









**EE Live!** Featuring **ESC**  
The Embedded Systems Conference

**Data-driven Interfaces  
for Embedded Systems**

**Matt Liberty**  
Jetperch LLC

#eelive Produced by EE Times

 ESC  
EMBEDDED SYSTEMS  
CONFERENCE
  BLACK HAT  
EMBEDDED
  INTERNET OF  
THINGS
  HARDWARE  
STARTUP
  ANDROID  
ENGINEERING
  FPGA  
ENGINEERING
  SUPER C++  
TUTORIAL
  UBM  
Tech

**EE Live!** Featuring **ESC**  
The Embedded Systems Conference

**Agenda**

- Introduction
- Interface description languages (from the networking world)
- Register specification languages (from the hardware world)
- Report from the field
- Conclusion and questions

Download from <http://jetperch.com/?p=107>

 ESC  
EMBEDDED SYSTEMS  
CONFERENCE
  BLACK HAT  
EMBEDDED
  INTERNET OF  
THINGS
  HARDWARE  
STARTUP
  ANDROID  
ENGINEERING
  FPGA  
ENGINEERING
  SUPER C++  
TUTORIAL
  UBM  
Tech

**EE Live!** Featuring **ESC**  
The Embedded Systems Conference

## Typical process for integrating a component

```

graph TD
    A[Search the manufacturer's website for a driver or example] --> B[Search the web for an example]
    B --> C[Create your own .h file and driver based upon the datasheet]
  
```

INDUSTRY FAIL

esc  
EMBEDDED SYSTEMS CONFERENCE

BLACK LIP TECHNOLOGIES

INTERNET OF THINGS

INDUSTRIAL SYSTEMS

ARMED INTELLIGENCE

FAAS OPERATIONS

EMBEDDED C++

UBM Tech

**EE Live!** Featuring **ESC**  
The Embedded Systems Conference

## Why am I speaking on this topic now?

- Share lessons from my recent experience
  - Researched latest and greatest technologies
  - Developed a custom definition language to meet the project needs
- Promote improvement: we as an industry can do better!
  - Inconsistent vendor-provided drivers
  - Continued need to create device drivers from datasheets:  
Exchange data, not datasheets!
  - Simplify messaging between devices, applications and servers

esc  
EMBEDDED SYSTEMS CONFERENCE

BLACK LIP TECHNOLOGIES

INTERNET OF THINGS

INDUSTRIAL SYSTEMS

ARMED INTELLIGENCE

FAAS OPERATIONS

EMBEDDED C++


UBM Tech

**EE Live!** Featuring **esc**  
The Embedded Systems Conference

## Common embedded software actions

- Communication
  - Get and set registers, often groups of registers
    - Initialization
    - "Steady state"
  - Send and receive messages / structures / packets
  - Move or forward data
- Data & signal processing
  - Business logic
  - Compute and control next actions
  - Handle events: interrupts and messages
- User interface

**Focus of this talk**



esc  
EMBEDDED SYSTEMS CONFERENCE

BLACK ICE  
TECHNOLOGY

INTEGRITY OF  
DESIGN

TECHNOLOGY  
SOLUTIONS

ARMED  
RESEARCH

FAIR  
SOFTWARE

CODE  
TECHNOLOGY

UBM  
Tech

**EE Live!** Featuring **esc**  
The Embedded Systems Conference

## Agenda

- Introduction
- Interface description languages
  - Google protocol buffers
  - Thrift
  - Avro
- Register specification languages
- Report from the field
- Conclusion and questions

esc  
EMBEDDED SYSTEMS CONFERENCE

BLACK ICE  
TECHNOLOGY

INTEGRITY OF  
DESIGN

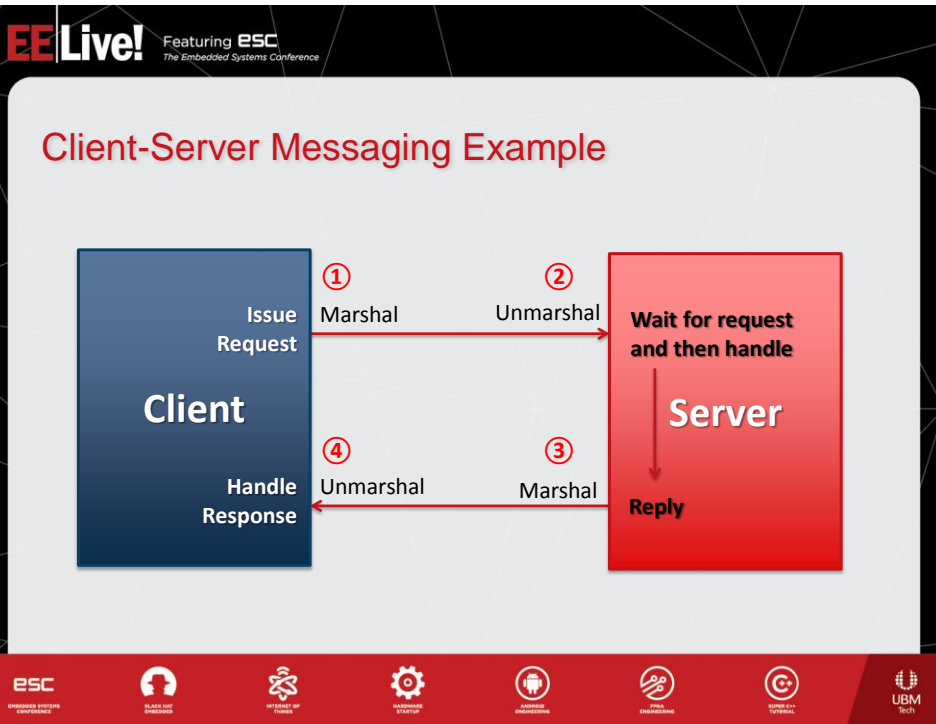
TECHNOLOGY  
SOLUTIONS

ARMED  
RESEARCH

FAIR  
SOFTWARE

CODE  
TECHNOLOGY

UBM  
Tech



**EE Live!** Featuring **ESC**  
The Embedded Systems Conference

## Interface Description Language (IDL)

- Specification language that describes the interface for a software component
  - Cross-platform data marshalling, sometimes called serialization
  - Reduce requirement to write tedious error-prone code
  - Usually compiled to an assortment of programming languages and operating systems
  - Accounts for machine word lengths and endianness
- Usually includes data validation on unmarshal
- Gracefully evolve messages over time (versioning)
- Often include remote procedure call (RPC) semantics or provide support for RPC
- Balance efficiency of data size, computation, development time

Logos at the bottom: ESC, BLACK ICE PROGRAMS, INTERNET OF THINGS, AUTONOMOUS SYSTEMS, ARMED INTELLIGENCE, FMS OPERATIONS, C++, UBM Tech.

## Brief History of IDLs and Network Interfaces

- 1980s: RPC and the original Interface Description Language
- 1990s: [CORBA](#)
- 1990s: Java RMI
- 2000: SOAP + XML
- 2000s: REST (Representational state transfer)
- 2002: [Google protocol buffers](#) (used by most Google services)
- 2007: [Thrift](#) (used by Facebook)
- 2009: [Avro](#) (used by Apache Hadoop)




## IDL Tradeoffs

- Descriptiveness
  - Complexity
  - Inclusion of “logic” and remote procedure calls
- Speed (CPU and transfer)
  - Binary, packed binary, human readable text
  - Error checking and data validation
  - Data packing & compression
- Robustness
  - Versioning & support for unknown future fields
  - Backwards compatibility
- Language synergy - how naturally does using an IDL fit into a chosen programming language



EE Live! Featuring **ESC**  
The Embedded Systems Conference



## Google Protocol Buffers

- Designed by Google starting in 2001
- Open-sourced in 2008 under permissive BSD license
- Active and well-used inside most Google services
- Supports C++, Java and Python officially
  - Focus on reliability and robustness
  - Unofficial [third-party support](#) for many more, but quality varies
  - Unofficial support for embedded C with static memory only
- protoc compiler generates “intermediate” code for target language

See <https://developers.google.com/protocol-buffers/docs/overview>

ESC  
EMBEDDED SYSTEMS CONFERENCE

BLACK LIP TECHNOLOGY

INTEGRITY OF FORM

ENGINEERING EXCELLENCE


ARMED INTELLIGENCE

FAIR OPERATIONS

UNIVERSITY OF TORONTO

UBM Tech

EE Live! Featuring **ESC**  
The Embedded Systems Conference



## Google Protocol Buffers

```

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
  repeated PhoneNumber phone = 4;
}

```

See <https://developers.google.com/protocol-buffers/docs/overview>

ESC  
EMBEDDED SYSTEMS CONFERENCE

BLACK LIP TECHNOLOGY

INTEGRITY OF FORM

ENGINEERING EXCELLENCE

ARMED INTELLIGENCE

FAIR OPERATIONS

UNIVERSITY OF TORONTO

UBM Tech

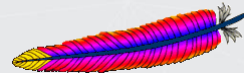
## Google Protocol Buffers: Encoding

- Key-Value pairs with type
  - varint: with 7 bits per 8-bit byte, LSB first leaving out leading zeros
  - 32-bit: fixed32, sfixed32, float
  - 64-bit: fixed64, sfixed64, double
  - Length delimited: varint length plus value for string, bytes, messages, packed repeated fields
- varint signed values use zig-zag encoding (not two's compliment) to limit the number of bytes required
- Unions are challenging to represent

See <https://developers.google.com/protocol-buffers/docs/overview>



## Apache Thrift



- Founded in 2007 to develop software services for the web
  - Focused on fully supporting cross-language services (RPC)
- Designed as successor to protocol buffers
  - Similar syntax to protocol buffers but with more features
- Not strongly coupled to a single protocol or serialization
- Supports MANY languages
- Basic types are:
  - bool, byte, i16, i32, i64, double
  - string: Encoding agnostic text or binary string
- thrift compiler generates “intermediate” code for target language

See <http://thrift.apache.org/docs/concepts/>  
<http://diwakergupta.github.io/thrift-missing-guide/>





## Apache Avro

- Started in 2009 and open-sourced under the Apache License
- Serialization with RPC capabilities
- Relies upon a schema
  - Eliminates per-field encoding overhead in serialized messages
  - Custom code generation not required
  - Uses JSON and somewhat ugly
- Avoids field IDs and instead uses the full schema to resolve differences
- Less stable than either Protocol Buffers or Thrift

See <http://avro.apache.org/docs/current/>



## Comparison of IDLs

Parameter	Protocol Buffers	Thrift	Avro
Serialization	✓	Multiple options	✓
Binary format	✓	✓	✓
RPC	_*	✓	✓
Schema	-	-	✓
Enumerations	✓	✓	✓
Constants	-	✓	-
Containers	-	List, set, map	Array, Map
License	BSD	Apache	Apache
Language support	3 (officially)	15+	4
Suitability for microcontrollers	Good**	Poor	Poor





## Other interesting networking technologies

- [ØMQ](#) (ZeroMQ)
  - Messaging library for distributed applications.
- [MQTT](#) (MQ Telemetry Transport)
  - Machine to machine connectivity protocol for devices to communicate to servers using publish/subscribe
  - Originally from IBM
- [XMPP](#) (Extensible Messaging and Presence Protocol)
  - Open-source instant messaging protocol, originally Jabber
- [AMQP](#) (Advanced Message Queuing Protocol)
  - Large-scale business messaging for the “cloud”
- [nanomsg](#)
  - Socket library for common communication patterns written in C
- [Wireshark](#) – protocol analyzer

## Agenda

- Introduction
- Interface description languages
- Register specification languages
  - IP-XACT (IEEE 1685-2009) and SystemRDL 1.0
  - ARM CMSIS-SVD for Cortex-M
- Report from the field
- Conclusion and questions

## Register Description Language (RDL)

- Formally define the registers and their bitfields
  - Verify correct definitions (no address collisions)
  - Single definition: don't repeat yourself (DRY)
- Generate output to a variety of files
  - C header files
  - RTL (Verilog, VHDL)
  - Documentation (HTML, DOC, PDF)



## IP-XACT (IEEE 1685-2009)



- XML format for defining hardware IP
- Component definition includes register definition (Section 6.10)
- Spec is available for free:  
<https://standards.ieee.org/findstds/standard/1685-2009.html>
- Originated with the SPIRIT Consortium which merged with Accellera in 2009



## IP-XACT Register Example

```

<spirit:register>
  <spirit:name>control</spirit:name>
  <spirit:description>Control register</spirit:description>
  <spirit:addressOffset>0x8</spirit:addressOffset>
  <spirit:size>32</spirit:size>
  <spirit:access>read-write</spirit:access>
  <spirit:field>
    <spirit:name>enable</spirit:name>
    <spirit:description>Enables the receiver</spirit:description>
    <spirit:bitOffset>0</spirit:bitOffset>
    <spirit:bitWidth>1</spirit:bitWidth>
  </spirit:field>
  <spirit:field>
    <!-- ... -->
  </spirit:field>
</spirit:register>

```

Extract from Section 6.10.2.3  
 IEEE 1685-2009 (page 104)



## SystemRDL 1.0



- Since source for device register description which can then be used to generate code and documentation
- 1.0 released in 2009
- The project lapsed, but Intel picked it up in 2012
- Uses embedded Perl statements to support more complex functionality
- SystemRDL compilers
  - Blueprint compiler: Cisco → Denali Software → Cadence → dead
  - CSRCompiler: Semifore (CSRSpec extends 1.0)

See <http://www.accellera.org/activities/committees/systemrdl/>



## Basic SystemRDL Register Example


```
addrmap my_component {
    reg {
        name = "HelloWorld";
        desc = "A register example.";
        regwidth = 16;
        field {
            name = "Hello";
            hw = rw; sw = r;
            fieldwidth = 8;
        } Hello [15:8] = 42;
        field {
            name = "World";
            hw = rw; sw = rw;
            fieldwidth = 8;
        } World [7:0] = 21;
    } MY_COMPONENT_HELLO_WORLD; // @0x00
};
```

## IEE 1685 and SystemRDL tools


- SystemRDL
  - [CSRCompiler](#) by Semifore
  - Blueprint compiler: Cisco → Denali Software → Cadence → dead
- Register management software
  - [Socrates Bitwise](#) by Doulog
  - [SpectraReg](#) by [PDTi](#) (inactive?)
- [Vregs](#) by Veripool
  - Extracts register definitions from documentation

**EE Live!** Featuring **esc**  
The Embedded Systems Conference

## ARM CMSIS-SVD




- CMSIS: Cortex Microcontroller Software Interface Standard
- SVD: System View Description
- Really just a stripped down IEEE 1685 variant!
- Most Cortex-M manufacturers provide these files which can be downloaded from the manufacturer website or [ARM](#).
- Downside: files from chip manufacturers are usually limited to just register names without the full datasheet descriptions.



**EE Live!** Featuring **esc**  
The Embedded Systems Conference

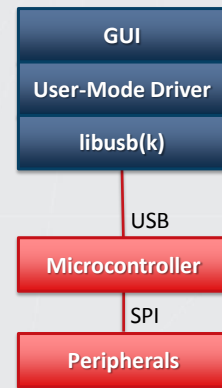
## Agenda

- Introduction
- Interface description languages
- Register specification languages
- Report from the field
- Conclusion and questions



## What I wanted

- Ability to get/set registers and properties
  - From API, GUI and regression tests
  - With usage flags: READ, WRITE, HIDDEN (not for GUI) and some others
- Ability to reliably transfer a number of other short command and control messages
- Protocol Buffers was leading choice for USB serialization, but did not easily support flags and GUI integration



## What I did

- Created a custom IDL (actually Python relying on a custom module)
  - Single parser which heavily relies on Python compiler
  - Multiple generators, one for each use
  - Generators use [Jinja2](#), an excellent Python templating engine

```

lis3dh = Registers('LIS3DH',
                  description='Accelerometer',
                  width=8, bit_order='msb', type='uint8')

lis3dh.Register('STATUS_REG_AUX', mode='r', addr=0x07, bits=[
    Bit('321ODR', description='1, 2 and 3 axis data overrun.', enum=[
        ('inactive', 0, 'no overrun has occurred'),
        ('active', 1, 'a new set of data has overwritten the previous ones')]),
    ...])

lis3dh.Value('OUT_ADC1', mode='r', description='1-axis acceleration data', type='int16')
lis3dh.Register('OUT_ADC1_L', mode='r', addr=0x08, alt='OUT_ADC1[7..0]',
               description='1-axis acceleration data, least significant byte')
lis3dh.Register('OUT_ADC1_H', mode='r', addr=0x09, alt='OUT_ADC1[15..8]',
               description='1-axis acceleration data, most significant byte')
  
```

## Status

- Successfully parses and generates the intermediate code
  - Easy to add and modify parameters / registers
  - Build system automatically generates associated files reliably
- Uses Python as the specification language poses complications
  - Users required to know some Python
  - Without sandboxing (hard in Python), can pose a security risk
- Not as general-purpose as I was hoping
  - Does not sufficiently separate message / register / parameter definition from serialization and transport.
  - Needs a rewrite before being more generally useful
  - ... but much promise!

## Challenges with data-driven interfaces

- Yet another language
  - with yet another compiler
  - Where does data-driven design end and functional design begin?
  - Suffers from least-common denominator issue
  - Does this increase or decrease maintenance cost?
- Poor implementations and confusing standards have sunk many past attempts
- Lack of clear, robust standard and tools across the industry
- Challenge in addressing the full market from small embedded devices to servers

## Promises of data-driven interfaces

- Generation of full tools including register inspection
  - Already used by tools like Atmel Studio
  - Automated parsing of transactions (SPI / I2C / UART / USB / IP) in protocol analyzers with detailed field extraction
  - Automated marshaling and unmarshaling of register transactions across SPI, I2C, USB and network with the same register definitions
- Don't repeat yourself : across the whole industry
- Faster time to market with fewer bugs



## What is an embedded firmware engineer to do?

- Expect more from your suppliers
  - Datasheets and bad examples are no longer enough!
  - Request register definition files, ideally standards-based.
  - Request that definition files contain full descriptive information for every register field.
- When your suppliers fail you, consider your alternatives.
  - Learn more about the options that are available.
  - Create definition files yourself?
- Have ideas or interest in improving the situation? Contact me!





## Conclusion

- Data-driven interface specifications are not new
- The rewards of data-driven interfaces are significant, but the embedded software industry has not yet adopted them
- Name things better: avoid abbreviations
- Write less code by using data to define system interaction
  - Less bug prone
  - Faster
  - Often more flexible
- Can use existing methodologies and tools to start now, but may need to spend time developing a customized solution



## Other References

- Protocol Buffers for embedded
  - [Nanopb](#)
  - [Protobuf-c](#)
  - [Protobuf-embedded-c](#)
- Wikipedia
  - [Interface description languages](#)
  - [Comparison of data serialization formats](#)



**EE Live!**

Featuring **ESC**  
*The Embedded Systems Conference*

**Data-driven Interfaces  
for Embedded Systems**

**Matt Liberty**  
**Jetperch LLC**  
[matt.liberty@jetperch.com](mailto:matt.liberty@jetperch.com)

#eelive

Produced by EE Times

