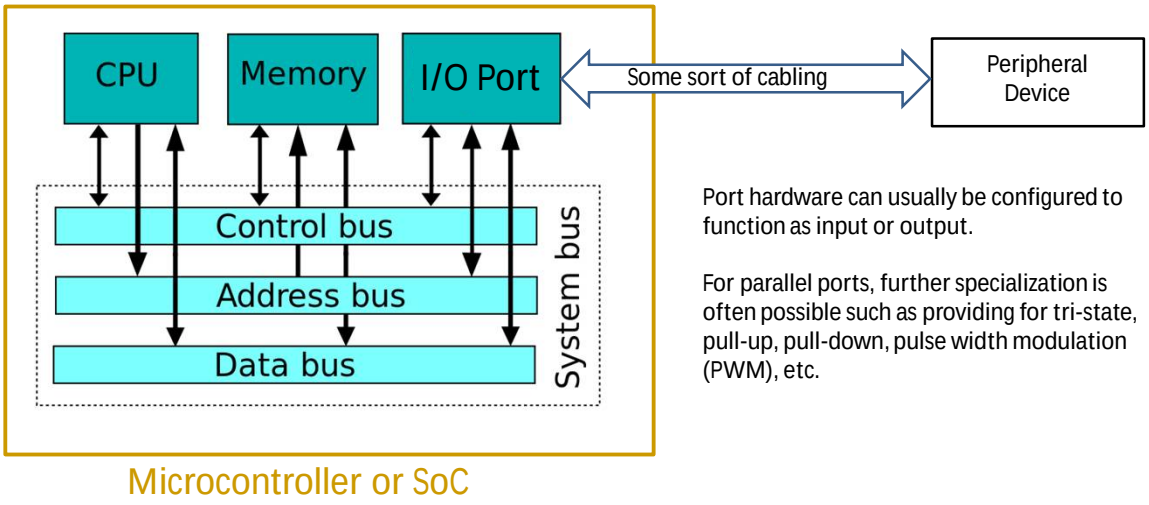


Illustration: https://commons.wikimedia.org/wiki/File:Computer_system_bus.svg

3

I/O Ports

A *port* is a set of hardware that creates an interface from the main CPU bus to some general I/O.
A *parallel* port is a data word wide (> 1 bit) in the cabling (occasionally ½ a data word wide or ¼ a data word wide, etc.)
E.g. a 32-bit CPU is likely to have 32-bit wide ports, but some ports might be 16 or 8 bits wide.



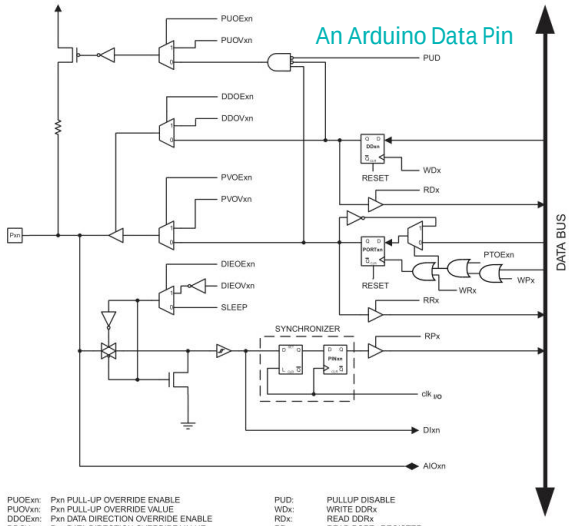
Port hardware can usually be configured to function as input or output.

For parallel ports, further specialization is often possible such as providing for tri-state, pull-up, pull-down, pulse width modulation (PWM), etc.

Illustration: https://commons.wikimedia.org/wiki/File:Computer_system_bus.svg

4

Parallel port—a typical schematic.
Raspberry Pi's Broadcom SoC chip is proprietary and a datasheet is not available to the public.
Below, a typical port on an AVR μC such as is used on an Arduino.



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE	PUD:	PULLUP DISABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE	WDx:	WRITE DDRx
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE	RDx:	READ DDRx
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE	RRx:	READ PORTx REGISTER
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE	WRx:	WRITE PORTx
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE	RPx:	READ PORTx PIN
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE	WPx:	WRITE PINx
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE	$clk_{I/O}$:	I/O CLOCK
SLEEP:	SLEEP CONTROL	DIxn:	DIGITAL INPUT PIN n ON PORTx
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE	AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

The schematic shows just one bit-slice of the parallel port.
The various control lines come from registers.
The registers are controlled with "setup" instructions
Typically you run setup instructions once only at startup.
The setup instructions configure the port is input, output…
Not shown—some ports include various protection circuits
--static zap protection
--overload protection

Each general digital I/O pin is primarily subject to three Important simultaneous control signals from registers.

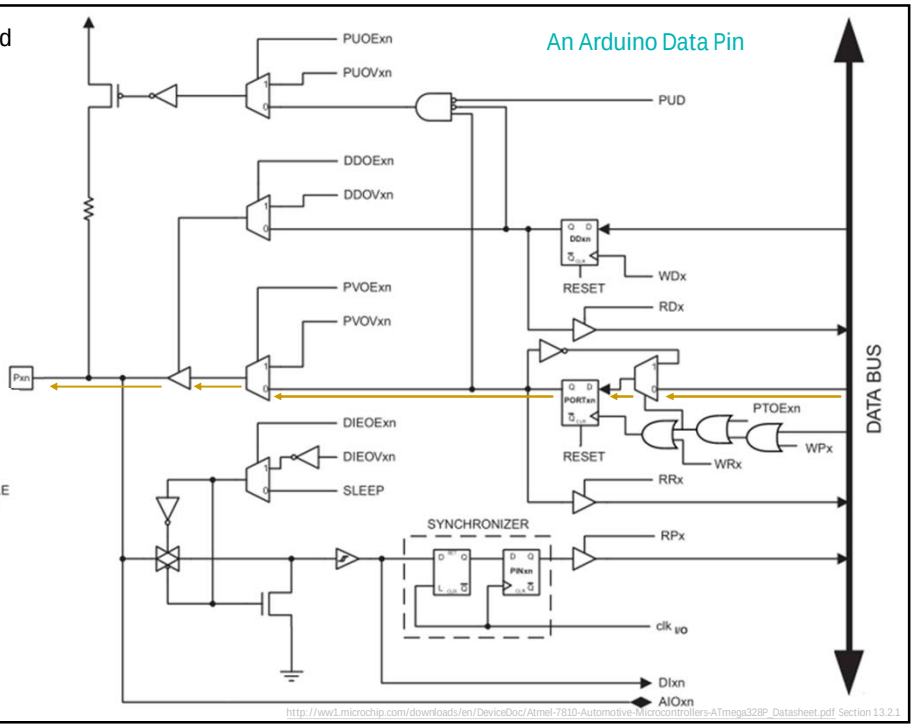
a.) Data direction (input or output)
b.) Output data (either as named, or creates a pull-up/dn)
c.) Input data (either as named, or verification of output)

In addition,
e.) Pullup/dn enable
f.) tristate enable (which may be controlled by another pin)

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

5

Suppose the port is to be configured for normal output.



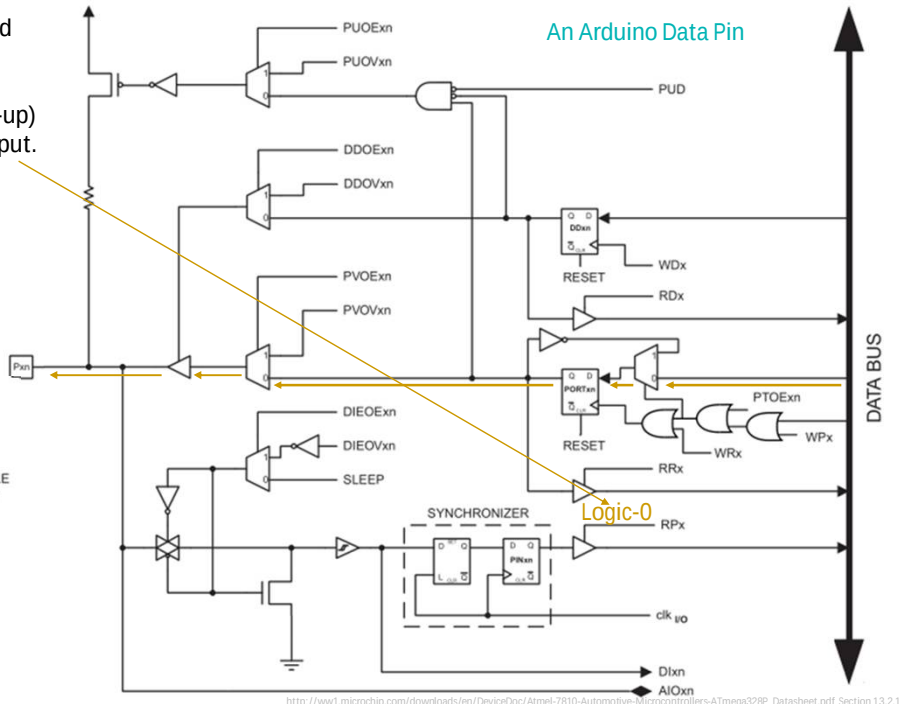
PUOExn:	Pxn PULL-UP OVERRIDE ENABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP:	SLEEP CONTROL
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD:	PULLUP DISABLE
WDx:	WRITE DDRx
RDx:	READ DDRx
RRx:	READ PORTx REGISTER
WRx:	WRITE PORTx
RPx:	READ PORTx PIN
WPx:	WRITE PINx
$clk_{I/O}$:	I/O CLOCK
DIxn:	DIGITAL INPUT PIN n ON PORTx
AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

6

Suppose the port is to be configured for normal output.

During set-up (may be part of boot-up) Disable all the hardware for data input.



PUOExn:	Pxn PULL-UP OVERRIDE ENABLE
PUOVxn:	Pxn PULL-UP OVERRIDE VALUE
DDOExn:	Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn:	Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn:	Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn:	Pxn PORT VALUE OVERRIDE VALUE
DIEOExn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn:	Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP:	SLEEP CONTROL
PTOExn:	Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD:	PULLUP DISABLE
WDx:	WRITE DDRx
RDx:	READ DDRx
RRx:	READ PORTx REGISTER
WRx:	WRITE PORTx
RPx:	READ PORTx PIN
WPx:	WRITE PINx
$clk_{I/O}$:	I/O CLOCK
DIxn:	DIGITAL INPUT PIN n ON PORTx
AIOxn:	ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

7

Tri-State Gates & Register Transfer Operations:

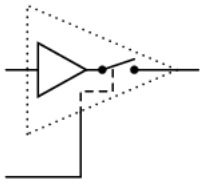
EGR 204 Review

They happen very frequently
They need to be fast—one clock edge to copy data from one register to another.
Every register needs to accommodate transfers. (Else impossible to load the register!)
Thus a minimal amount of hardware to support each transfer pathway is desirable.

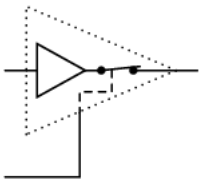
To address this specialized need an analog technique is used:

A switching mechanism is used to connect or disconnect a gate output to a bus.
On the left the switch is shown "open" meaning disconnected, no current flows.
On the right the switch is shown "closed" meaning connected, current may flow.
Imagine that I add a handle to control the switch. Imagine that the handle is Electrically activated. If E_n = 0 switch is open. If E_n = 1 switch is closed.

Now wrap this all up into one symbol:



Open: no current possible

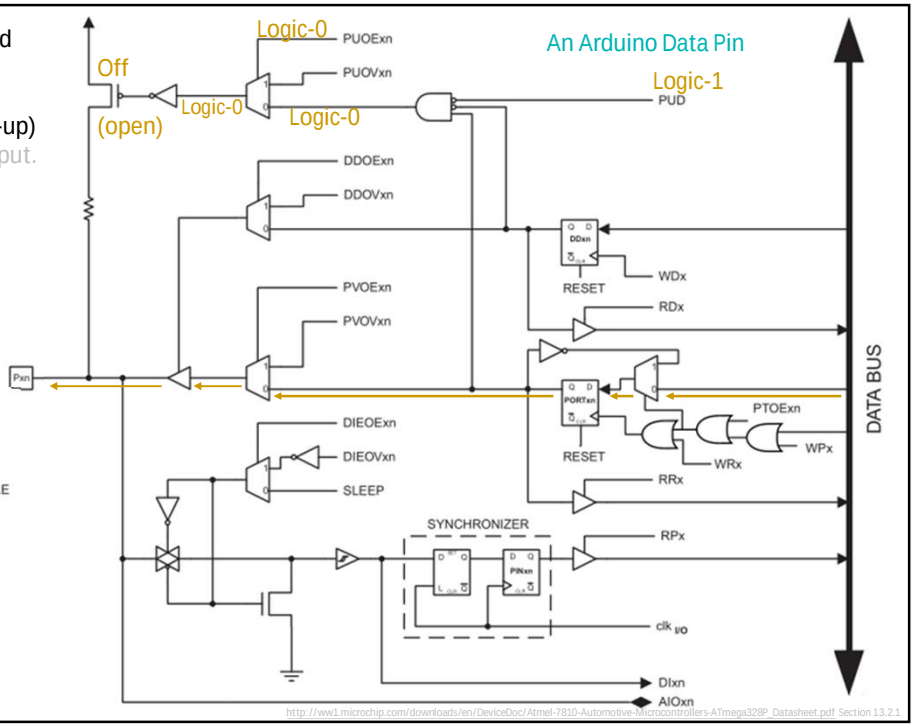


Closed: current may flow

8

Suppose the port is to be configured for normal output.

During set-up (may be part of boot-up)
Disable all the hardware for data input.
Turn off the output pull-up.



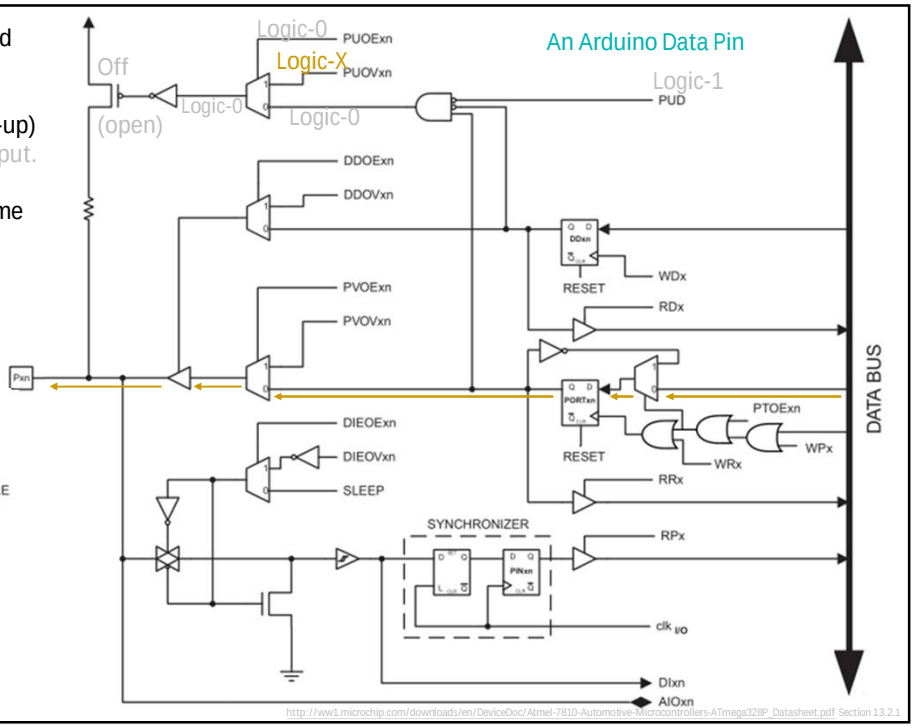
PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

9

Suppose the port is to be configured for normal output.

During set-up (may be part of boot-up)
Disable all the hardware for data input.
Turn off the output pull-up.
Some configuration option(s) become don't cares.



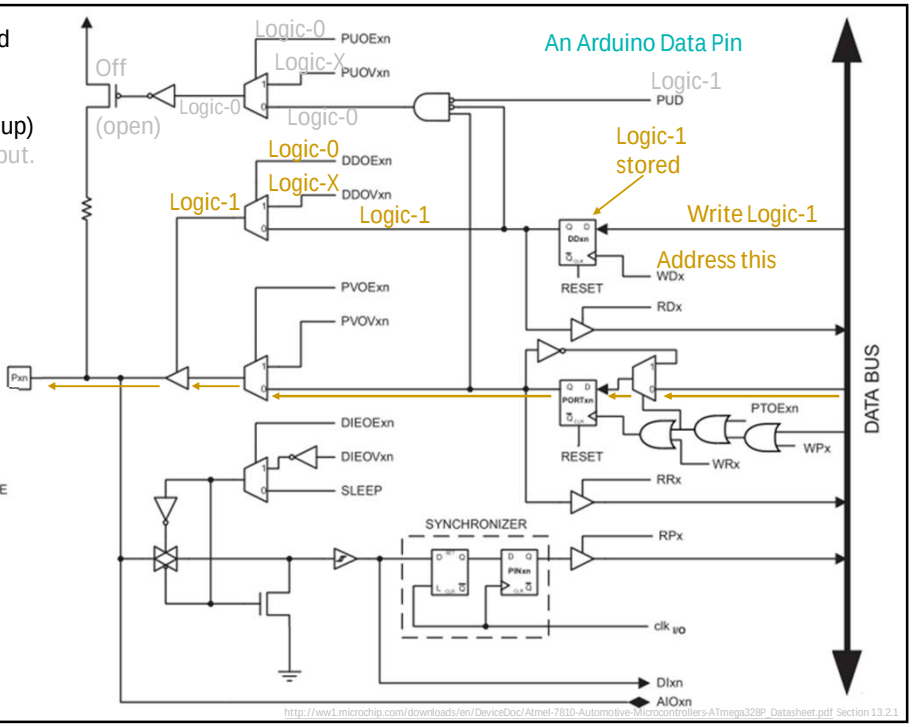
PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

10

Suppose the port is to be configured for normal output.

During set-up (may be part of boot-up)
Disable all the hardware for data input.
Turn off the output pull-up.
Enable the output driver.



PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

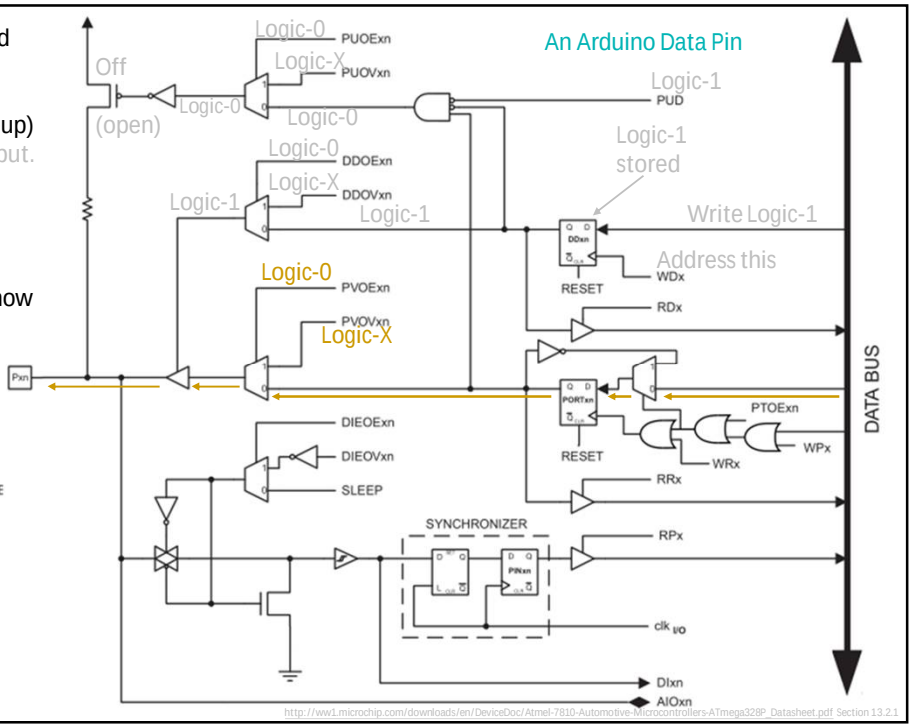
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

11

Suppose the port is to be configured for normal output.

During set-up (may be part of boot-up)
Disable all the hardware for data input.
Turn off the output pull-up.
Enable the output driver.
Select the output port register

Contents of PORTxn register are now driven to pin Pxn.



PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

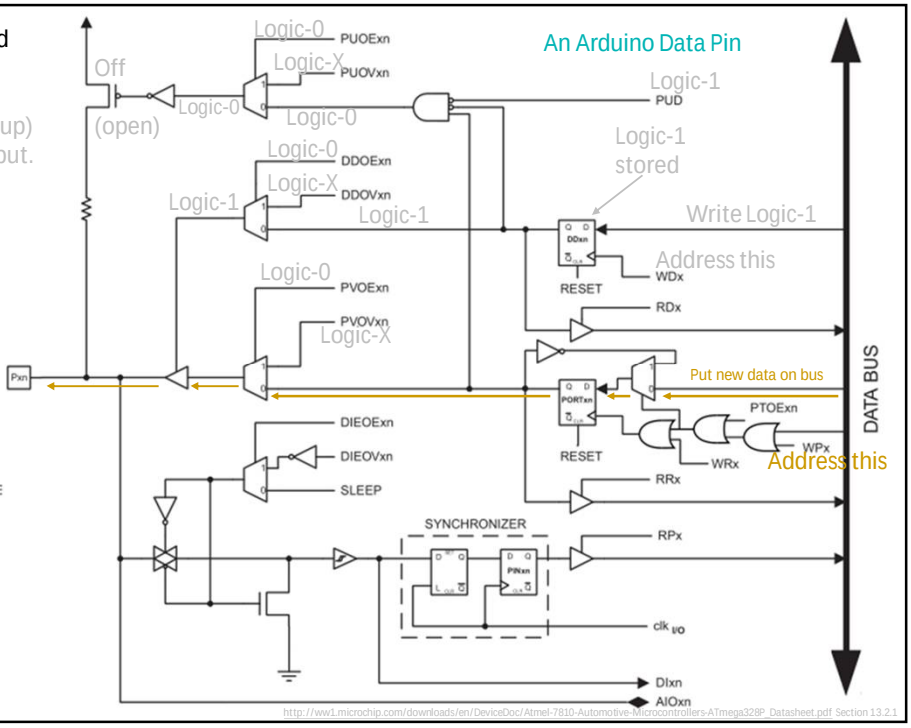
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

12

Suppose the port is to be configured for normal output.

During set-up (may be part of boot-up)
Disable all the hardware for data input.
Turn off the output pull-up.
Enable the output driver.
Select the output port register

During normal program operation,
Address WPx, new data to Pxn

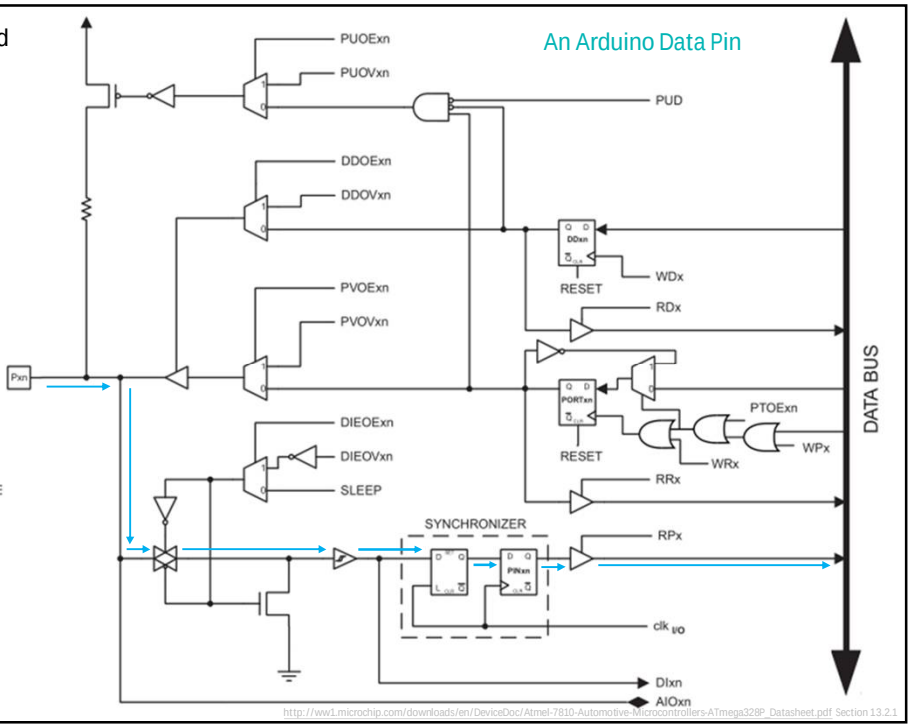


PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

13

Suppose the port is to be configured for normal input.



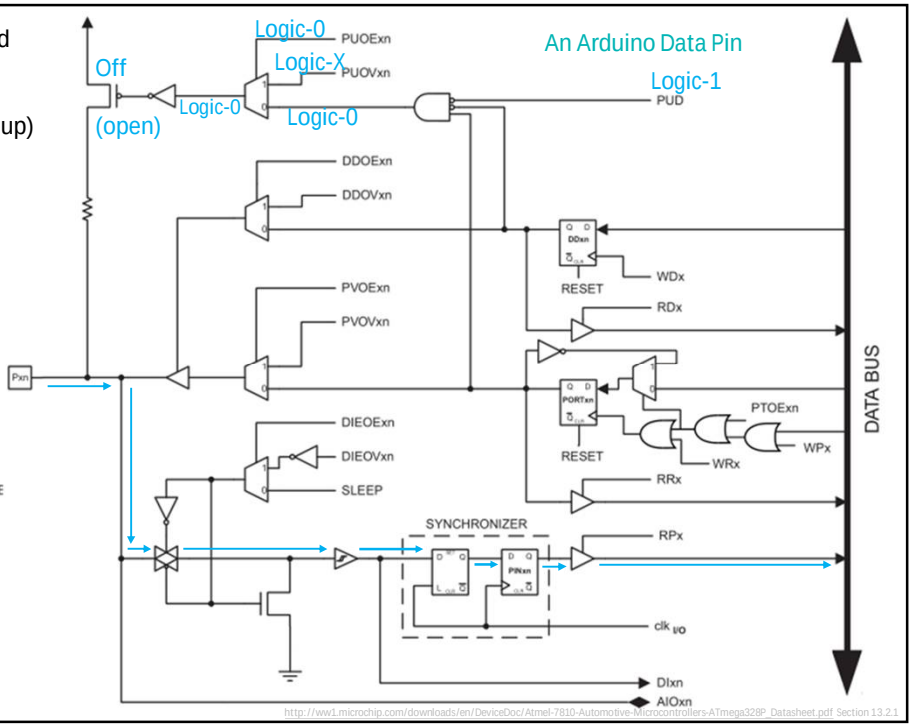
PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

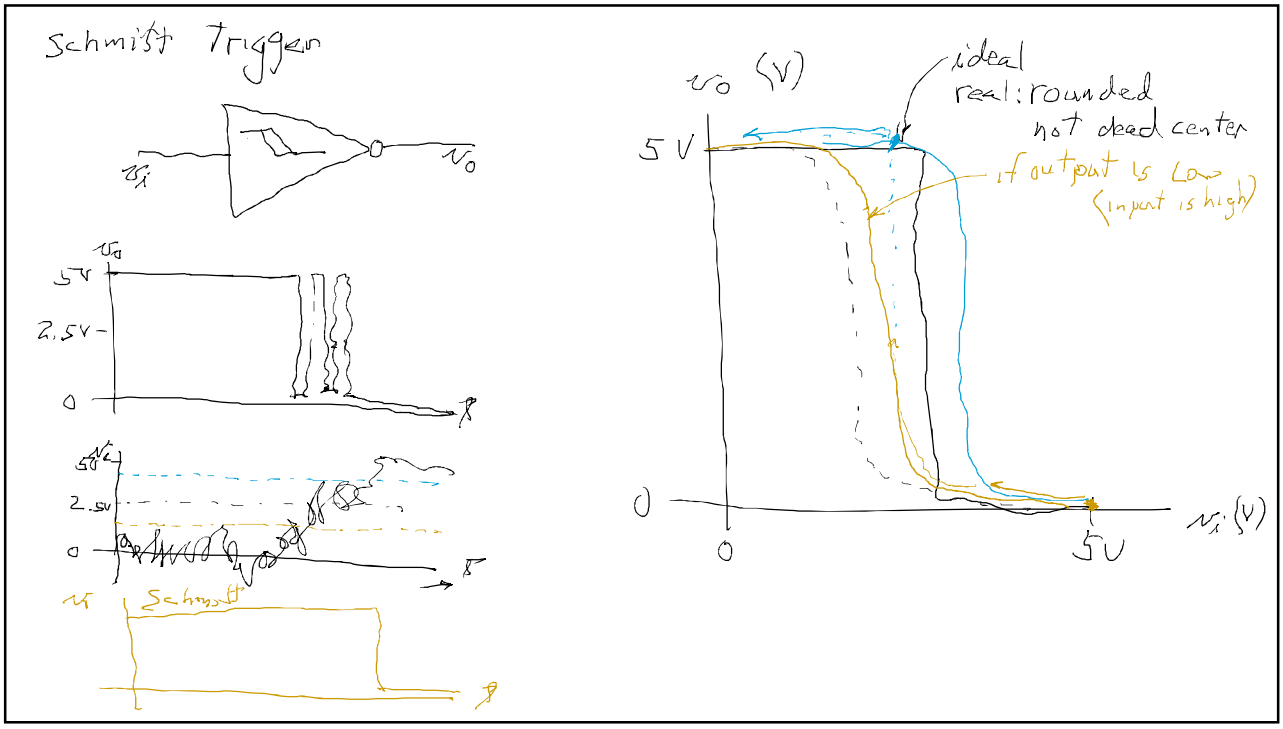
14

Suppose the port is to be configured for normal input.

During set-up (may be part of boot-up) Disable output pull-up.



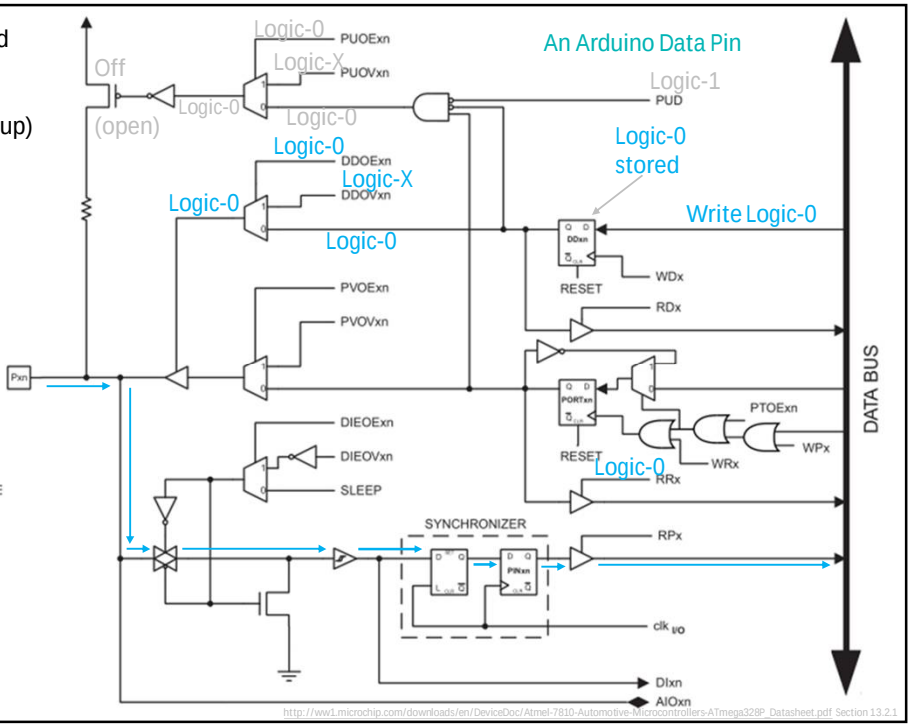
15



16

Suppose the port is to be configured for normal input.

During set-up (may be part of boot-up)
Disable output pull-up.
Disable output driver.



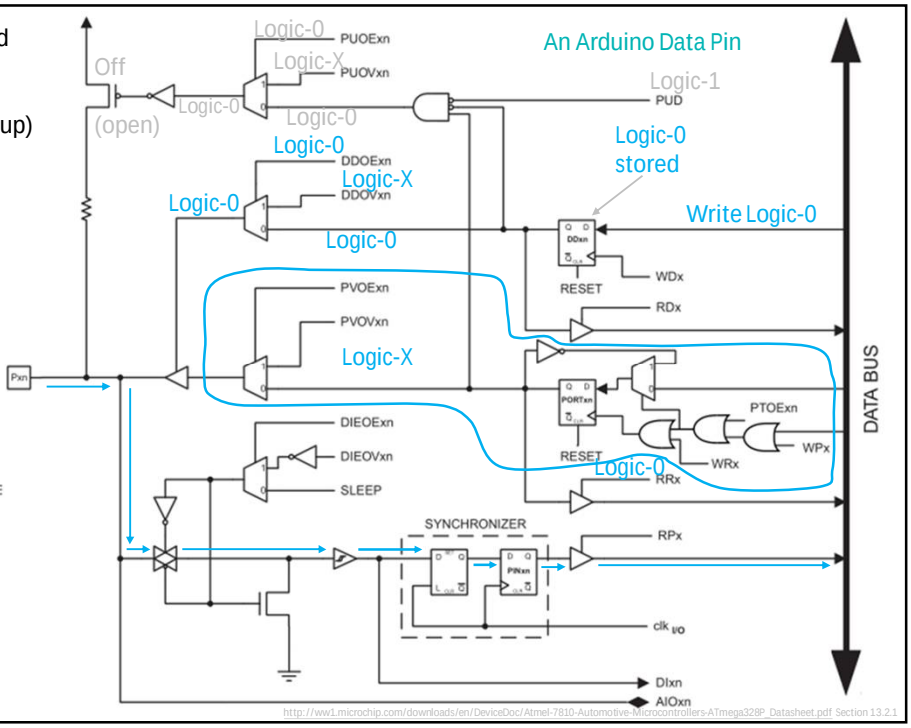
PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

17

Suppose the port is to be configured for normal input.

During set-up (may be part of boot-up)
Disable output pull-up.
Disable output driver.



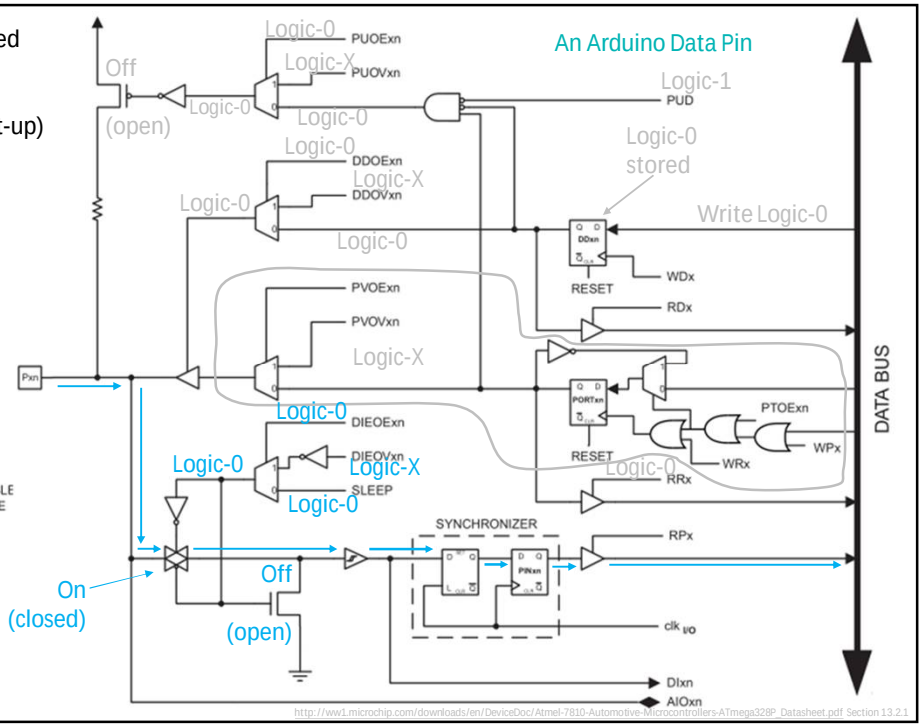
PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf Section 13.2.1

18

Suppose the port is to be configured for normal input.

During set-up (may be part of boot-up)
Disable output pull-up.
Disable output driver.
Enable the input buffer.



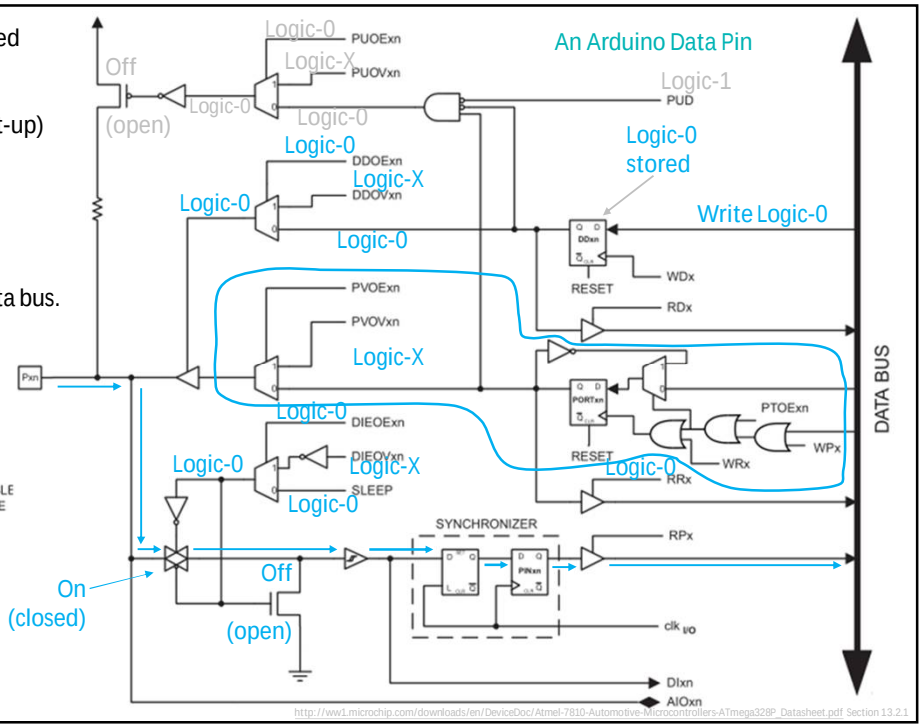
PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

19

Suppose the port is to be configured for normal input.

During set-up (may be part of boot-up)
Disable output pull-up.
Disable output driver.
Enable the input buffer.

During normal operation
Address RPx. Data from sync to data bus.



PUOExn: Pxn PULL-UP OVERRIDE ENABLE
PUOVxn: Pxn PULL-UP OVERRIDE VALUE
DDOExn: Pxn DATA DIRECTION OVERRIDE ENABLE
DDOVxn: Pxn DATA DIRECTION OVERRIDE VALUE
PVOExn: Pxn PORT VALUE OVERRIDE ENABLE
PVOVxn: Pxn PORT VALUE OVERRIDE VALUE
DIEOExn: Pxn DIGITAL INPUT-ENABLE OVERRIDE ENABLE
DIEOVxn: Pxn DIGITAL INPUT-ENABLE OVERRIDE VALUE
SLEEP: SLEEP CONTROL
PTOExn: Pxn, PORT TOGGLE OVERRIDE ENABLE
PUD: PULLUP DISABLE
WDx: WRITE DDRx
RDx: READ DDRx
RRx: READ PORTx REGISTER
WRx: WRITE PORTx
RPx: READ PORTx PIN
WPx: WRITE PINx
$clk_{I/O}$: I/O CLOCK
DIxn: DIGITAL INPUT PIN n ON PORTx
AIOxn: ANALOG INPUT/OUTPUT PIN n ON PORTx

20

Generally speaking, what is a parallel I/O port?

One word-width of the hardware depicted on the bit-slice schematic of the previous slide.
Say the port is a byte wide. Then 8x the previous schematic.
Say the port is 64-bits wide. Then 64x the previous schematic.

Parallel ports on microcontrollers are very configurable.
There is no standard that describes exactly how one of these parallel ports should work. You must consult the datasheet of the processor for each processor type that you work with.

A note of interest: Many people use the initialism "GPIO" to stand for General Purpose Input/output."
This is a reference to the type of parallel port being described here.

One must not confuse "GPI**O**" with "GPI**B**" which is a different thing—a very specific type of parallel port.

21

Parallel Ports of the Past

The type of parallel port described on previous slides should not be confused with some legacy ports which are only subsets of what has been presented.

The "Centronics Parallel Port" was popular on legacy IBM PC's and clones of those.
Those are indeed parallel ports, a subset of what is described above,
but they are not nearly so configurable as a general parallel port.

The Centronics parallel port is now highly standardized.
The "Centronics parallel port standard" is IEEE-1284. It is 8-bits, bidirectional.
The Centronics parallel port standard requires a "DB-25 female" connector
on the computer and a "female Centronics connector" on the printer.
A Centronics parallel printer cable has the mating connectors,
A "DB-25 male" and a "male Centronics connector" on the other end.

Another mostly obsolete parallel port standard of historical significance
(still in some minor use) is the GPI**B**, (not GPIO) or HP-IB port, defined by IEEE-488.
It is 8-bit, bidirectional. Devices always have a female connector.
Cables always have a male/female stacking connector on each end.

IEEE-488 connectors.
on a cable.
(Both ends are identical.)

Female DB-25, Centronics parallel port
on a computer

Female Centronics parallel port
on a printer

GPIB parallel port
on the back of an oscilloscope

https://commons.wikimedia.org/wiki/File:Centronics-36F.jpg
https://commons.wikimedia.org/wiki/File:Parallel_computer_printer_port.jpg
https://commons.wikimedia.org/wiki/File:IEEE-488-Stecker2.jpg

22

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or Hi-Z

23

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies

24

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies
a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

25

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies
a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

Gadfly: an annoying person (among other meanings)

b.) *Busy-waiting* (aka *gadfly* I/O): Use a status bit to check the I/O device before reading or writing to it. Result: while I/O device is busy, CPU needs to wait and monitor, hence the name.
(The CPU is analogous to kids in the back seat, "Are we there yet? Are we there yet? Are we there yet????" The kids don't do homework while waiting, they busy themselves only with the pestering question.)

26

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies

a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

Gadfly: an annoying person (among other meanings)

b.) *Busy-waiting* (aka *gadfly* I/O): Use a status bit to check the I/O device before reading or writing to it. Result: while I/O device is busy, CPU needs to wait and monitor, hence the name.
(The CPU is analogous to kids in the back seat, "Are we there yet? Are we there yet? Are we there yet?. . ."
The kids don't do homework while waiting, they busy themselves only with the pestering question.)

c.) *Periodic polling*: Similar to Busy-waiting, but CPU may work on other threads of code while waiting on a busy I/O device. (Requires a timer interrupt—i.e requires additional hardware.)
(The kids do homework while waiting. Every five minutes a bell rings and they ask the question.)

27

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies

a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

Gadfly: an annoying person (among other meanings)

b.) *Busy-waiting* (aka *gadfly* I/O): Use a status bit to check the I/O device before reading or writing to it. Result: while I/O device is busy, CPU needs to wait and monitor, hence the name.
(The CPU is analogous to kids in the back seat, "Are we there yet? Are we there yet? Are we there yet?. . ."
The kids don't do homework while waiting, they busy themselves only with the pestering question.)

c.) *Periodic polling*: Similar to Busy-waiting, but CPU may work on other threads of code while waiting on a busy I/O device. (Requires a timer interrupt—i.e requires additional hardware.)
(The kids do homework while waiting. Every five minutes a bell rings and they ask the question.)

d.) *Interrupt driven:* The I/O device has a method in hardware to request I/O service.
(The kids stick to their homework until told that they have arrived at their destination.)

28

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies

a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

Gadfly: an annoying person (among other meanings)

b.) *Busy-waiting* (aka *gadfly* I/O): Use a status bit to check the I/O device before reading or writing to it. Result: while I/O device is busy, CPU needs to wait and monitor, hence the name.
(The CPU is analogous to kids in the back seat, "Are we there yet? Are we there yet? Are we there yet?. . ."
The kids don't do homework while waiting, they busy themselves only with the pestering question.)

c.) *Periodic polling*: Similar to Busy-waiting, but CPU may work on other threads of code while waiting on a busy I/O device. (Requires a timer interrupt—i.e requires additional hardware.)
(The kids do homework while waiting. Every five minutes a bell rings and they ask the question.)

d.) *Interrupt driven:* The I/O device has a method in hardware to request I/O service.
(The kids stick to their homework until told that they have arrived at their destination.)

e.) *Direct Memory Access*: The I/O device takes over the CPU bus and writes directly into memory without CPU supervision. (The kids are not in the car!)

29

SUMMARY SLIDE

How can the port be controlled? How is I/O accomplished with this hardware?

It is a two-step process
1.) Setup: place bits in various control registers to establish . . .
--direction of I/O, input or output
--if input, and no connection to it, default high (enable pull up resistor), or default low (if available), or random
--if output, what is the initial output before the first write after power-up?, 1, 0 or X

2.) Do the actual I/0. There are various strategies

a.) *Blind-cycle*: Just do it immediately as the code runs. Not in coordination with the I/O device.
(Ready or not, hear I come!)

Gadfly: an annoying person (among other meanings)

b.) *Busy-waiting* (aka *gadfly* I/O): Use a status bit to check the I/O device before reading or writing to it. Result: while I/O device is busy, CPU needs to wait and monitor, hence the name.
(The CPU is analogous to kids in the back seat, "Are we there yet? Are we there yet? Are we there yet?. . ."
The kids don't do homework while waiting, they busy themselves only with the pestering question.)

c.) *Periodic polling*: Similar to Busy-waiting, but CPU may work on other threads of code while waiting on a busy I/O device. (Requires a timer interrupt—i.e requires additional hardware.)
(The kids do homework while waiting. Every five minutes a bell rings and they ask the question.)

d.) *Interrupt driven:* The I/O device has a method in hardware to request I/O service.
(The kids stick to their homework until told that they have arrived at their destination.)

e.) *Direct Memory Access*: The I/O device takes over the CPU bus and writes directly into memory without CPU supervision. (The kids are not in the car!)

30

Let's take a look at a parallel port I/O operation on an Arduino Uno board.

A few words about this entire environment. . .

Arduino is a registered trademark—in Italy.

Arduino is a business—a company that sells hardware and software.
It began life as a spinout from a graduate school in Italy, The Interaction Design Institute Ivrea.
The company has a checkered history of disputes regarding ownership rights, licenses, and copyrights.
In about 2017 Arduino entered some type of partnership with ARM Holdings.
The processor used in Arduino hardware is usually an AVR-family processor made by Atmel—not ARM.
"The recent goings on in the world of Arduino would be suitable for fictionalization as part of an epic boardroom drama TV show." https://www.i-programmer.info/news/91-hardware/11195-the-mystery-of-arduino-a-arm.html

We will be using the *Arduino* integrated development environment (IDE).

Thus the phrase "Arduino IDE" (with no modifiers attached) functions as a noun, a reference to a software program.
But of course, being humans, we usually say something like, "start Arduino," just as we would say, "start MS-Word."

We will be using the *Arduino Uno* hardware platform. Of course, this is usually just called, "an Arduino."

When somebody says "Arduino," what is meant needs to be somewhat pulled from the context.

31

Let's take a look at a parallel port I/O operation on an Arduino Uno board.

The Arduino Uno is programmed in a language that Arduino (company) calls *wiring*. (Who knows why that name!)

The Arduino IDE is written in Java. But the *wiring* language used on the Arduino boards is a dialect of C/C++
It is hardly a dialect. Pretty much everything in C/C++ works.
The most obvious difference is that the user of the IDE does not get to write the `main` procedure. It is a given.

Your generic C "hello world" program looks like this:

```
#include <stdio.h>
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

But the Arduino IDE has a built-in `main` program that you never see and cannot change. It is something like. . .

```
#include <stdio.h>
#include <stuff_to_set_internal_variables_dependent_on_your_Arduino_hardware>
void main() {
    check_for_download();
    setup();                    //The user must supply the setup() code
    while (1) {
       check_for_download();
       loop();                  //The user must supply the loop() code
    }
    return 0;
}
```

The user does not get to write `main()`. You must confine your code to `setup()` and `loop()`.

32

Let's take a look at a parallel port I/O operation on an Arduino Uno board.

But the Arduino IDE has a built-in `main` program that you never see and cannot change. It is something like. . .

```
#include <stdio.h>
#include <stuff_to_set_internal_variables_dependent_on_your_Arduino_hardware>
void main() {
    check_for_download();
    setup();                  //The user must supply the setup() code
    while (1) {
       check_for_download();
       loop();                //The user must supply the loop() code
    }
    return 0;
}
```

The user does not get to write `main()`. You must confine your code to `setup()` and `loop()`.

Additionally, `main()` defines a bunch of variables for you to help you relate your code to the hardware.

Additionally, the IDE gives you immediate access to a bunch of procedures like `delay`, `digitalwrite`, `digitalread`. . . Thus the "wiring" language is really just C/C++ in a wrapper.

33

```
/*Blink
  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control.  On the UNO, MEGA and ZERO it is attached to digital
  pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to the correct LED pin independent of which board is
  used.  If you want to know what pin the on-board LED is connected to on your Arduino model, check the
  Technical Specs of your board at:  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

34

`LED_BUILTIN`, `OUTPUT`, `HIGH`, and `LOW` are variables provided by `main()` and standardized within the Arduino IDE.

```
/*Blink, sans the header comments

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}



// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

If you use the IDE to send a new program to your Arduino Uno board, then after a completion of the `loop()` procedure the `main()` program will notice the demand to download and do that. At the end of the download it will reset the Arduino Uno. The new `setup()` will run one time, then the new `loop()` will run over and over, etc.

This is assuming that you have not programmed or wired something to prevent communicating the download request!

35

`LED_BUILTIN`, `OUTPUT`, `HIGH`, and `LOW` are variables provided by `main()` and standardized within the Arduino IDE.

```
/*Blink, sans the header comments

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}



// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

If you use the IDE to send a new program to your Arduino Uno board, then after a completion of the `loop()` procedure the `main()` program will notice the demand to download and do that. At the end of the download it will reset the Arduino Uno. The new `setup()` will run one time, then the new `loop()` will run over and over, etc.

This is assuming that you have not programmed or wired something to prevent communicating the download request!

An example of **Blind-Cycle I/O**

36

Where is this parallel I/O hardware on the Arduino?

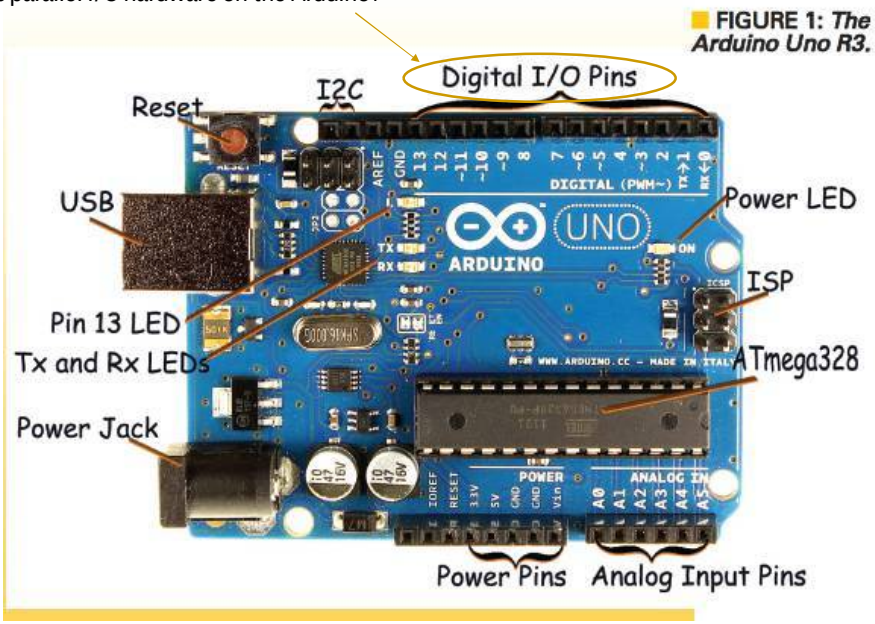


Illustration ©Joe Pardue, Used by Permission

37