Welcome to the Scaffold Workshop!

hardwear.io 2019

Karim Abdellatif
Olivier Heriveaux

Hardware Wallet Manufacturer

Donjon
Ledger's Security Research Team

You will need **Python3**
with the packages pyserial, progressbar2, scaffold

```
git clone https://github.com/Ledger-Donjon/scaffold.git
cd scaffold/api && pip3 install .
pip3 install pyserial progressbar2
```

... or ...

Get the **VirtualBox** virtual machine from the USB keys

Required Hardware:

- A Scaffold board
- A breadboard daughterboard
- An ATMEGA328 microcontroller
- Some wires

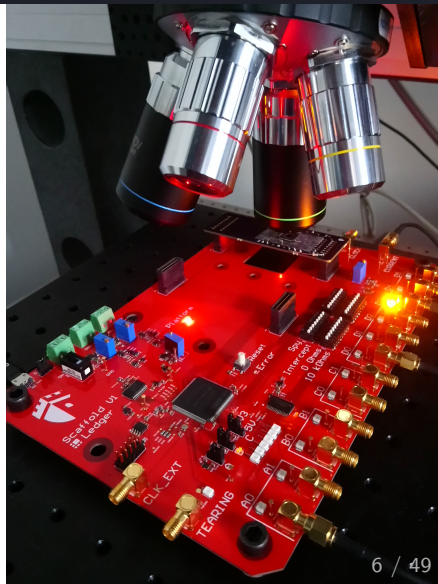
THE SCAFFOLD BOARD

Designed for the security evaluation of
Integrated Circuits

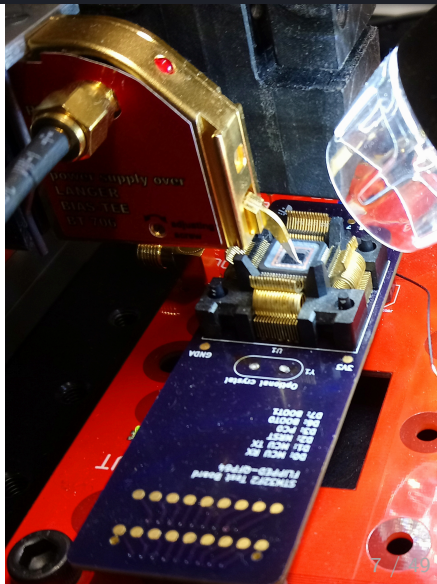
- Versatility
- Fast attack scripts development
- Integration with other equipments / instruments

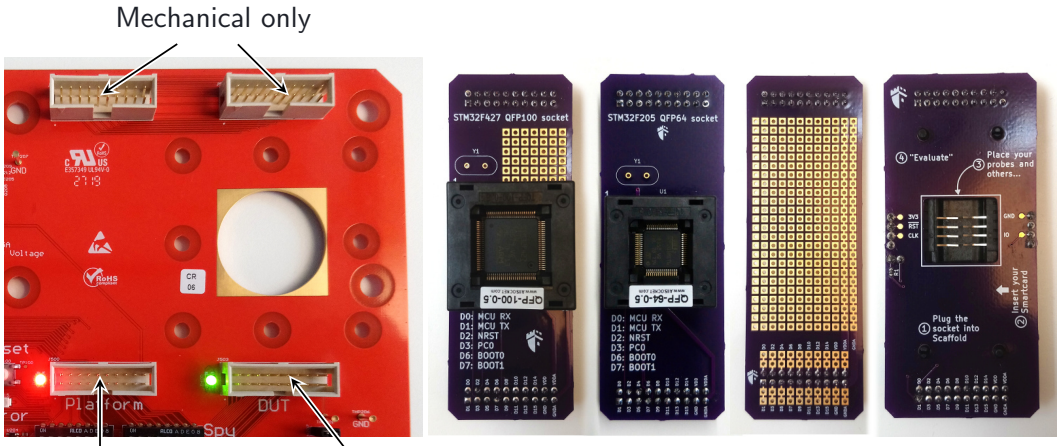
Open-Source and Open-Hardware
https://github.com/Ledger-Donjon/scaffold

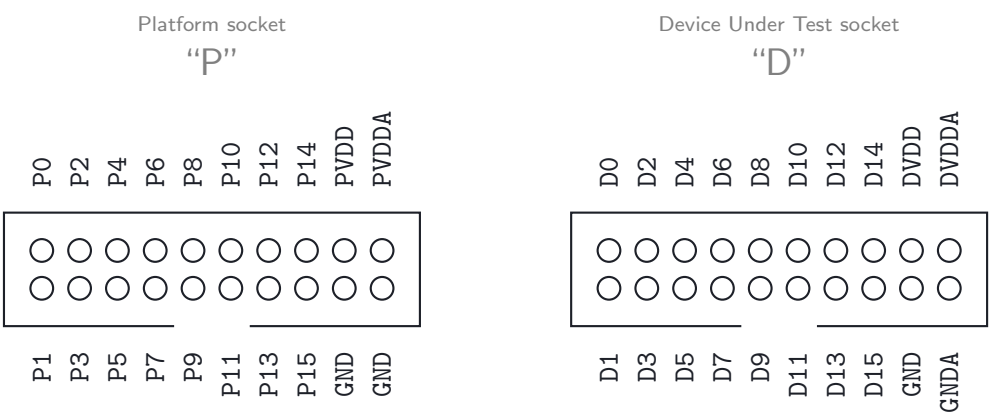
Used on all our security lab test benches:
Side-Channels, Glitches, Laser Fault Injection...

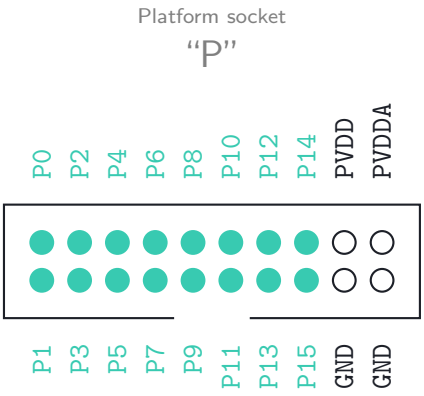


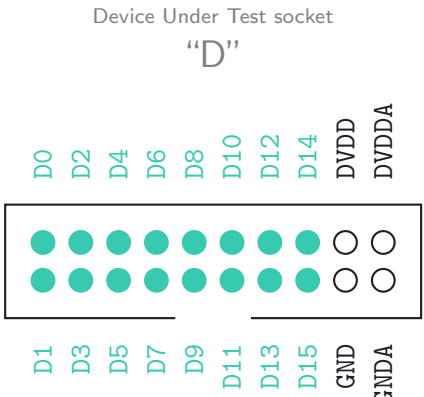
- Sockets for daughterboards
- Communication peripherals:
  UART, I^{2}C, SPI, ISO7816
- Triggering
- Pulse Generators with 10 ns resolution
- Measurement resistor with analog amplifier
- 4 x AlphaNov laser source compatible outputs
- On-board adjustable power supplies
- FPGA protections against EMFI
- An awesome Python3 API




Mechanical only
Platform
DUT
STM32F427 QFP100 socket
STM32F205 QFP64 socket
D0: MCU RX
D1: MCU TX
D2: NRST
D3: PC0
D6: BOOT0
D7: BOOT1
"Evaluate"
Place your probes and others...
Insert your Smartcard
Plug the socket into Scaffold
Platform socket
(Not used today)
Device Under Test socket
(We will use this one!)

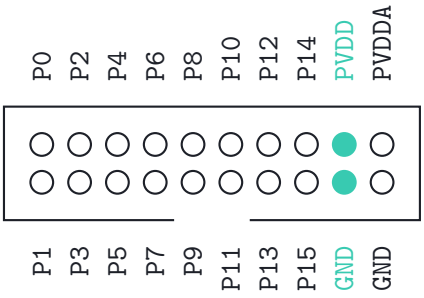

Platform socket
"P"
P0 P2 P4 P6 P8 P10 P12 P14 PVDD PVDDA
P1 P3 P5 P7 P9 P11 P13 P15 GND GND
Device Under Test socket
"D"
D0 D2 D4 D6 D8 D10 D12 D14 DVDD DVDDA
D1 D3 D5 D7 D9 D11 D13 D15 GND GNDA


Platform socket
“P”
P0
P2
P4
P6
P8
P10
P12
P14
PVDD
PVDDA
P1
P3
P5
P7
P9
P11
P13
P15
GND
GND


Device Under Test socket
“D”
D0
D2
D4
D6
D8
D10
D12
D14
DVDD
DVDDA
D1
D3
D5
D7
D9
D11
D13
D15
GND
GNDA

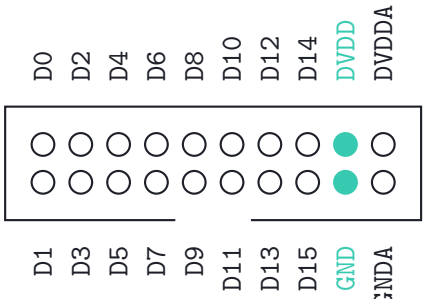
Platform socket

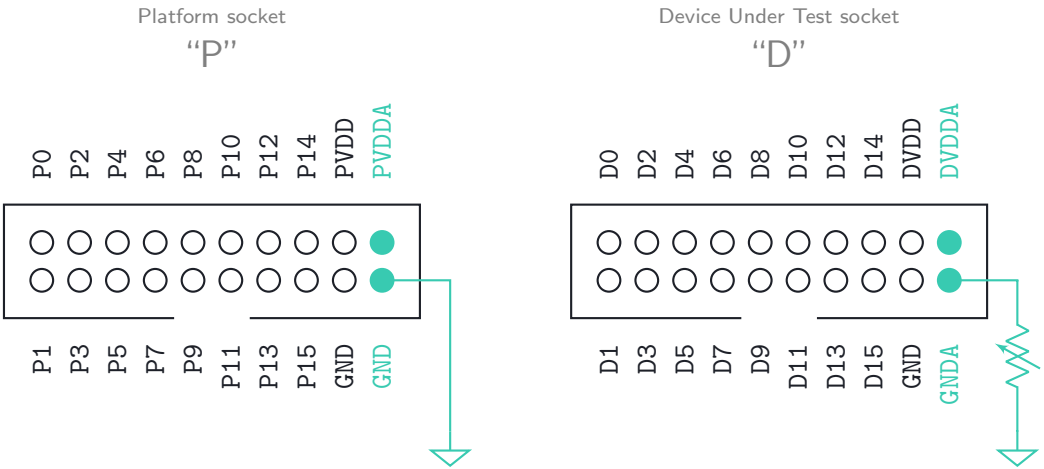
"P"

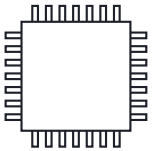


Device Under Test socket

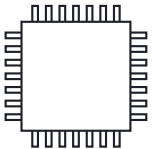
"D"

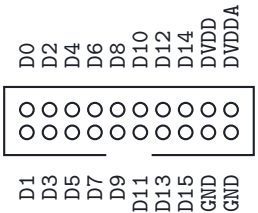



Platform socket
“P”
P0
P2
P4
P6
P8
P10
P12
P14
PVDD
PVDDA
P1
P3
P5
P7
P9
P11
P13
P15
GND
GND
Device Under Test socket
“D”
D0
D2
D4
D6
D8
D10
D12
D14
DVDD
DVDDA
D1
D3
D5
D7
D9
D11
D13
D15
GND
GNDA

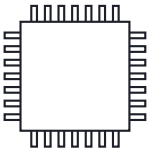

D0
D2
D4
D6
D8
D10
D12
D14
DVDD
DVDDA
D1
D3
D5
D7
D9
D11
D13
D15
GND
GND

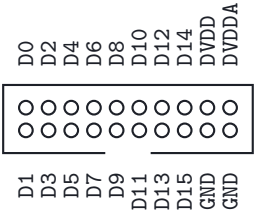
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
```



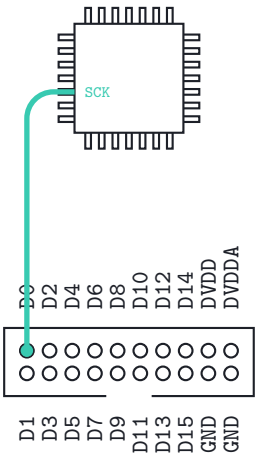


```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
```

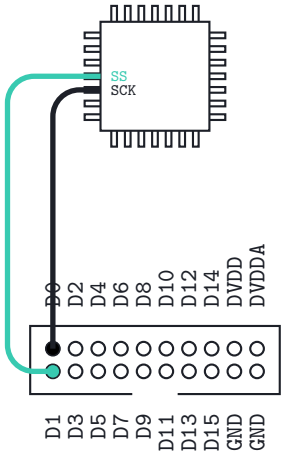




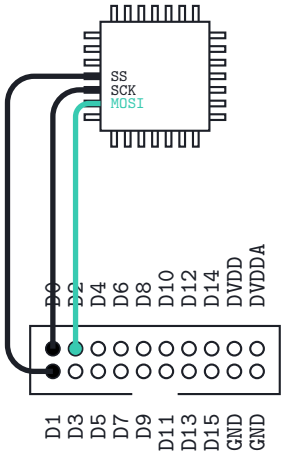
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
```



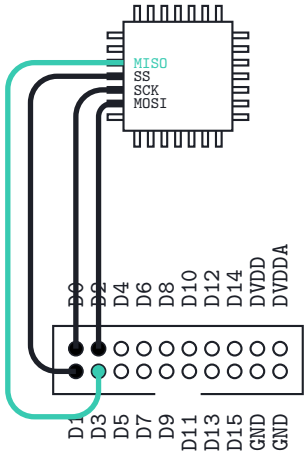
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
```



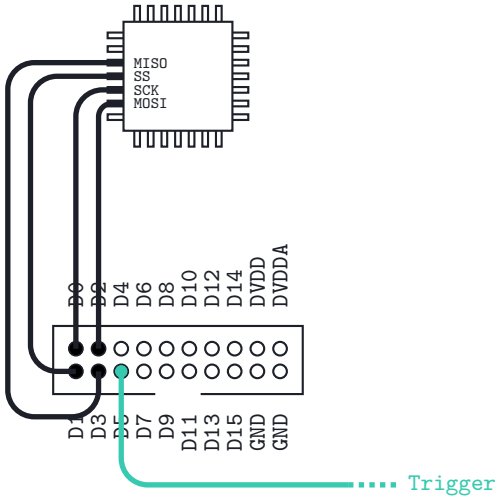
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
```



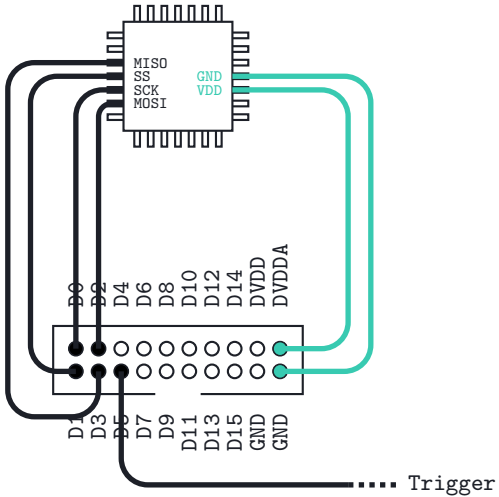
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
spi.miso << scaffold.d3
```



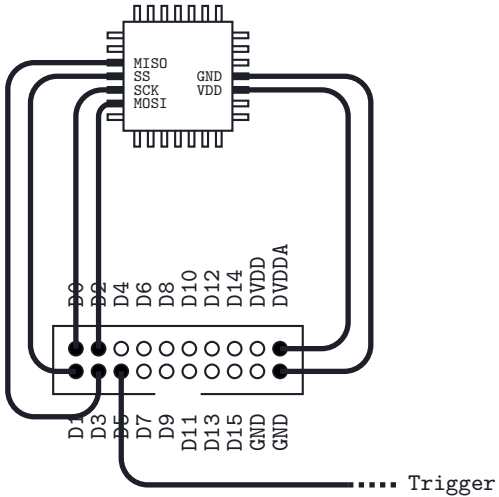
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
spi.miso << scaffold.d3
scaffold.d5 << spi.trigger
```



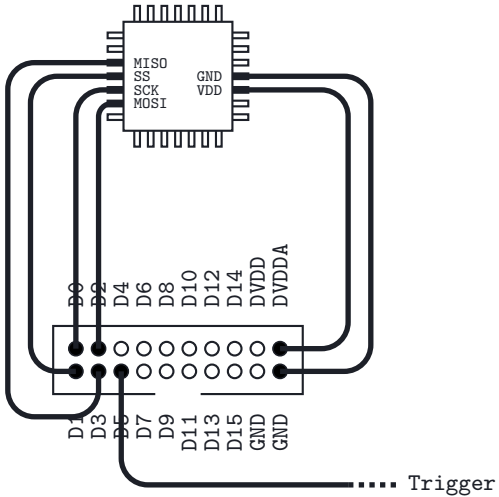
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
spi.miso << scaffold.d3
scaffold.d5 << spi.trigger
```



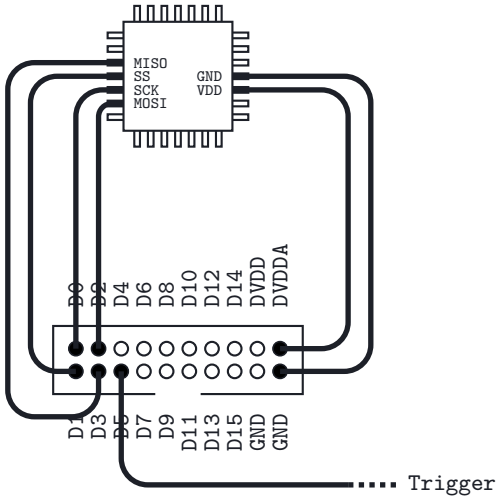
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
spi.miso << scaffold.d3
scaffold.d5 << spi.trigger
scaffold.power.dut = 1
```



```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
spi.miso << scaffold.d3
scaffold.d5 << spi.trigger
scaffold.power.dut = 1
spi.transmit(0xaa)
```



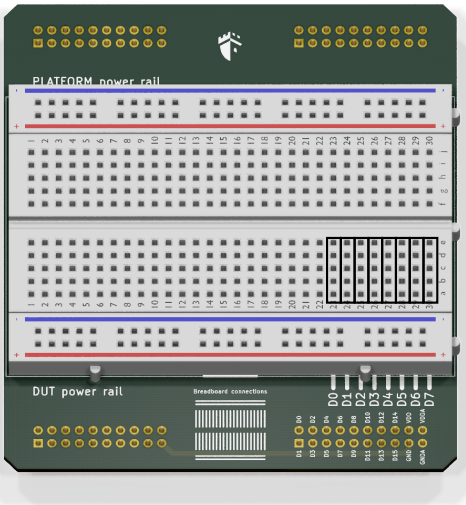
```
from scaffold import *
scaffold = Scaffold('/dev/ttyUSB0')
spi = scaffold.spi0
spi.sck >> scaffold.d0
spi.ss >> scaffold.d1
spi.mosi >> scaffold.d2
spi.miso << scaffold.d3
scaffold.d5 << spi.trigger
scaffold.power.dut = 1
spi.transmit(0xaa)
spi.transmit(0xbb, trigger=True)
```



LET’S TRY OUT

Platform power supply
(We won't need it)

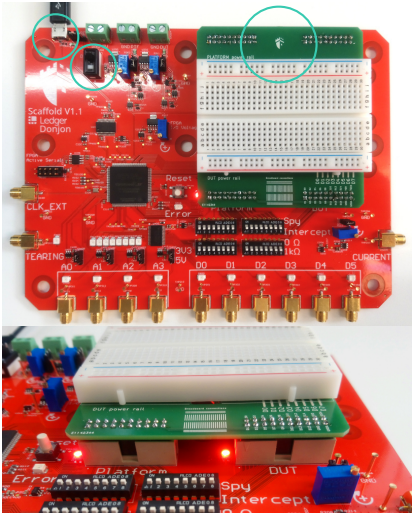
DUT power supply
(We will use this one)



D0 to D7 columns

1. Plug-in the daughterboard
   Donjon Shield up!
2. Connect the Scaffold board to the computer using the USB cable
3. Power-on the board
   Yellow LEDs should flash

DO NOT change board settings!
(jumpers, potentiometers, dip-switches)



Run the test script:

```
cd /home/user/workshop/
python3 scripts/connect.py -d /dev/ttyUSB0 (Linux users)
python3 scripts/connect.py -d COM1 (Windows users)
```

You should see:

```
Yeah it works!
Board arch version:  0.7
```

On linux you may need to add rights to access to `/dev/ttyUSB0`:

```
sudo addgroup your_username dialout && sudo reboot
```

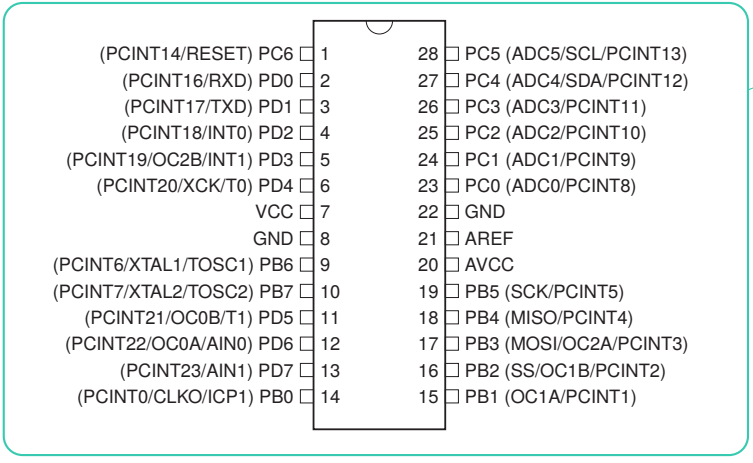
ATMEGA PLAYGROUND

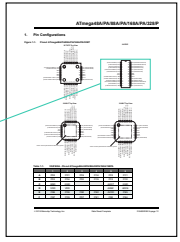
We want to:

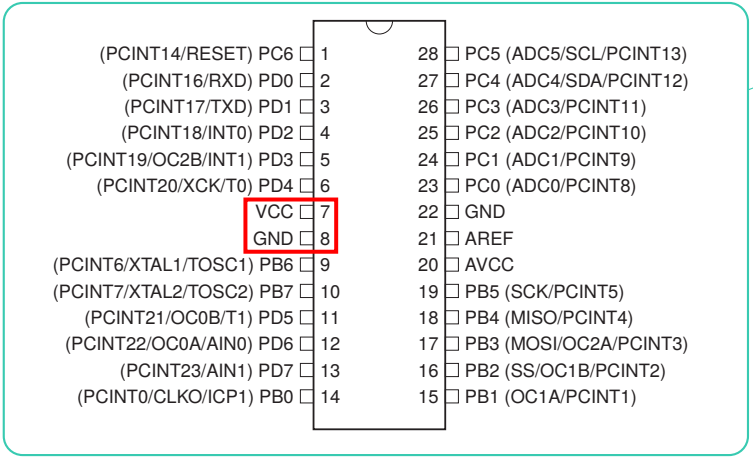
- have an ATMEGA328 microcontroller
- run it from an external clock
- being able to load code from Scaffold

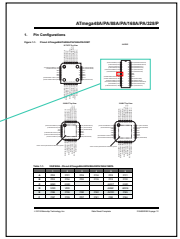
We will need to connect:

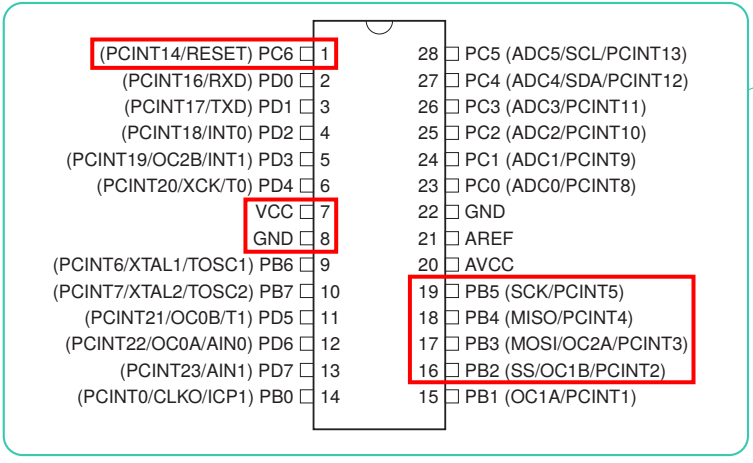
- The ATMEGA328 microcontroller
- Its power supply
- A clock source
- Programming pins connected to Scaffold

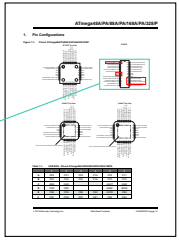

(PCINT14/RESET) PC6 1
(PCINT16/RXD) PD0 2
(PCINT17/TXD) PD1 3
(PCINT18/INT0) PD2 4
(PCINT19/OC2B/INT1) PD3 5
(PCINT20/XCK/T0) PD4 6
VCC 7
GND 8
(PCINT6/XTAL1/TOSC1) PB6 9
(PCINT7/XTAL2/TOSC2) PB7 10
(PCINT21/OC0B/T1) PD5 11
(PCINT22/OC0A/AIN0) PD6 12
(PCINT23/AIN1) PD7 13
(PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK/PCINT5)
18 PB4 (MISO/PCINT4)
17 PB3 (MOSI/OC2A/PCINT3)
16 PB2 (SS/OC1B/PCINT2)
15 PB1 (OC1A/PCINT1)

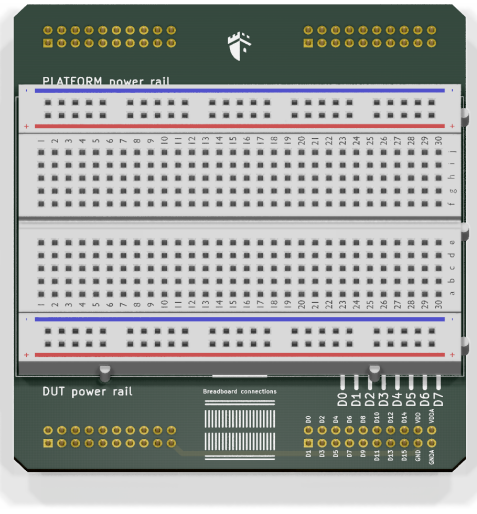



(PCINT14/RESET) PC6 1
(PCINT16/RXD) PD0 2
(PCINT17/TXD) PD1 3
(PCINT18/INT0) PD2 4
(PCINT19/OC2B/INT1) PD3 5
(PCINT20/XCK/T0) PD4 6
VCC 7
GND 8
(PCINT6/XTAL1/TOSC1) PB6 9
(PCINT7/XTAL2/TOSC2) PB7 10
(PCINT21/OC0B/T1) PD5 11
(PCINT22/OC0A/AIN0) PD6 12
(PCINT23/AIN1) PD7 13
(PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK/PCINT5)
18 PB4 (MISO/PCINT4)
17 PB3 (MOSI/OC2A/PCINT3)
16 PB2 (SS/OC1B/PCINT2)
15 PB1 (OC1A/PCINT1)



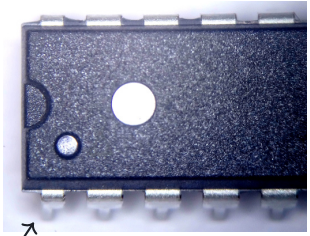

(PCINT14/RESET) PC6 1
(PCINT16/RXD) PD0 2
(PCINT17/TXD) PD1 3
(PCINT18/INT0) PD2 4
(PCINT19/OC2B/INT1) PD3 5
(PCINT20/XCK/T0) PD4 6
VCC 7
GND 8
(PCINT6/XTAL1/TOSC1) PB6 9
(PCINT7/XTAL2/TOSC2) PB7 10
(PCINT21/OC0B/T1) PD5 11
(PCINT22/OC0A/AIN0) PD6 12
(PCINT23/AIN1) PD7 13
(PCINT0/CLKO/ICP1) PB0 14
28 PC5 (ADC5/SCL/PCINT13)
27 PC4 (ADC4/SDA/PCINT12)
26 PC3 (ADC3/PCINT11)
25 PC2 (ADC2/PCINT10)
24 PC1 (ADC1/PCINT9)
23 PC0 (ADC0/PCINT8)
22 GND
21 AREF
20 AVCC
19 PB5 (SCK/PCINT5)
18 PB4 (MISO/PCINT4)
17 PB3 (MOSI/OC2A/PCINT3)
16 PB2 (SS/OC1B/PCINT2)
15 PB1 (OC1A/PCINT1)



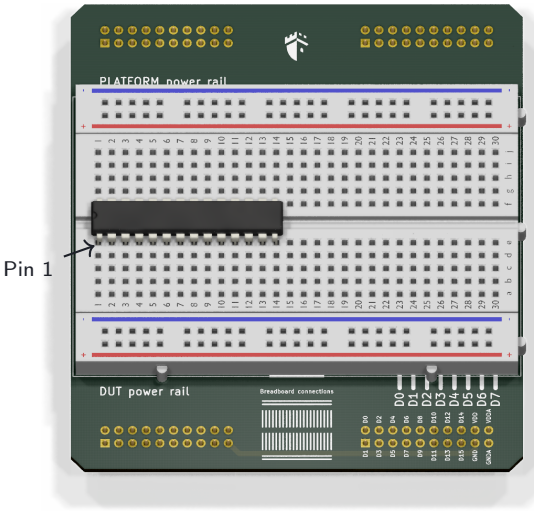

PLATFORM power rail
DUT power rail
Breadboard connections
D0
D1
D2
D3
D4
D5
D6
D7

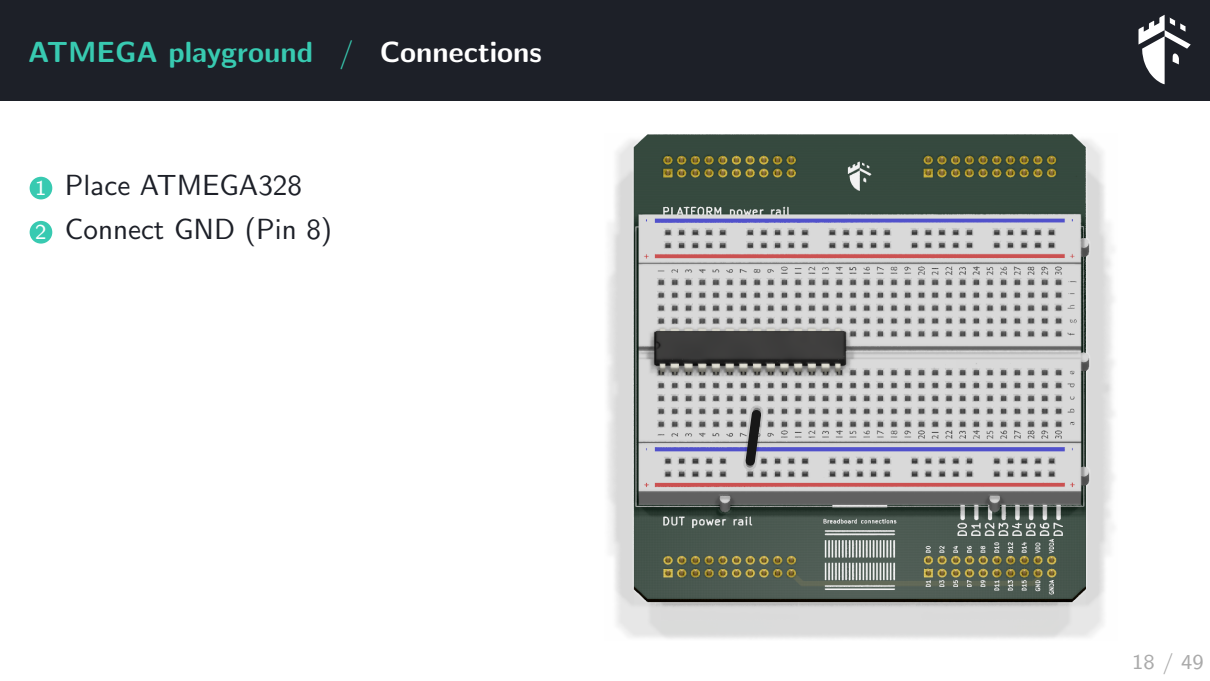
1. Place ATMEGA328

Pin 1 to the left, column 1.



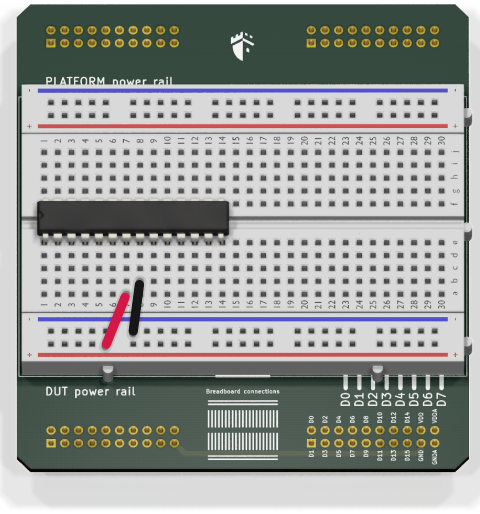
Pin 1



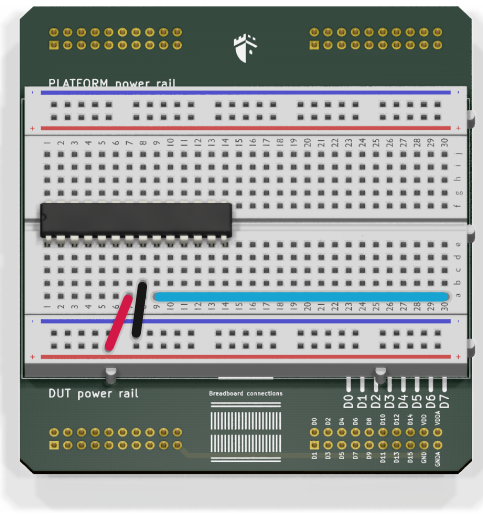


1. Place ATMEGA328
2. Connect GND (Pin 8)

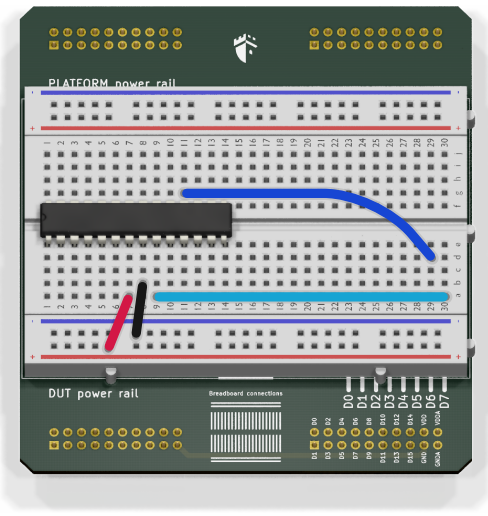
1. Place ATMEGA328
2. Connect GND (Pin 8)
3. Connect VCC (Pin 7)



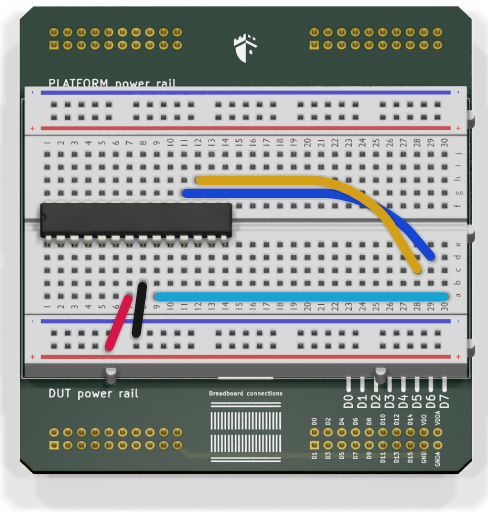
1. Place ATMEGA328
2. Connect GND (Pin 8)
3. Connect VCC (Pin 7)
4. Connect D7 to XTAL1 (Pin 9)



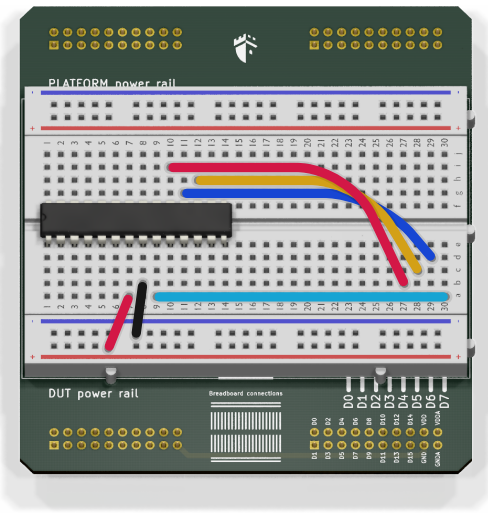
1. Place ATMEGA328
2. Connect GND (Pin 8)
3. Connect VCC (Pin 7)
4. Connect D7 to XTAL1 (Pin 9)
5. Connect D6 to MISO (Pin 18)

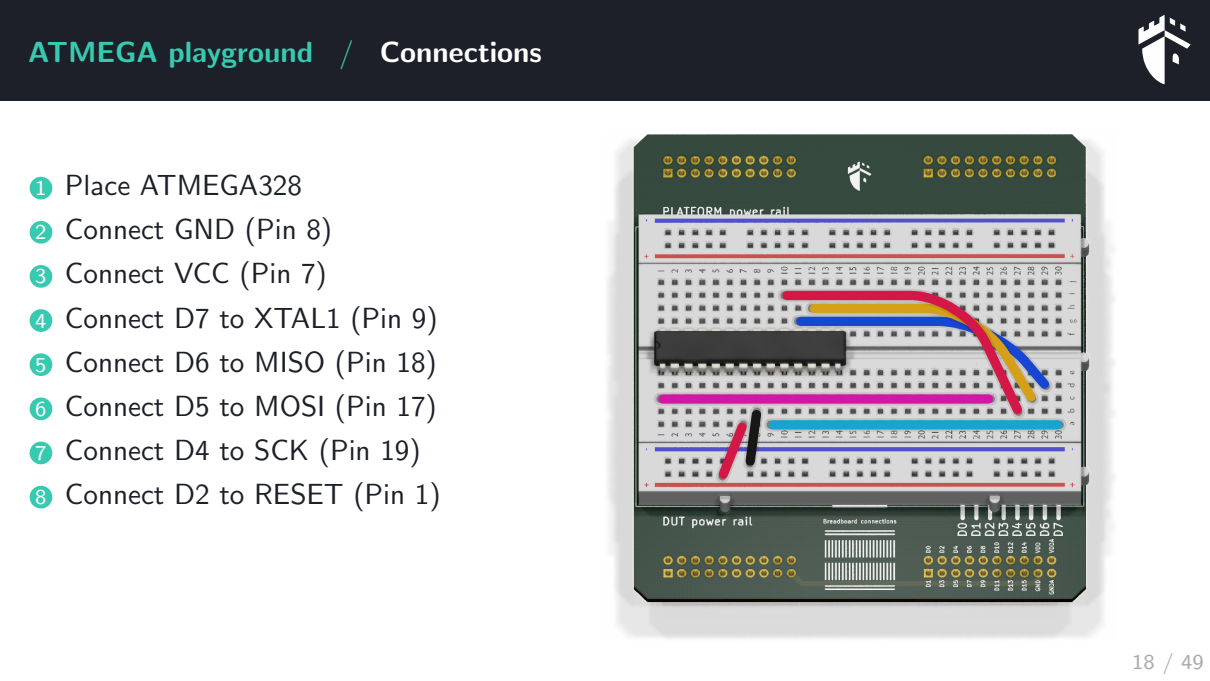


1. Place ATMEGA328
2. Connect GND (Pin 8)
3. Connect VCC (Pin 7)
4. Connect D7 to XTAL1 (Pin 9)
5. Connect D6 to MISO (Pin 18)
6. Connect D5 to MOSI (Pin 17)



1. Place ATMEGA328
2. Connect GND (Pin 8)
3. Connect VCC (Pin 7)
4. Connect D7 to XTAL1 (Pin 9)
5. Connect D6 to MISO (Pin 18)
6. Connect D5 to MOSI (Pin 17)
7. Connect D4 to SCK (Pin 19)





1. Place ATMEGA328
2. Connect GND (Pin 8)
3. Connect VCC (Pin 7)
4. Connect D7 to XTAL1 (Pin 9)
5. Connect D6 to MISO (Pin 18)
6. Connect D5 to MOSI (Pin 17)
7. Connect D4 to SCK (Pin 19)
8. Connect D2 to RESET (Pin 1)

Let's test:

```
cd /home/user/workshop/
python3 scripts/atmega.py -d COM1
```

You should see:

```
Serial programming mode enabled
Device signature:  0x1e9514
Device:  ATMEGA328
L-Fuse:  0x62
All done!
```

(This means the ATMEGA circuit has been recognized!)

CHALLENGE 1: GLITCHING A LOOP

Let's load the first challenge program:

```
cd /home/user/workshop/
python3 scripts/atmega.py -d COM1 --load challenges/ch1-loop.bin
```

This script uses the SPI peripheral of Scaffold to program the Flash memory of the microcontroller like any real programmer.
The fuses are also programmed to enable the external clock source.

The challenge program uses the UART of the microcontroller to communicate.
When the byte `0x01` is received, a loop incrementing a counter is started.
At the end of the loop, the microcontroller sends the value of the counter.

When sending over the UART:

`01` (hexa, 1 byte)

The device should respond with:

`aaaaaaaa10270000efd8ffffbbbbbbbb` (hexa, 16 bytes)

`aaaaaaaa`: response header
`10270000`: counter value (little endian). Here 10000.
`efd8ffff`: complement of counter, for checking.
`bbbbbbbb`: response footer

When sending over the UART:

`01` (hexa, 1 byte)

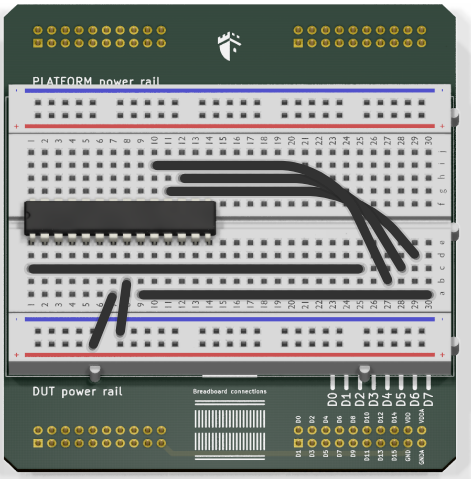
The device should respond with:

`aaaaaaaa10270000efd8ffffbbbbbbbb` (hexa, 16 bytes)

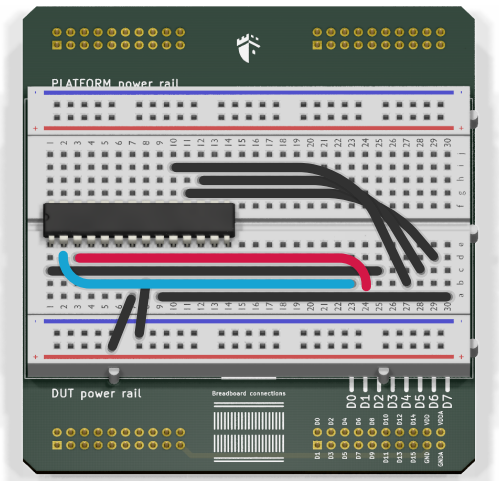
`aaaaaaaa`: response header
`10270000`: counter value (little endian). Here 10000.
`efd8ffff`: complement of counter, for checking.
`bbbbbbbb`: response footer

We need to communicate with the device using a UART.

```
int main()
{
  uart_init(9600);
  sei();

  for (;;){
    uint8_t command = uart_read_u8(); // Waits for command byte
    if (command == 0x01){
      volatile uint32_t j = 0; // Volatile to prevent compiler optimizations
      for (uint32_t i = 0; i < 10000; ++i){ // Very long loop
        j += 1;
      }
      uart_write_u32(0xaaaaaaaa); // Header
      uart_write_u32(j); // Send counter value
      uart_write_u32(j ^ 0xffffffff); // Send complement for checking
      uart_write_u32(0xbbbbbbbb); // Footer
    }
  }
}
```


PLATFORM power rail
DUT power rail
Breadboard connections
D0
D1
D2
D3
D4
D5
D6
D7

1. Connect D0 to RXD (Pin 2)
2. Connect D1 to TXD (Pin 3)



```
from scaffold import *
from binascii import hexlify

scaffold = Scaffold('/dev/ttyUSB0')
```

```
from scaffold import *
from binascii import hexlify

scaffold = Scaffold('/dev/ttyUSB0')

clock = scaffold.clock0
clock.frequency = 1e6  # Set clock frequency to 1 MHz
clock.out >> scaffold.d7  # Connect clock to ATMEGA XTAL1
```

```
from scaffold import *
from binascii import hexlify

scaffold = Scaffold('/dev/ttyUSB0')

clock = scaffold.clock0
clock.frequency = 1e6  # Set clock frequency to 1 MHz
clock.out >> scaffold.d7  # Connect clock to ATMEGA XTAL1

uart = scaffold.uart0
uart.tx >> scaffold.d0  # To ATMEGA RXD
uart.rx << scaffold.d1  # From ATMEGA TXD
```

```
from scaffold import *
from binascii import hexlify

scaffold = Scaffold('/dev/ttyUSB0')

clock = scaffold.clock0
clock.frequency = 1e6  # Set clock frequency to 1 MHz
clock.out >> scaffold.d7  # Connect clock to ATMEGA XTAL1

uart = scaffold.uart0
uart.tx >> scaffold.d0  # To ATMEGA RXD
uart.rx << scaffold.d1  # From ATMEGA TXD

# Power-on the device and wait 100 ms for the device to be ready
scaffold.power.restart_dut(ton=0.1)
```

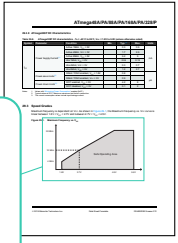
```
from scaffold import *
from binascii import hexlify

scaffold = Scaffold('/dev/ttyUSB0')

clock = scaffold.clock0
clock.frequency = 1e6  # Set clock frequency to 1 MHz
clock.out >> scaffold.d7  # Connect clock to ATMEGA XTAL1

uart = scaffold.uart0
uart.tx >> scaffold.d0  # To ATMEGA RXD
uart.rx << scaffold.d1  # From ATMEGA TXD

# Power-on the device and wait 100 ms for the device to be ready
scaffold.power.restart_dut(ton=0.1)

uart.flush()  # Discard received junk bytes
uart.transmit(b'\x01')  # Send the command to start the loop
res = uart.receive(16)  # Receive the response
print(hexlify(res).decode())  # Print the response
```

The device should respond with:

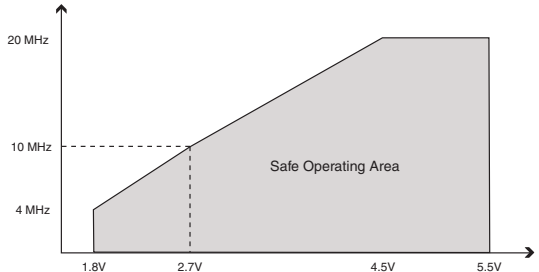
`aaaaaaaa10270000efd8ffffbbbbbbbb`

`aaaaaaaa`: response header
`10270000`: counter value (little endian). Here 10000.
`efd8ffff`: complement of counter, for checking.
`bbbbbbbb`: response footer

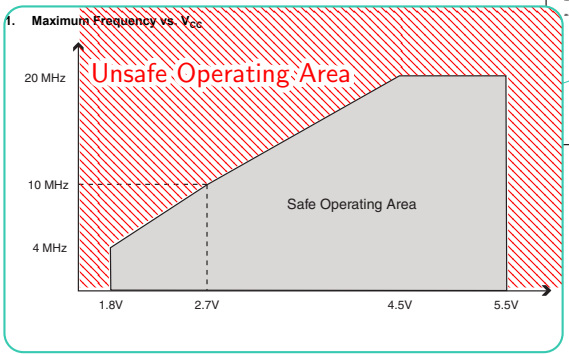
From the datasheet:
Maximum frequency: 20 MHz

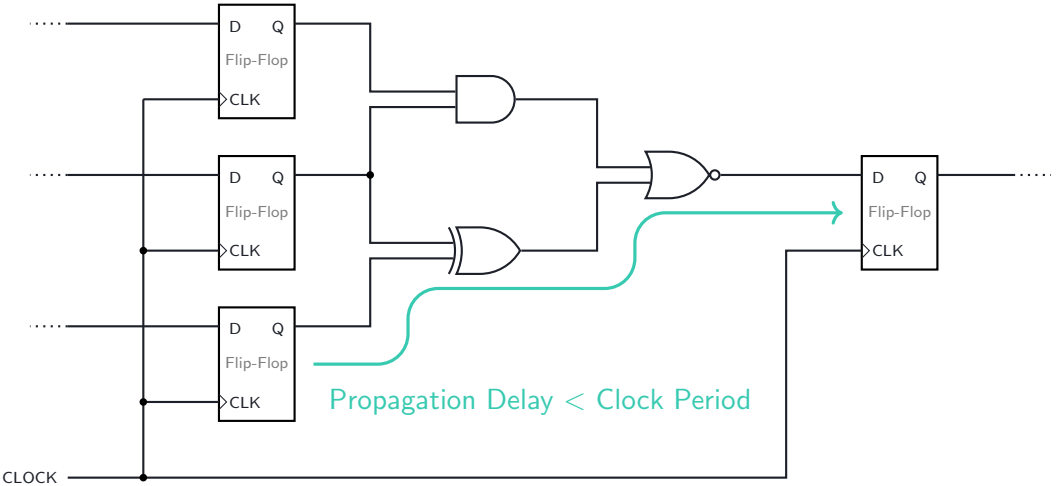


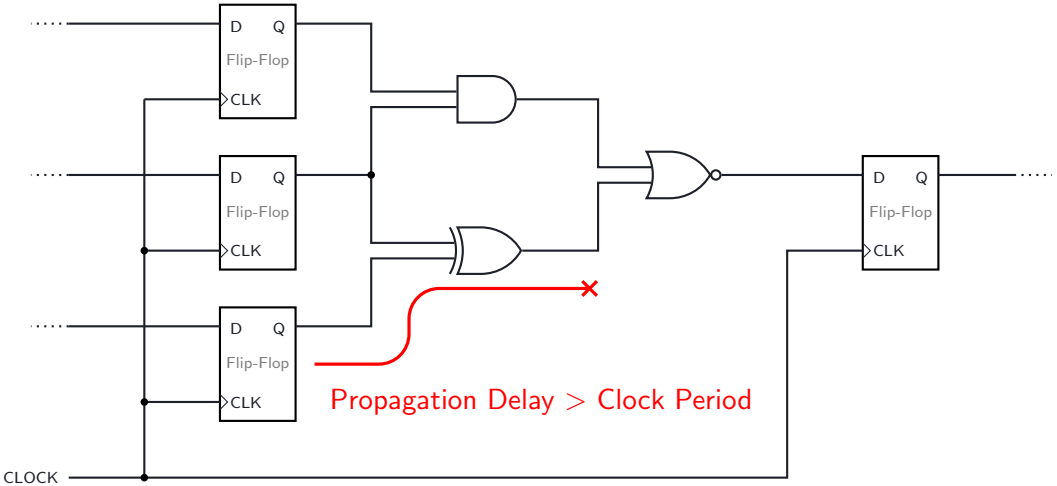
1. **Maximum Frequency vs. V_{CC}**



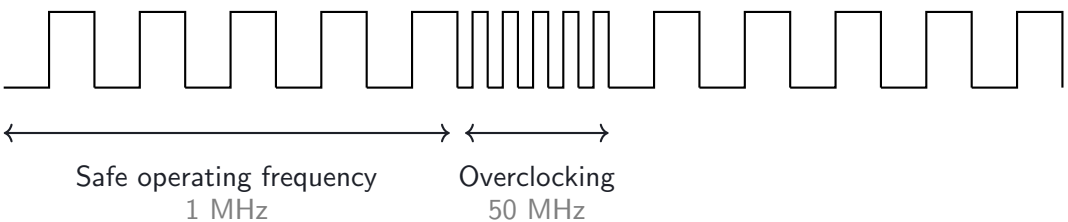
From the datasheet:
Maximum frequency: 20 MHz

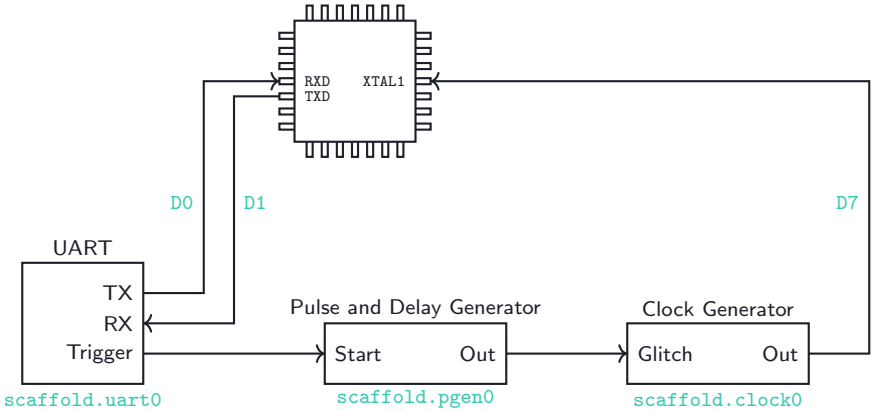



D
Q
Flip-Flop
CLK
D
Q
Flip-Flop
CLK
D
Q
Flip-Flop
CLK
D
Q
Flip-Flop
CLK
CLOCK
Propagation Delay < Clock Period


D
Q
Flip-Flop
CLK
D
Q
Flip-Flop
CLK
D
Q
Flip-Flop
CLK
D
Q
Flip-Flop
CLK
CLOCK
Propagation Delay > Clock Period

Goal: **Inject faults** during program execution.
How: **Overclock** the circuit during a short period of time.




RXD
TXD
XTAL1
D0
D1
D7
UART
TX
RX
Trigger
Pulse and Delay Generator
Start
Out
Clock Generator
Glitch
Out
scaffold.uart0
scaffold.pgen0
scaffold.clock0

During attack setup, add:

```
# Connect scaffold peripherals
pgen = scaffold.pgen0
uart.trigger >> pgen.start
pgen.out >> clock.glitch
```

During attack setup, add:

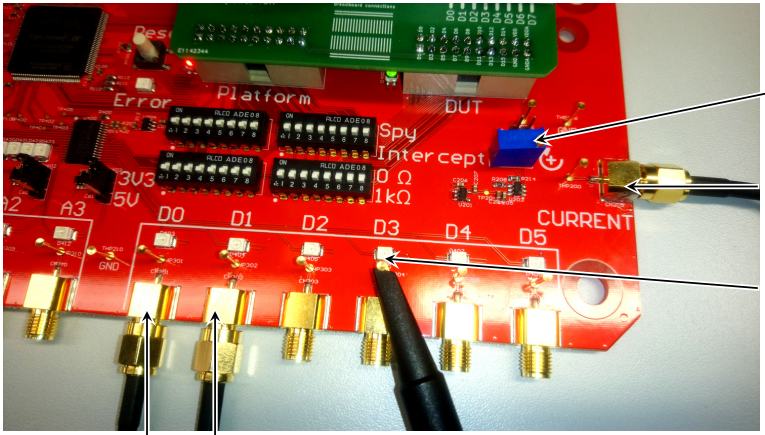
```
# Connect scaffold peripherals
pgen = scaffold.pgen0
uart.trigger >> pgen.start
pgen.out >> clock.glitch

# Configure the clock peripheral for glitching
clock.glitch_frequency = 50e6  # 50 MHz overclocking
clock.glitch_count = 20  # 20 fast clock edges
```

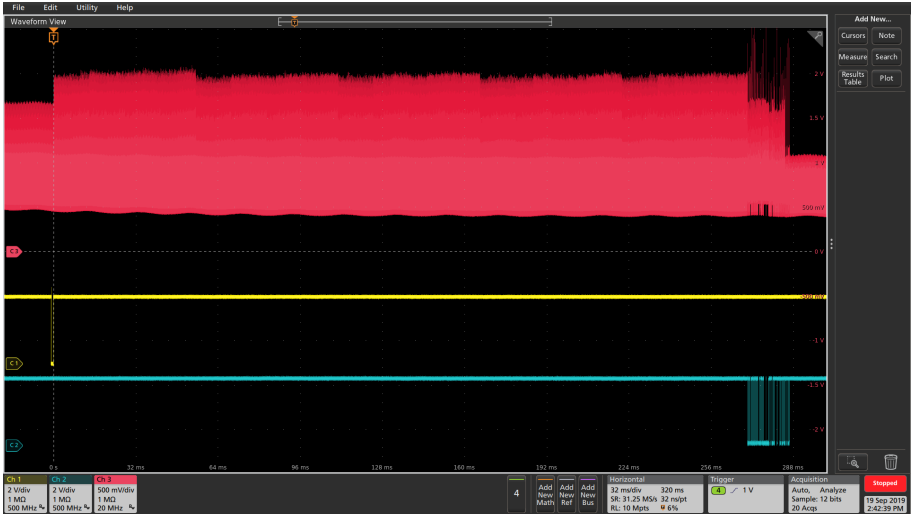
During attack setup, add:

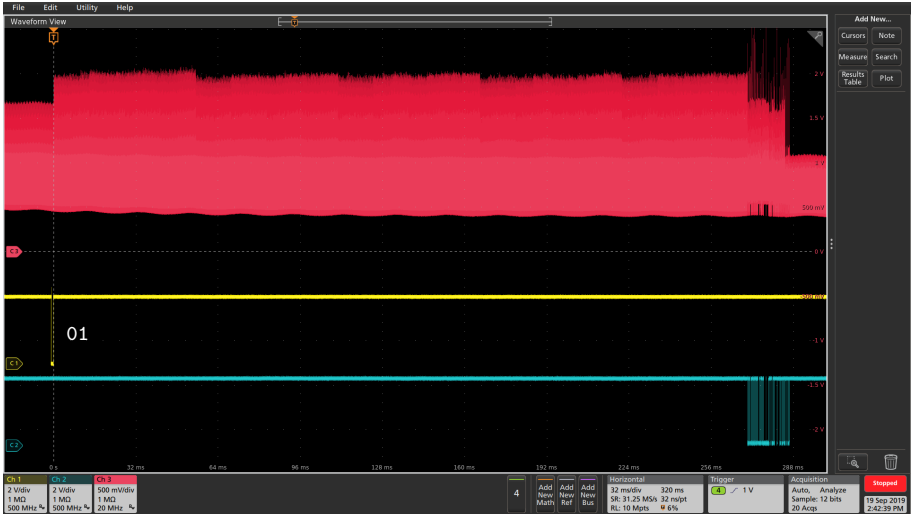
```
# Connect scaffold peripherals
pgen = scaffold.pgen0
uart.trigger >> pgen.start
pgen.out >> clock.glitch

# Configure the clock peripheral for glitching
clock.glitch_frequency = 50e6  # 50 MHz overclocking
clock.glitch_count = 20  # 20 fast clock edges

# Configure the delay generator
pgen.delay = ???  # TODO: look at the power trace!
pgen.width = 10e-9  # 10 ns
```

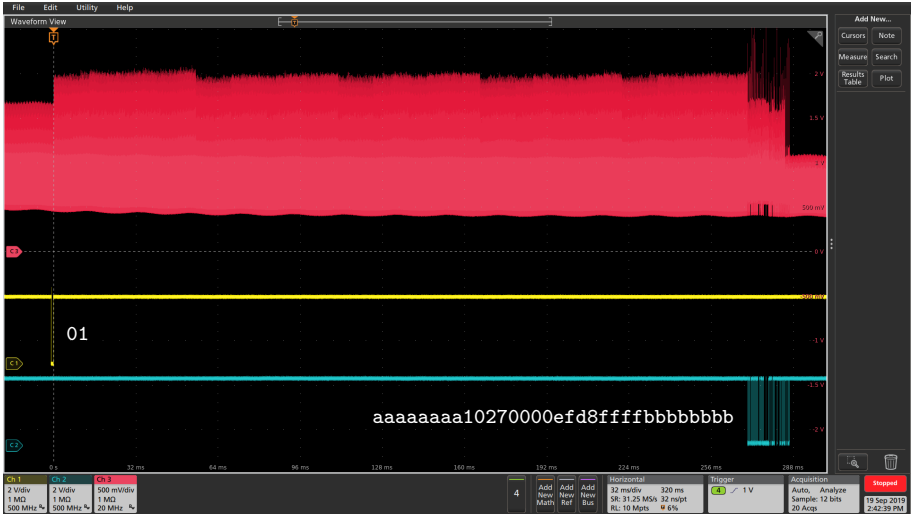
Modify command test:

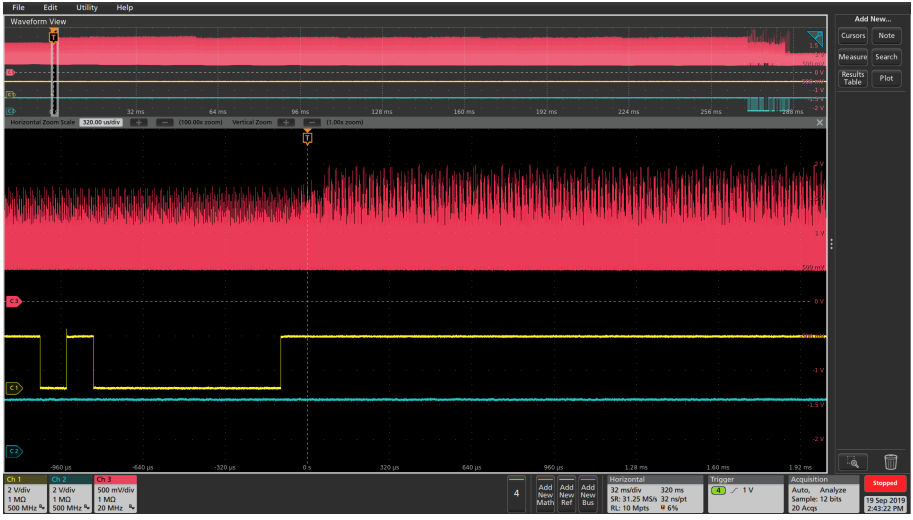
```
uart.flush()  # Discard received junk bytes
uart.transmit(b'\x01', trigger=True)  # Enable trigger after transmission
# Glitching may crash the device.
# Configure timeout to avoid waiting for a response that may never come...
scaffold.timeout = 0.5  # Half a second
res = uart.receive(16)  # Receive the response
```


Measurement Resistor
Power Trace output
Trigger
uart.trigger >> scaffold.d3
TX
RX
Platform
DUT
Spy
Intercept
CURRENT
D0
D1
D2
D3
D4
D5
3V3
5V
GND


File
Edit
Utility
Help
Waveform View
Add New...
Cursors
Note
Measure
Search
Results Table
Plot
2 V
1.5 V
1 V
500 mV
C1
0 V
-500 mV
-1 V
C1
-1.5 V
-2 V
C2
0 s
32 ms
64 ms
96 ms
128 ms
160 ms
192 ms
224 ms
256 ms
288 ms
Ch 1
2 V/div
1 MΩ
500 MHz
Ch 2
2 V/div
1 MΩ
500 MHz
Ch 3
500 mV/div
1 MΩ
20 MHz
4
Add New Math
Add New Ref
Add New Bus
Horizontal
32 ms/div 320 ms
SR: 31.25 MS/s 32 ns/pt
RL: 10 Mpts 6%
Trigger
4 1 V
Acquisition
Auto, Analyze
Sample: 12 bits
20 Acqs
Stopped
19 Sep 2019
2:42:39 PM


File
Edit
Utility
Help
Waveform View
Add New...
Cursors
Note
Measure
Search
Results Table
Plot
01
C1
C2
0 s
32 ms
64 ms
96 ms
128 ms
160 ms
192 ms
224 ms
256 ms
288 ms
Ch 1
2 V/div
1 MΩ
500 MHz
Ch 2
2 V/div
1 MΩ
500 MHz
Ch 3
500 mV/div
1 MΩ
20 MHz
Add New Math
Add New Ref
Add New Bus
Horizontal
32 ms/div
320 ms
SR: 31.25 MS/s
32 ns/pt
RL: 10 Mpts
6%
Trigger
1 V
Acquisition
Auto, Analyze
Sample: 12 bits
20 Acqs
Stopped
19 Sep 2019
2:42:39 PM


File Edit Utility Help
Waveform View
Add New...
Cursors Note
Measure Search
Results Table Plot
01
aaaaaaaa10270000efd8ffffbbbbbbbb
Ch 1 2 V/div 1 MΩ 500 MHz
Ch 2 2 V/div 1 MΩ 500 MHz
Ch 3 500 mV/div 1 MΩ 20 MHz
Horizontal 32 ms/div 320 ms SR: 31.25 MS/s 32 ns/pt RL: 10 Mpts 6%
Trigger 1 V
Acquisition Auto, Analyze Sample: 12 bits 20 Acqs
Stopped
19 Sep 2019 2:42:39 PM


File Edit Utility Help
Waveform View
Add New...
Cursors Note
Measure Search
Results Table Plot
32 ms 64 ms 96 ms 128 ms 160 ms 192 ms 224 ms 256 ms 288 ms
Horizontal Zoom Scale 320.00 us/div (100.00x zoom) Vertical Zoom (1.00x zoom)
-960 µs -640 µs -320 µs 0 s 320 µs 640 µs 960 µs 1.28 ms 1.60 ms 1.92 ms
Ch 1 2 V/div 1 MΩ 500 MHz
Ch 2 2 V/div 1 MΩ 500 MHz
Ch 3 500 mV/div 1 MΩ 20 MHz
Add New Math
Add New Ref
Add New Bus
Horizontal 32 ms/div 320 ms SR: 31.25 MS/s 32 ns/pt RL: 10 Mpts 6%
Trigger 1 V
Acquisition Auto, Analyze Sample: 12 bits 20 Acqs
Stopped
19 Sep 2019 2:43:22 PM

- Faults can be rare events:
  run a loop until the response changes.
- Don't forget to restart the device before each test.
- ATMEGA may crash, Scaffold API will raise a TimeoutError:
  handle this exception in the script.
- Parse the response correctly and print the counter value.

If this is too difficult: try executing the solution script directly:

```
python3 solutions/spoiler-alert/solution1-loop.py
```

```
while True:  # Try glitching in loop
    scaffold.power.restart_dut(ton=0.1)  # Restart device before each test
    uart.flush()  # Discard received junk bytes
    uart.transmit(b'\x01', trigger=True)  # Enable trigger after transmission
    # Glitching may crash the device.
    # Configure timeout to avoid waiting for a response that may never come...
    scaffold.timeout = 0.5  # Half a second
    # Handle timeout errors correctly
    try:
        res = uart.receive(16)  # Receive the response
        print(hexlify(res).decode())  # Print the response
    except TimeoutError as e:
        print('Timeout')
```

CHALLENGE 2: GLITCHING A PIN VERIFICATION

Let's load the second challenge program:

```
cd /home/user/workshop/
python3 scripts/atmega.py -d COM1 --load challenges/ch2-pin.bin
```

The PIN program waits for command byte `0x02` and 8 digits PIN code from the UART.

- If the PIN is correct: a secret 16 bytes string is returned.
- If the PIN is incorrect: an error 16 bytes string is returned.

Your next mission: **Get the secret!**[1]

[1] Without running `strings ch2-pin.bin`...

When sending over the UART:

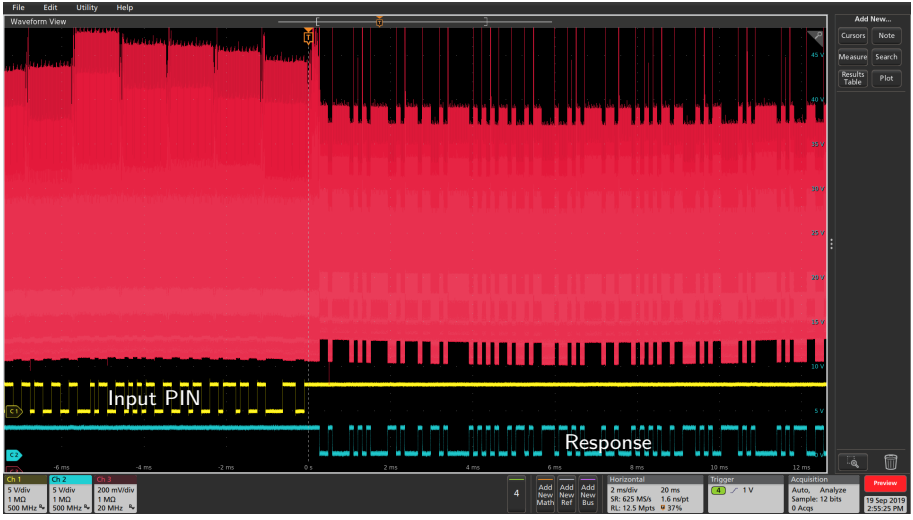
02 (hexadecimal, 1 byte) + `"12345678"` (string, 8 bytes)

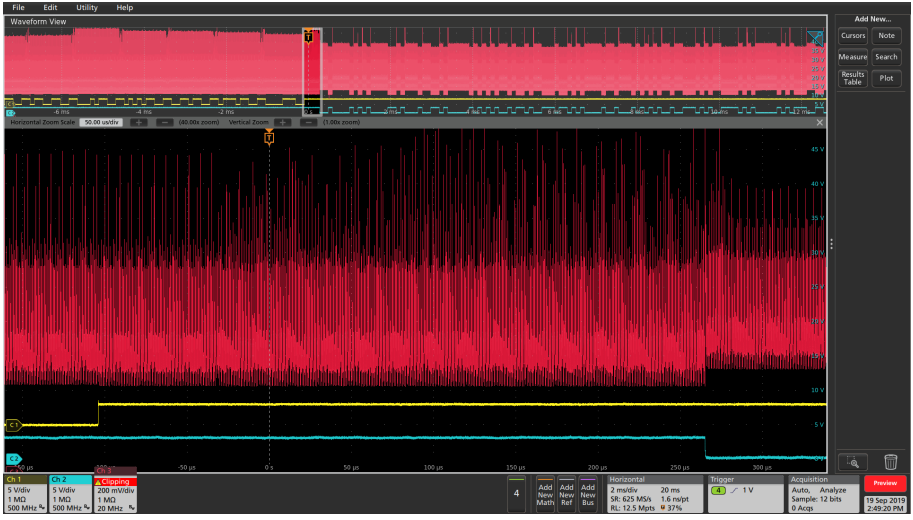
The device should respond with:

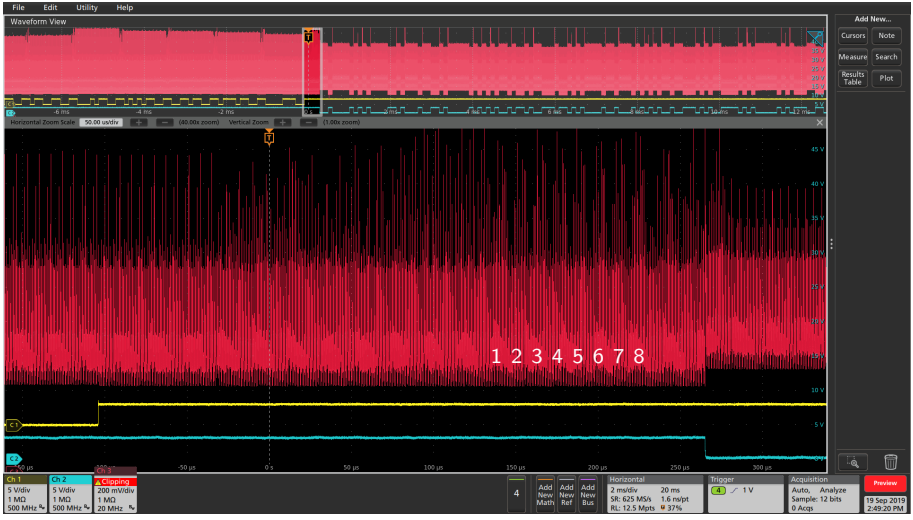
`"BadPin!TryAgain!"` (string, 16 bytes)

Adapt your attack script from challenge 1:

- Change the command sent to the ATMEGA328
- Try different glitch injection time
- Look at the power trace for timing: don't shoot too early, or tool late...


File Edit Utility Help
Waveform View
Add New...
Cursors
Note
Measure
Search
Results Table
Plot
Input PIN
Response
Ch 1 5 V/div 1 MΩ 500 MHz
Ch 2 5 V/div 1 MΩ 500 MHz
Ch 3 200 mV/div 1 MΩ 20 MHz
Add New Math
Add New Ref
Add New Bus
Horizontal 2 ms/div 20 ms SR: 625 MS/s 1.6 ns/pt RL: 12.5 Mpts 37%
Trigger 1 V
Acquisition Auto, Analyze Sample: 12 bits 0 Acqs
Preview
19 Sep 2019 2:55:25 PM


File Edit Utility Help
Waveform View
Add New...
Cursors
Note
Measure
Search
Results Table
Plot
Horizontal Zoom Scale 50.00 us/div
Vertical Zoom
45 V
40 V
35 V
30 V
25 V
20 V
15 V
10 V
5 V
C1
C2
-50 µs
0 s
50 µs
100 µs
150 µs
200 µs
250 µs
300 µs
Ch 1
5 V/div
1 MΩ
500 MHz
Ch 2
5 V/div
1 MΩ
500 MHz
Clipping
200 mV/div
1 MΩ
20 MHz
Add New Math
Add New Ref
Add New Bus
Horizontal
2 ms/div 20 ms
SR: 625 MS/s 1.6 ns/pt
RL: 12.5 Mpts 37%
Trigger
1 V
Acquisition
Auto, Analyze
Sample: 12 bits
0 Acqs
Preview
19 Sep 2019
2:49:20 PM


File Edit Utility Help
Waveform View
Add New...
Cursors
Note
Measure
Search
Results Table
Plot
Horizontal Zoom Scale 50.00 us/div
1 2 3 4 5 6 7 8
Ch 1 5 V/div 1 MΩ 500 MHz
Ch 2 5 V/div 1 MΩ 500 MHz
Clipping 200 mV/div 1 MΩ 20 MHz
Horizontal 2 ms/div 20 ms SR: 625 MS/s 1.6 ns/pt RL: 12.5 Mpts 37%
Trigger 1 V
Acquisition Auto, Analyze Sample: 12 bits 0 Acqs
Preview
19 Sep 2019 2:49:20 PM

CHALLENGE 3: MORE THAN YOU EXPECT...

Let's load the third challenge program:

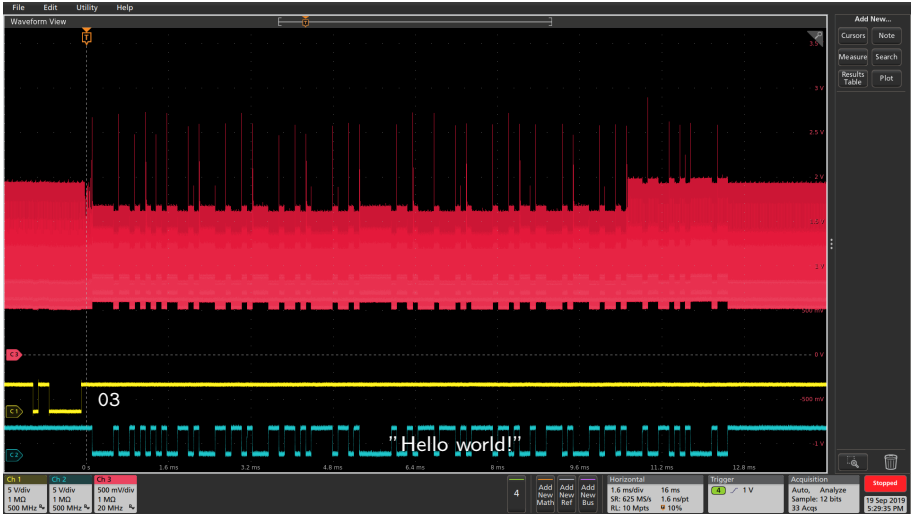
```
cd /home/user/workshop/
python3 scripts/atmega.py -d COM1 --load challenges/ch3-overflow.bin
```

Challenge 3 program accepts two commands:

- 03 (hexa, 1 byte): returns the string "Hello world!"
- 04 (hexa, 1 byte): return the checksum of an internal **secret** string.

Tips:

- \0
- Timing is important...
- No math involved here...


File Edit Utility Help
Waveform View
Add New...
Cursors Note
Measure Search
Results Table Plot
03
"Hello world!"
Ch 1 5 V/div 1 MΩ 500 MHz
Ch 2 5 V/div 1 MΩ 500 MHz
Ch 3 500 mV/div 1 MΩ 20 MHz
Add New Math
Add New Ref
Add New Bus
Horizontal 1.6 ms/div 16 ms SR: 625 MS/s 1.6 ns/pt RL: 10 Mpts 10%
Trigger 1 V
Acquisition Auto, Analyze Sample: 12 bits 33 Acqs
Stopped
19 Sep 2019 5:29:35 PM