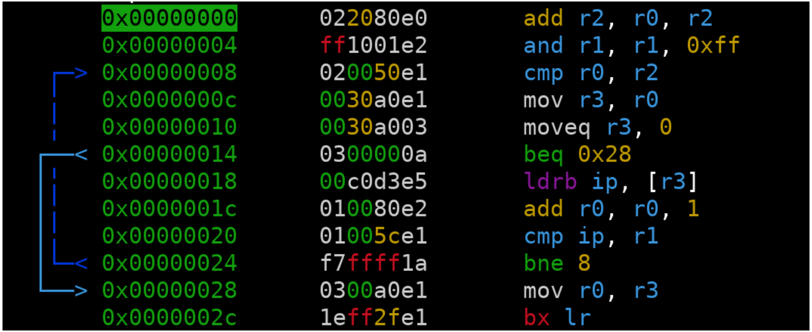
ENGR-E 399/599: Embedded systems reverse engineering

Lecture 7: I/O interfaces

Austin Roach
ahroach@iu.edu

February 23, 2022

Mystery function



```
0x00000000    022080e0    add r2, r0, r2
0x00000004    ff1001e2    and r1, r1, 0xff
0x00000008    020050e1    cmp r0, r2
0x0000000c    0030a0e1    mov r3, r0
0x00000010    0030a003    moveq r3, 0
0x00000014    0300000a    beq 0x28
0x00000018    00c0d3e5    ldrb ip, [r3]
0x0000001c    010080e2    add r0, r0, 1
0x00000020    01005ce1    cmp ip, r1
0x00000024    f7ffff1a    bne 8
0x00000028    0300a0e1    mov r0, r3
0x0000002c    1eff2fe1    bx lr
```

- How many arguments does the function take?
- What are the types of the arguments?
- What does the function do?
- What does the function return?

Mystery function revealed

```
void *memchr(const void *s, int c, size_t n)
```

Description

The `memchr()` function scans the initial `n` bytes of the memory area pointed to by `s` for the first instance of `c`. Both `c` and the bytes of the memory area pointed to by `s` are interpreted as unsigned char.

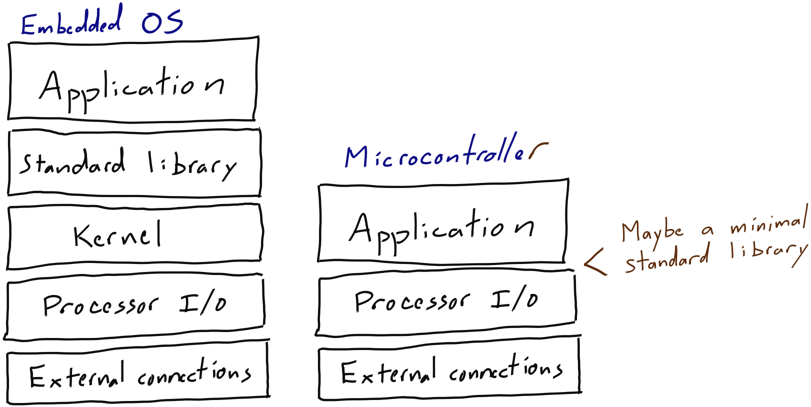
Return value

The `memchr()` function returns a pointer to the matching byte or NULL if the character does not occur in the given memory area.

Today's plan

- AVR I/O interfaces
  - Why I/O interfaces matter
  - Overview of some I/O interfaces
  - Connecting I/O in firmware to the physical world
  - I/O interfaces in Ghidra
- Automating interaction with an I/O interface
  - Motivation: Cryptographic hash functions
  - Automating UART interaction
    - pySerial
- Bonus: Bootloader reverse-engineering

Software stacks



Microcontroller applications lack standard library/kernel interface to abstract away I/O operations

Why I/O matters

- I/O operations define how the microcontroller interacts with the world
  - How information enters into the microcontroller
  - How the microcontroller influences the external world
- Inputs and outputs are fundamental to understanding functionality

AVR memory-mapped/port-mapped I/O overview

- All I/O registers can be accessed from the data memory space
  - Accessed using load/store instructions
- Some I/O registers can also be accessed using special I/O instructions
  - `IN`, `OUT`, `SBI`, `CBI`, `SBIC`, `SBIS`

General purpose (digital) I/O

- Control individual I/O pins
- Arranged into ‘Ports’ of 8 pins
  - Fits 8-bit register/memory widths
- Can be set as inputs or outputs
- Outputs can be driven high or low
- Inputs can be set with or without pull-up resistors
- *Each pin is individually configurable*

ATmega 328p datasheet, Section 18: I/O ports

DDRx registers

- DDRxn bit set (1): Port x Pin n is an output
- DDRxn bit cleared (0): Port x Pin n is an input

ATmega 328p datasheet, Section 18.2.1: Configuring the Pin

PORTx registers

When Port x Pin n is an output:

- PORTxn bit set (1): Port x Pin n is driven high
- PORTxn bit clear (0): Port x Pin n is driven low

When Port x Pin n is an input:

- PORTxn bit set (1): Port x Pin n has internal pull-up resistor activated
- PORTxn bit clear (0): Port x Pin n has internal pull-up resistor disabled

ATmega 328p datasheet, Section 18.2.1: Configuring the Pin

PINx registers

- Writing ‘1’ toggles the state of PORTxn, whether it is an input or output
- Reading reads the state of input pins

ATmega 328p datasheet, Section 18.2.2: Toggling the Pin
ATmega 328p datasheet, Section 18.2.2: Reading the Pin Value

Example: victory() function LED controls

Control interfaces

- Not every interface can be controlled by bit-banging GPIO pins
- Some registers control hardware blocks implementing more complicated interfaces or microcontroller features

UART control registers

- USART I/O Data Register 0 (`UDR0`)
  - Transmit buffer (`TXB`) accessed by write
  - Receive buffer (`RXB`) accessed by read
- USART Control and Status Register 0 A (`UCSR0A`)
- USART Control and Status Register 0 B (`UCSR0B`)
- USART Control and Status Register 0 C (`UCSR0C`)
- USART Baud Rate 0 Register Low (`UBRR0L`)
- USART Baud Rate 0 Register High (`UBRR0H`)

ATmega 328p datasheet, Section 24.12: USART Register Description

Example: UART interaction in uart_intro.bin

Example: Timer interaction in uart_intro.bin

I/O multiplexing

- More I/O interfaces than available physical pins
- Pin assignments are shared
- Functionality depends on configuration

ATmega 328p datasheet, Section 6: I/O Multiplexing

Where these signals go: schematic review

Ghidra .pspec file

```
...
<symbol name="PINB" address="mem:0x23"/>
<symbol name="DDRB" address="mem:0x24"/>
<symbol name="PORTB" address="mem:0x25"/>
<symbol name="PINC" address="mem:0x26"/>
<symbol name="DDRC" address="mem:0x27"/>
<symbol name="PORTC" address="mem:0x28"/>
<symbol name="PIND" address="mem:0x29"/>
<symbol name="DDRD" address="mem:0x2a"/>
...
```

Cross-references in Ghidra

- See what parts of code are reading or writing to an address
- Determine which parts of a code interact with a particular interface

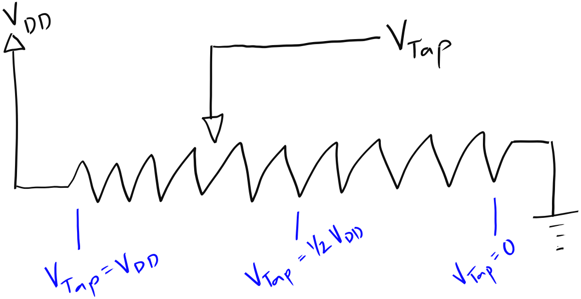
This only works if code is properly disassembled, and computed addresses are reconstructed through data-flow analysis

When cross-references fail

In `arduino_uno_blink.bin`:

- GPIO configuration (Port B Pin 5) starting at code:1a2
- GPIO configuration (Port B Pin 5) in function at code:070

A note on potentiometers...



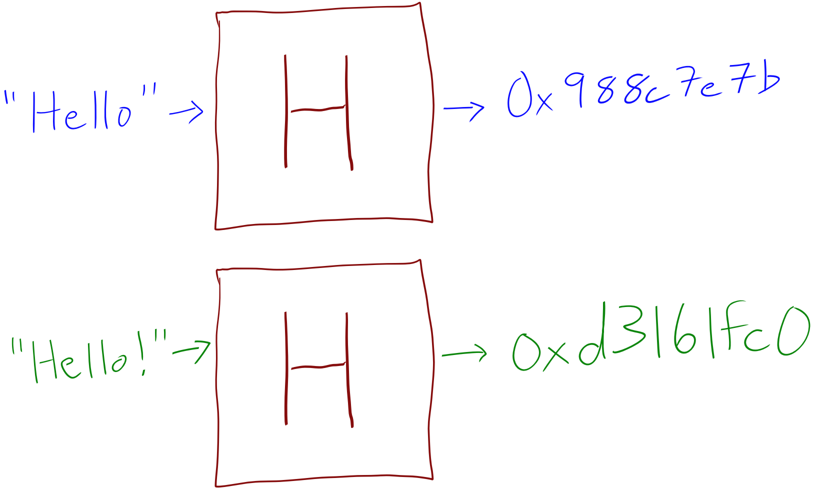
A potentiometer can act like a voltage divider!

DON'T SHORT IT OUT

Please don't connect power to one terminal, ground to the tap, and then crank it all the way!

Automating interaction

Hash function walkthrough



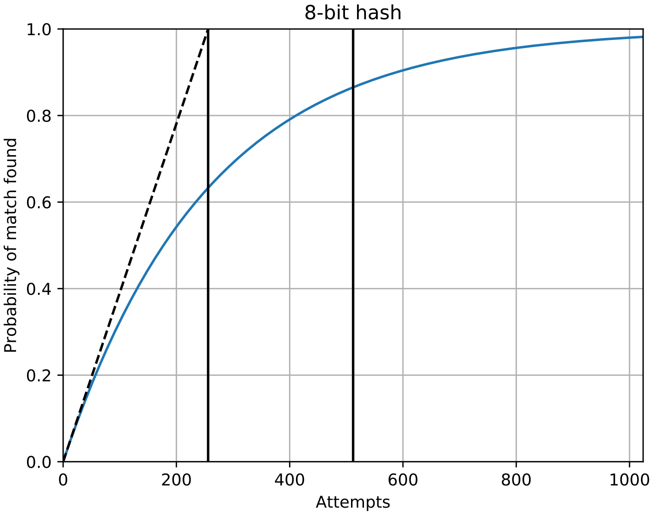
Cryptographic hash function properties

- Deterministic
- Infeasible to find a message that produces a given hash (preimage resistance)
- Infeasible to find two different messages with the same hash value (collision resistance)
- A small change should result in a large change to the hash value (avalanche effect)

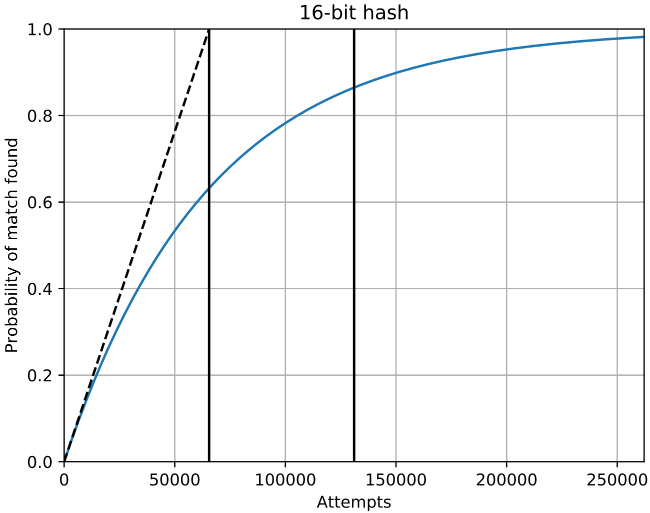
How many possible hash values are there?

- 8-bit hash: 256
- 16-bit hash: 65,536
- 24-bit hash: 16,777,216
- 32-bit hash: 4,294,967,296
- 128-bit hash: 340,282,366,920,938,463,463,374,607,431,768,211,456
- 160-bit hash: 1,461,501,637,330,902,918,203,684,832,716,283,019,655,932,542,976
- 256-bit hash: 115,792,089,237,316,195,423,570,985,008,687,907,853,269,984,665,640,564,039,457,584,007,913,129,639,936

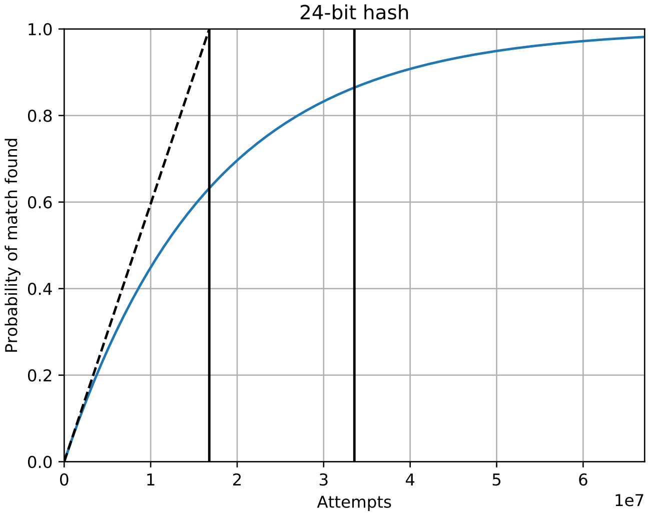
Cumulative probability of finding a match: 8-bit hash



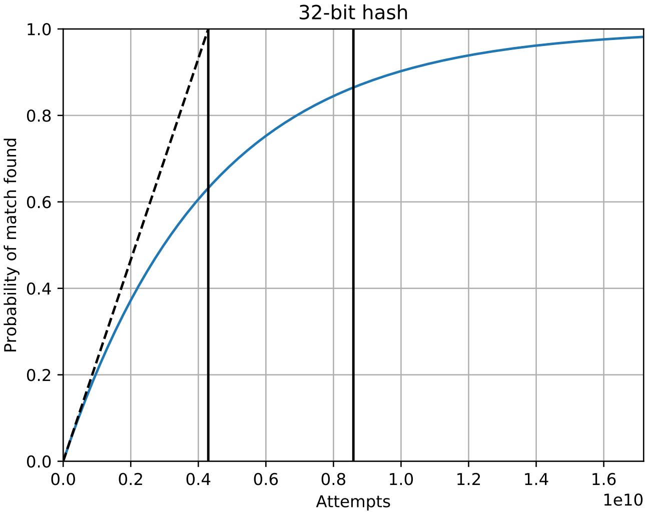
Cumulative probability of finding a match: 16-bit hash



Cumulative probability of finding a match: 24-bit hash



Cumulative probability of finding a match: 32-bit hash



Approach

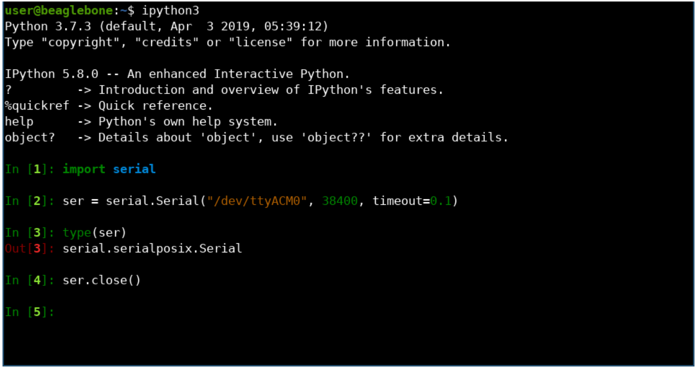
- Working backward to find input is infeasible
- The best we can do is brute force
- No one wants to do this by hand

We need to automate our interaction with the program to find a matching hash

Automating embedded systems interactions

1. Need a way to provide input to the system
   - UART
2. Need a way to determine the system state
   - Feedback messages on UART
3. Need a way to mutate inputs
   - Program that you will write

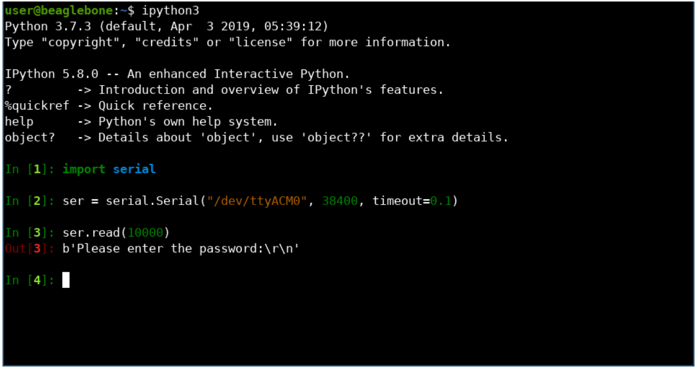
python3-serial: Initializing a serial object



```
user@beaglebone:~$ ipython3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
Type "copyright", "credits" or "license" for more information.

IPython 5.8.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import serial

In [2]: ser = serial.Serial("/dev/ttyACM0", 38400, timeout=0.1)

In [3]: type(ser)
Out[3]: serial.serialposix.Serial

In [4]: ser.close()

In [5]:
```

- Must set baud rate
- You probably want to set a read timeout

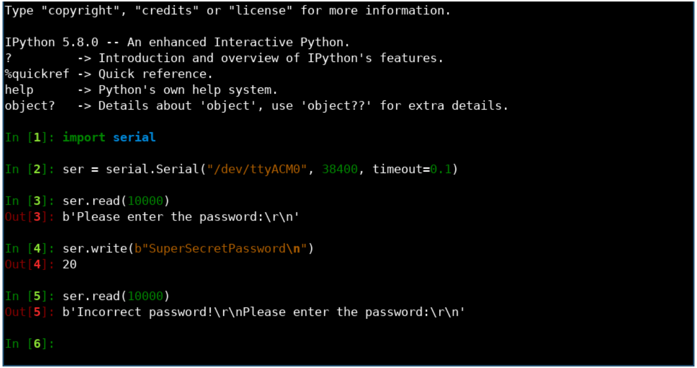
python3-serial: Reading



```
user@beaglebone:~$ ipython3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
Type "copyright", "credits" or "license" for more information.

IPython 5.8.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import serial

In [2]: ser = serial.Serial("/dev/ttyACM0", 38400, timeout=0.1)

In [3]: ser.read(10000)
Out[3]: b'Please enter the password:\r\n'

In [4]: 
```

- Either read exactly the number of characters that you want
- Or read a *bunch* of characters with the timeout set

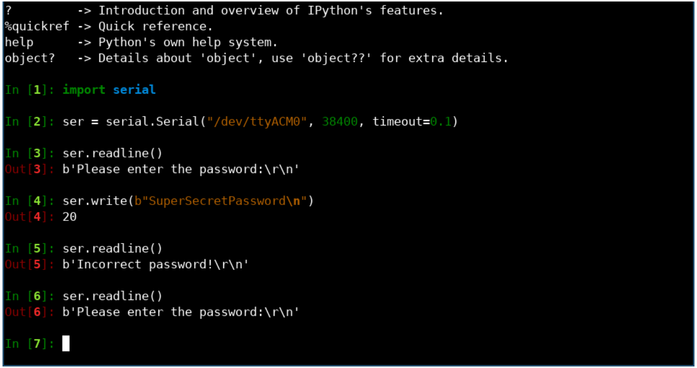
python3-serial: Writing



```
Type "copyright", "credits" or "license" for more information.

IPython 5.8.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import serial

In [2]: ser = serial.Serial("/dev/ttyACM0", 38400, timeout=0.1)

In [3]: ser.read(10000)
Out[3]: b'Please enter the password:\r\n'

In [4]: ser.write(b"SuperSecretPassword\n")
Out[4]: 20

In [5]: ser.read(10000)
Out[5]: b'Incorrect password!\r\nPlease enter the password:\r\n'

In [6]:
```

- Don't forget a newline!

python3-serial: readline() method



```
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import serial

In [2]: ser = serial.Serial("/dev/ttyACM0", 38400, timeout=0.1)

In [3]: ser.readline()
Out[3]: b'Please enter the password:\r\n'

In [4]: ser.write(b"SuperSecretPassword\n")
Out[4]: 20

In [5]: ser.readline()
Out[5]: b'Incorrect password!\r\n'

In [6]: ser.readline()
Out[6]: b'Please enter the password:\r\n'

In [7]: 
```

- Might be more convenient for our newline-terminated responses

Bonus: bootloader reverse-engineering

Example: Bootloaders

- Common in microcontroller-based systems
- Allow code to be updated without attaching a programming device
- Reset settings encoded in fuses enable bootloader behavior
- For some fuse settings, code readout may be disabled for programmer but allowed through bootloader

Let's look at the factory-default Arduino Uno 'blink' program

Default fuse settings

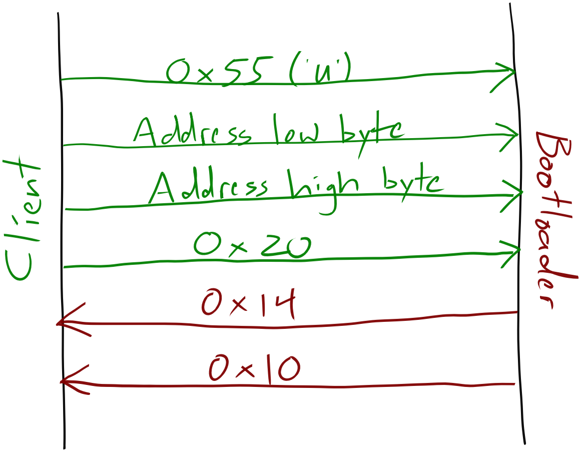
HFUSE: 0xde

- BOOTSZ1 = 1
- BOOTSZ0 = 1
- BOOTRST = 0

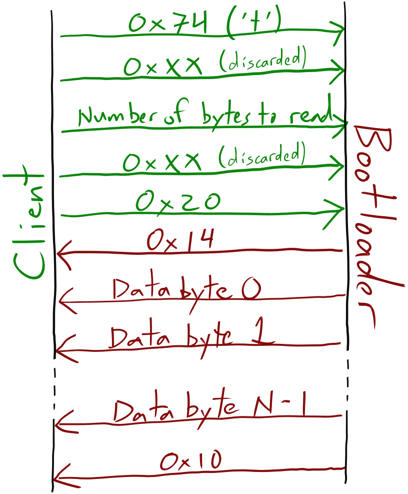
BOOTSZ values: Boot Flash section addresses 0x3f00-0x3fff

BOOTRST value: Reset vector moved to start of Boot Flash section

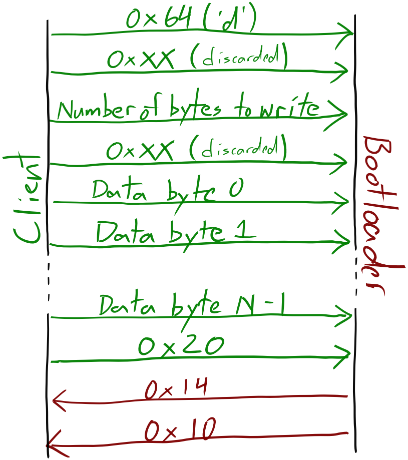
Set program address



Read program memory



Write program memory



Bootloader client demo