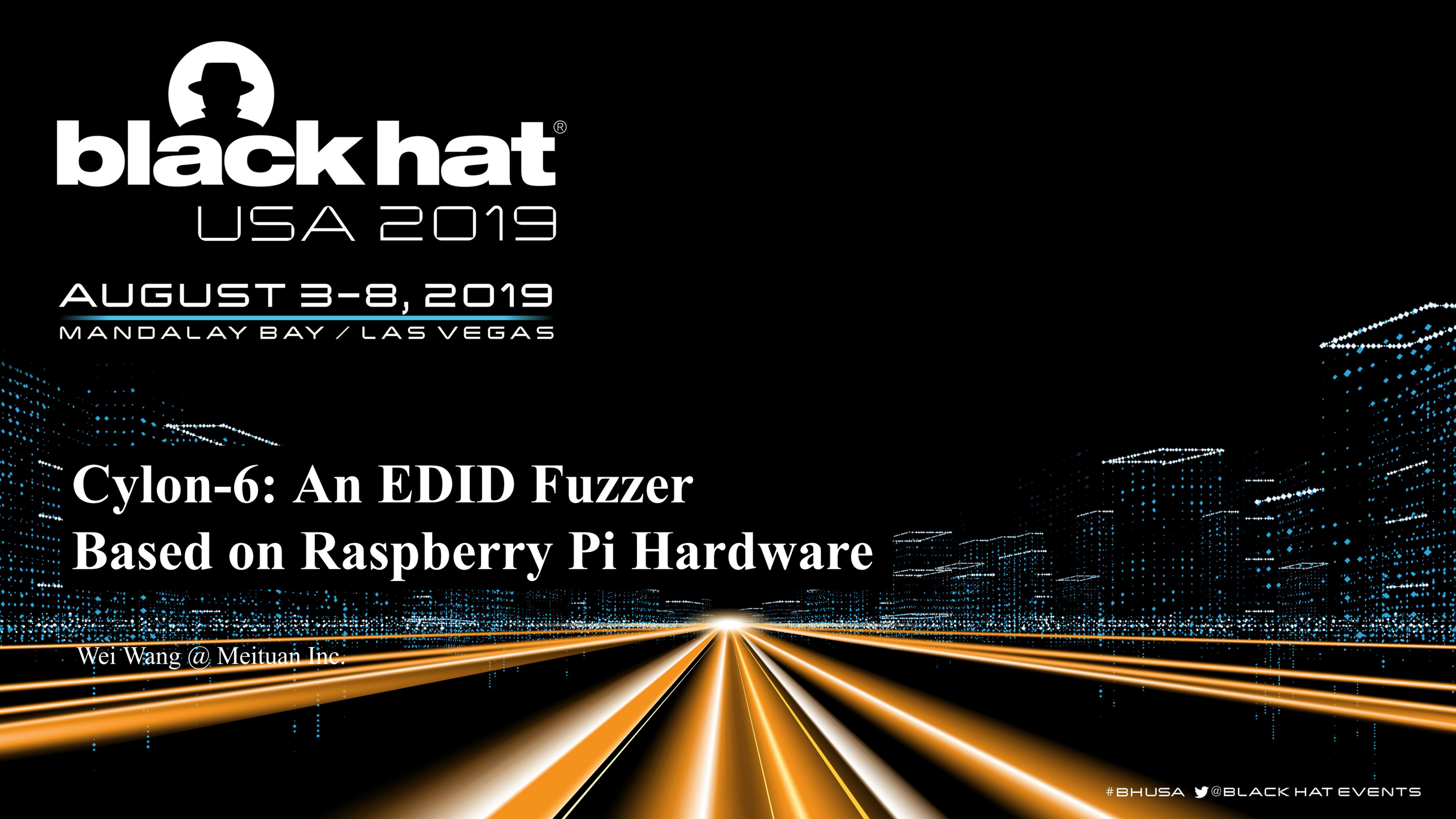


Cylon-6: An EDID Fuzzer Based on Raspberry Pi Hardware

Wei Wang @ Meituan Inc.

About me

王伟
Wei Wang

- Security Researcher @ Meituan Inc.
- BH USA 2018 presenter, interested in kernel, networking, IoT, etc.
- (nothing more …

- Email: io@stdio.io
- Github: @kings-way

Agenda

1. **Previous research by others**
2. **Introduction to EDID and DDC**
3. **Build with Raspberry Pi**
4. **Comparison with Arduino**
5. **Demo and Q&A**

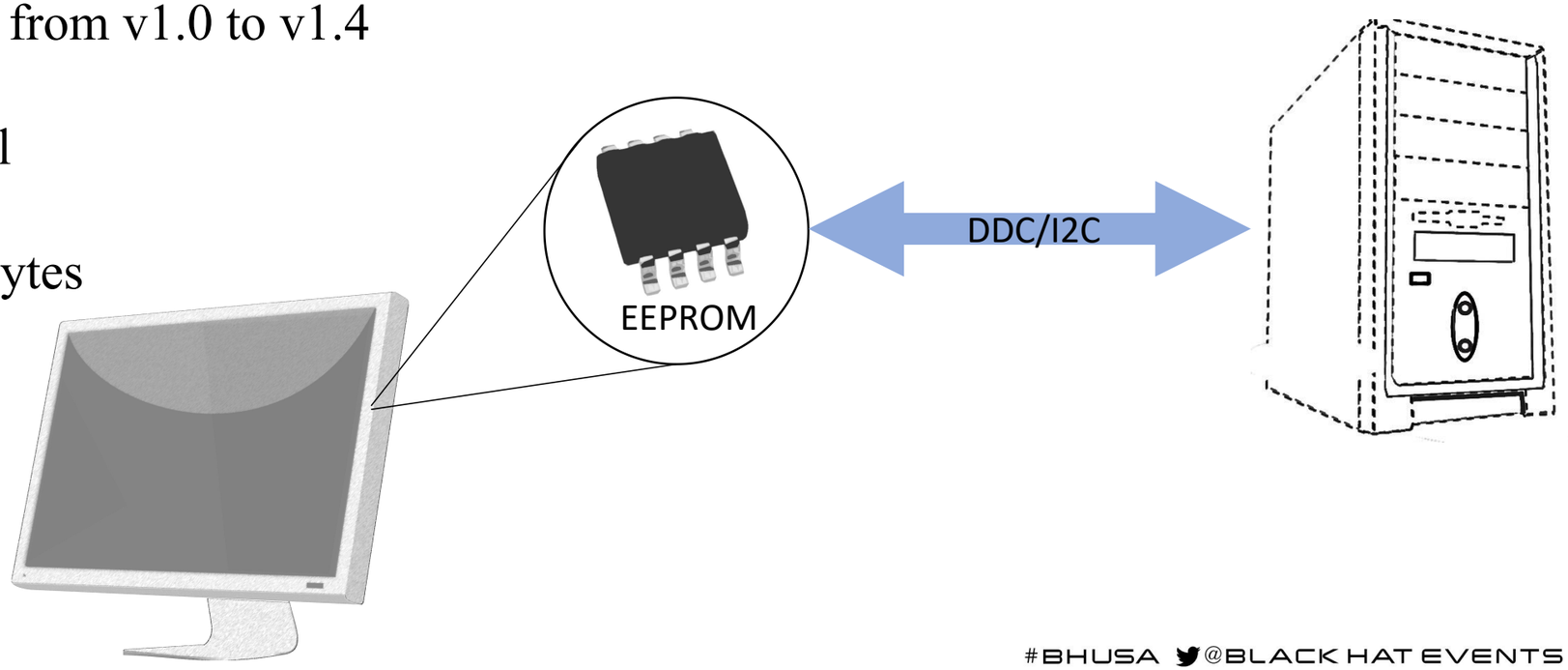
1. Previous research

- *BlackHat EU 2012, Andy Davis*
  HDMI – Hacking Displays Made Interesting
  throughly analysis about DDC/EDID/CEC, built a fuzzer based on Arduino

- *DefCon 2015, Joshua Smith*
  HIGH-DEF FUZZING EXPLOITATION OVER HDMI-CEC
  mainly focused on CEC fuzzing, based on USB-CEC adapter

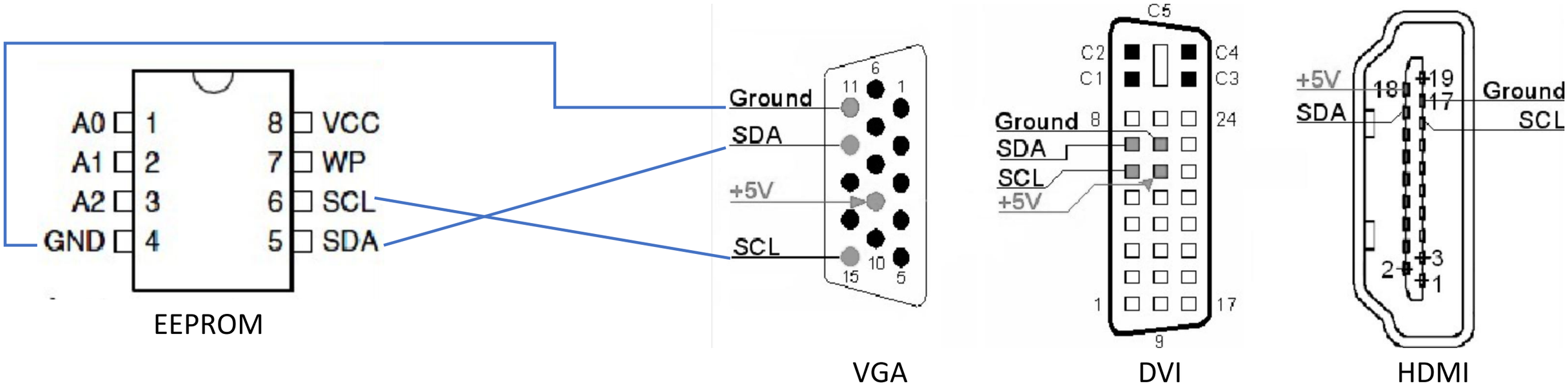
2. Introduce EDID

- EDID stands for “Extended Display Identification Data”
- Used to describe monitor capabilities, like sizes
- Formats defined by VESA, range from v1.0 to v1.4
- Transmitted through DDC channel
- Stored in EEPROM, 128 or 256 Bytes



2. Introduce DDC

- DDC is built on I2C, directly connected to EEPROM
- VGA/DVI/HDMI/DP all have DDC channel

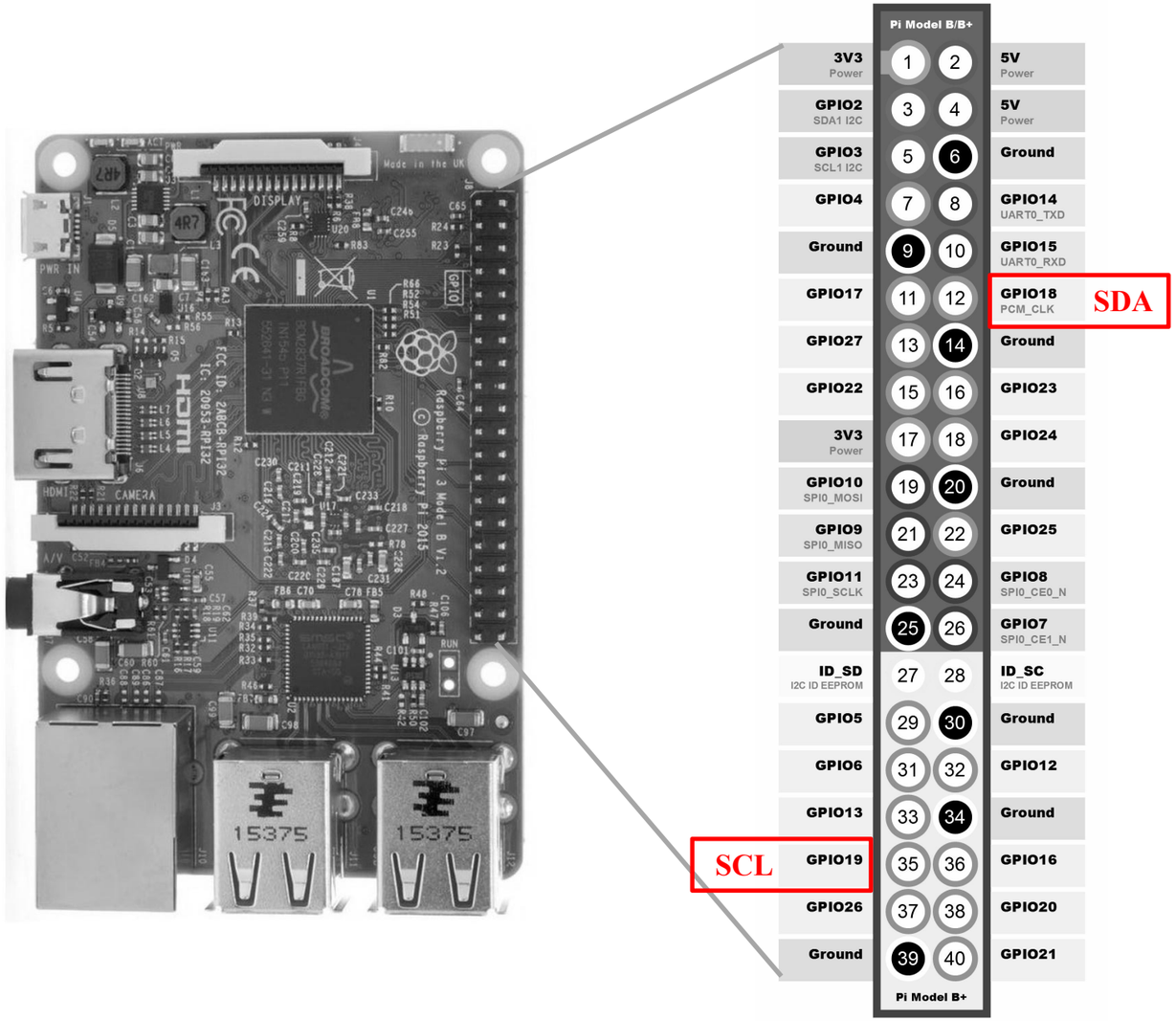


3. Build with RPi

What does RPi have

- **BSC/I2C Master**
  GPIO 2 <==> SDA
  GPIO 3 <==> SCL

- **BSC/I2C Slave**
  GPIO 18 <==> SDA
  GPIO 19 <==> SCL



3. Build with RPi

Some libs to consider

- **joan2937/pigpio**
  C library
  supports BSC slave
  offers /dev and socket interface

- **rsta2/circle**
  bare metal C++ library
  supports BSC slave
  many examples

- **hendric-git/bsc-slave**
  Linux kernel module
  controller to BSC slave mode

3. Build with RPi

Some libs to consider

- **joan2937/pigpio**
  C library
  supports BSC slave
  offers /dev and socket interface

  User mode program,
  too slow to respond to I2C operations ✗

- **rsta2/circle**
  bare metal C++ library
  supports BSC slave
  many examples

  Super Fast, simply to use ✓

- **hendric-git/bsc-slave**
  Linux kernel module
  controller to BSC slave mode

  Not widely compatible ✗

3. Build with RPi

Problems & Solutions

1) ACK/NACK

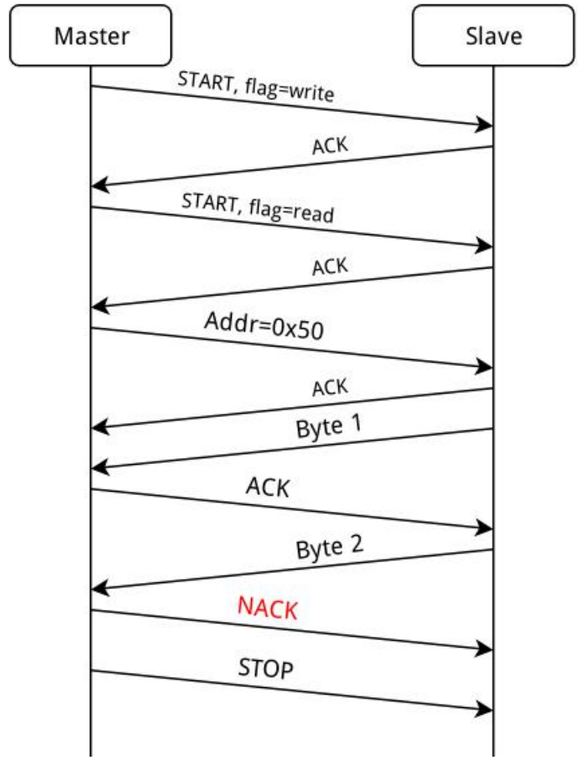
When Master reads from slave, it will send:

ACK ==> Another byte is desired

NACK ==> Read finished, terminate transmission

Problems:

- Handled by hardware
- ACK/NACK status is not available to us with RPi BCM2835
- So, slave doesn't know how many bytes the master wants to read.



Typical master read timeline

3. Build with RPi

Problems & Solutions

1) ACK/NACK

When Master reads from slave, it will send:

ACK ==> Another byte is desired

NACK ==> Read finished, terminate transmission

Approach 1:

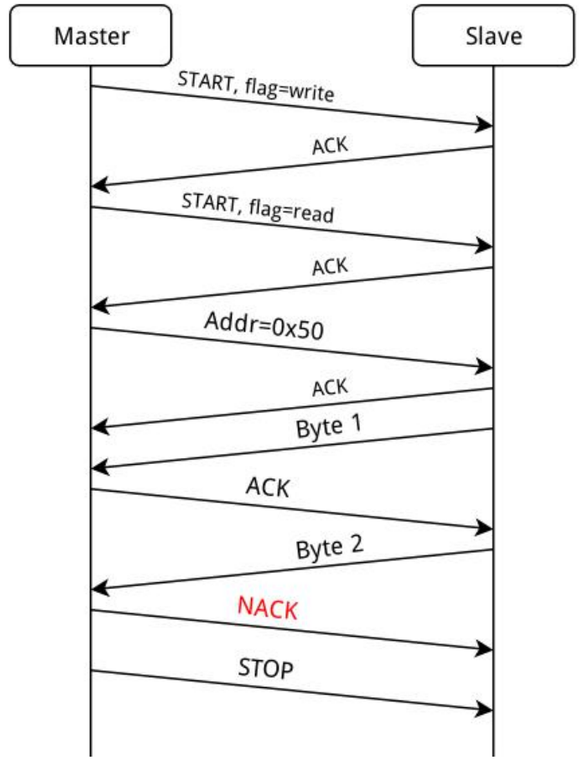
Slave keeps writing to TX FIFO till all EDID data been transmitted

Result: Failed

Bytes lost when TX FIFO is full;

Master will get garbage data on another transmission;

==> Can not transfer complete EDID data.



Typical master read timeline

3. Build with RPi

Problems & Solutions

1) ACK/NACK

When Master reads from slave, it will send:

ACK ==> Another byte is desired

NACK ==> Read finished, terminate transmission

Approach 2:

Always to fully fill the TX FIFO (16 bytes) if there is any space left;

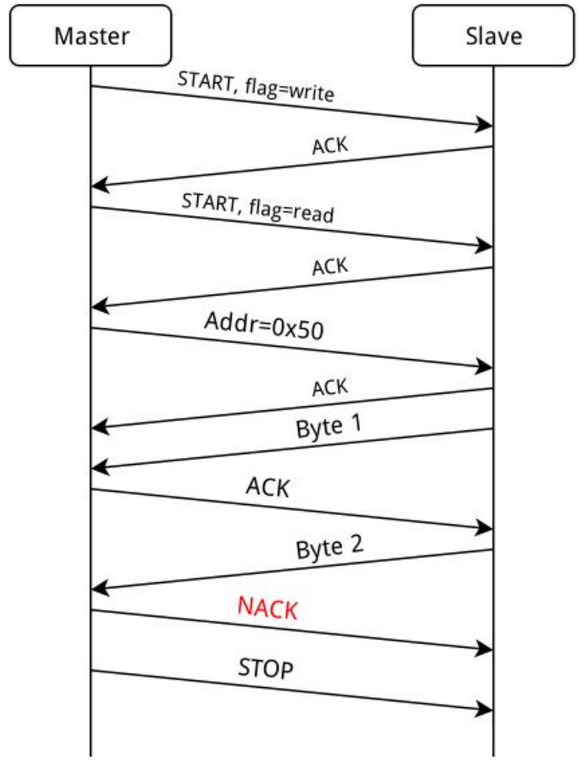
Clear the TX FIFO on every START.

Result: Failed

The 'BRK' bit within 'CR' Register (control register) does not work

*BCM2835 chipset is to blame

*BRK=1: Stop operation and clear the FIFOs



Typical master read timeline

3. Build with RPi

Problems & Solutions

1) ACK/NACK

When Master reads from slave, it will send:

ACK ==> Another byte is desired

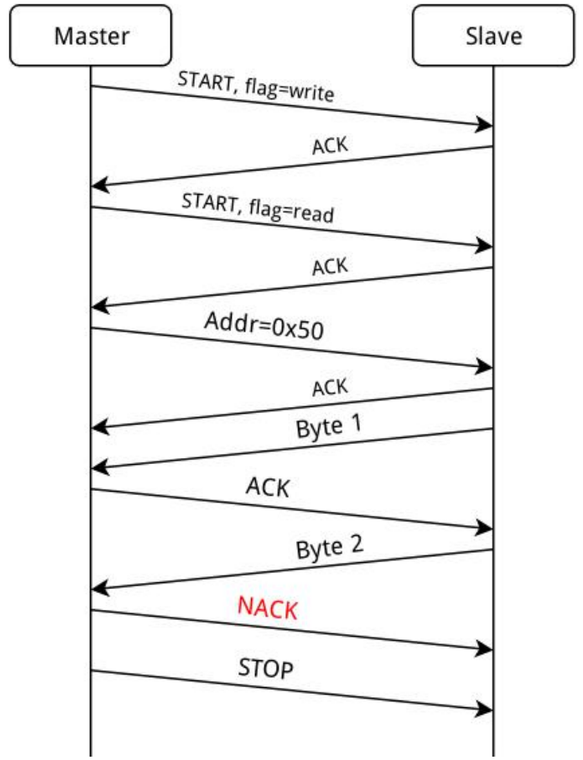
NACK ==> Read finished, terminate transmission

Approach 3:

Always to fully fill the TX FIFO (16 bytes) if there is any space left;

Set a counter for bytes written, stop when all EDID bytes transmitted.

Result: Succeed



Typical master read timeline

3. Build with RPi

Problems & Solutions

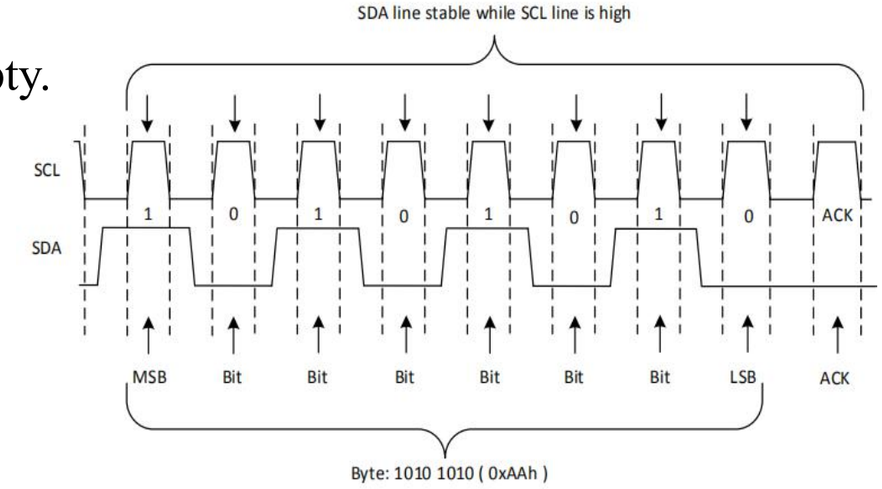
2) Too slow to act

I2C bus protocol requires strict timing;
User mode code and RPi kernel do not offer real time capability;
Master keeps getting the same last byte if slave's TX FIFO is empty.

Approach:

Use bare metal library like circle.

Result: Succeed



Example of Single Byte Data Transfer
(Copyright © 2015, Texas Instruments Incorporated)

3. Build with RPi

Problems & Solutions

3) Kernel and drivers probe EDID differently

For Linux AMD GPU with HDMI:

* read 8 bytes at 0x0 ==> probe
* read 1 byte at 0x0 ==> probe
* read 128 bytes at 0x0
* read 128 bytes at 0x80

For Linux Intel GPU with HDMI:

* read 1 byte at 0x0 ==> probe
* read 128 bytes at 0x0
* read 128 bytes at 0x80

Approach:

dirty hack with the code, transmit different length of data for different GPU.

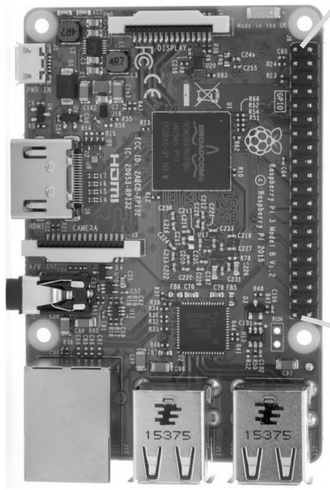
Result: Just works

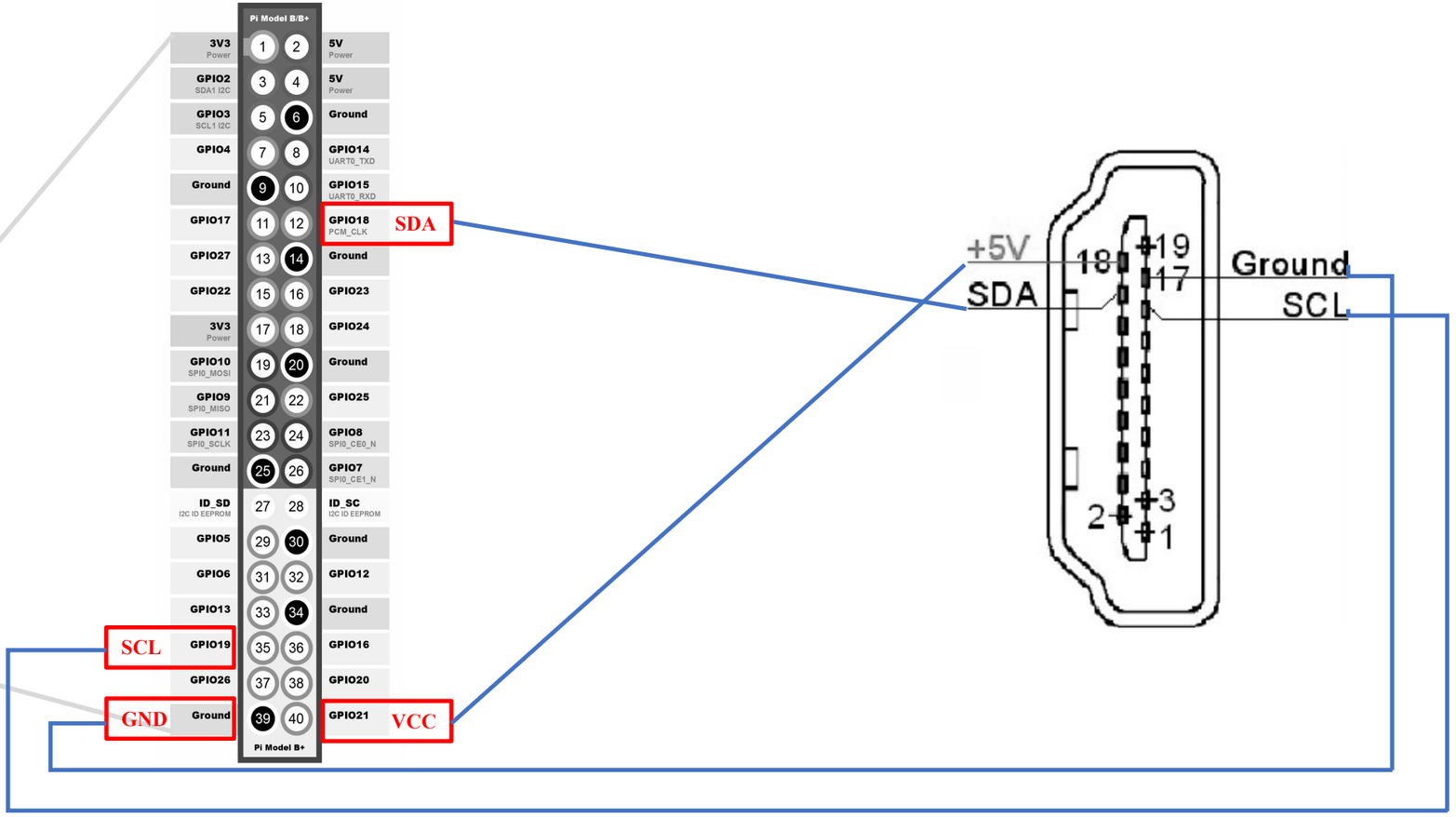
3. Build with RPi

Connecting the wire

- GPIO 18 <==> SDA
- GPIO 19 <==> SCL

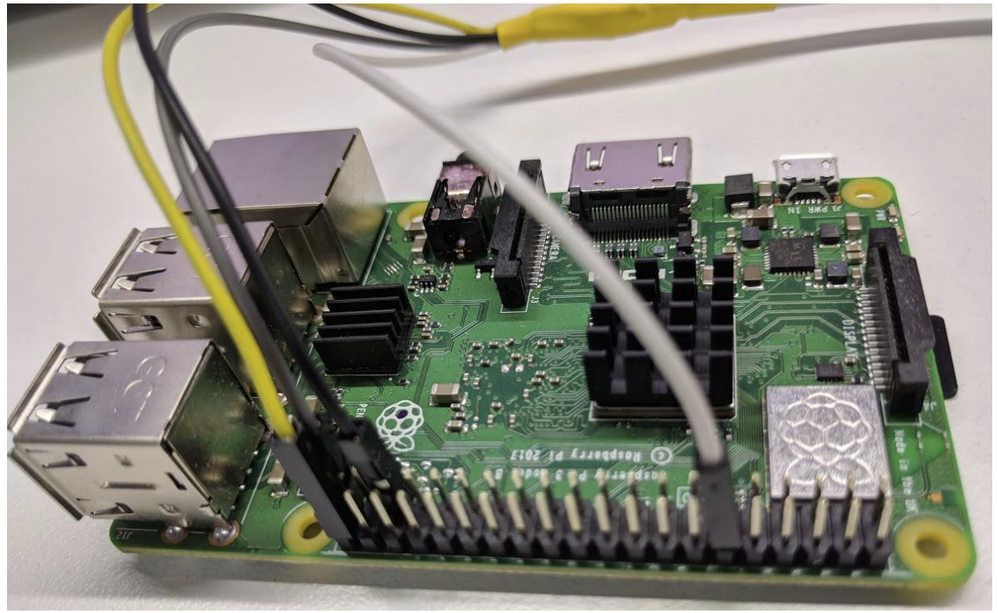
- GPIO 21 <==> VCC/+3V
- Pin 39 <==> Ground

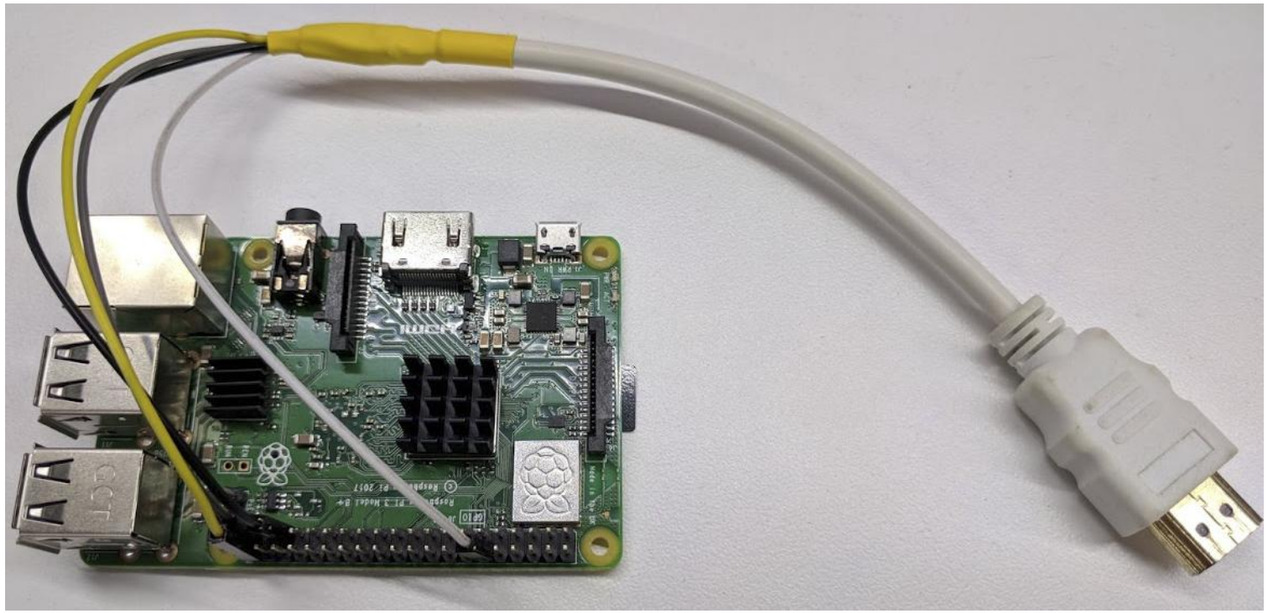




3. Build with RPi

Connecting the wire





3. Build with RPi

Additional work

- **Auto Plug in/out**
  since we use GPIO 21 as the VCC power, we are able to make the HDMI “virtually” connected or disconnected by setting the voltage high or low.

- **Fundamental EDID data format**
  Preamble bits: 0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0x00
  Checksum: data at 0x7f and 0xff holds checksum.
  Sum of 128 bytes equals 0 (mod 256).

- **EDID data mutation**
  a good mutation engine is essential to a fuzzer, but it has not been implemented yet;
  (may never be implemented :)

4. Comparison

with Arduino

	Raspberry Pi	some Arduino boards
Full I2C Slave support	No	Yes
Full SPI Slave support	No (hardly working)	Yes
User controllable ACK/SACK	No	Yes
Bare metal libraries	few	many
Peripheral Interface	many	few
Simply to Use	Yes	No
Cost of device	High	Low

Both RPi and Arduino have some advantages and disadvantages. But when it comes to EEPROM simulation, Arduino has better hardware support, making it rather easy to finish the job. Truth always hurts :(

5. Demo and Q&A

It should be blank here :)