Improving Hardware Security through Type-Aware Systems Design

Wim Vanderbauwhede

School of Computing
University of Glasgow

12 September 2017

BORDER PATROL
always on guard

Who are we?

Academic Partners

Industrial Partners

Problems with System-on-Chip Construction

TECHNOLOGY

Researcher finds vulnerability in popular microchips used in Android and iPhones

https://www.cyberscoop.com/researcher-finds-vulnerability-popular-microchips-used-android-iphones/

THUNDER STRIKES —

"Thunderstrike 2" rootkit uses Thunderbolt accessories to infect Mac firmware

New version of the exploit can spread via e-mail and infected websites.

ANDREW CUNNINGHAM (US) - 5/8/2015, 22:01

https://arstechnica.co.uk/apple/2015/08/thunderstrike-2-rootkit-uses-thunderbolt-accessories-to-infect-mac-firmware/

RISK ASSESSMENT —

The hijacking flaw that lurked in Intel chips is worse than anyone thought

Patch for severe authentication bypass bug won't be available until next week.

DAN GOODIN (US) - 7/5/2017, 09:05

https://arstechnica.co.uk/security/2017/05/the-hijacking-flaw-that-lurked-in-intel-chips-is-worse-than-anyone-thought/

Securing IoT Medical Devices—Are We There Yet?

To ensure secure communications in a given design, developers must consider integrating key security- and safety-related features that help to harden a medical device against any malicious activity.

Mar 20, 2017

http://www.electronicdesign.com/iot/securing-iot-medical-devices-are-we-there-yet

Aim: Type-Driven Development of SoCs

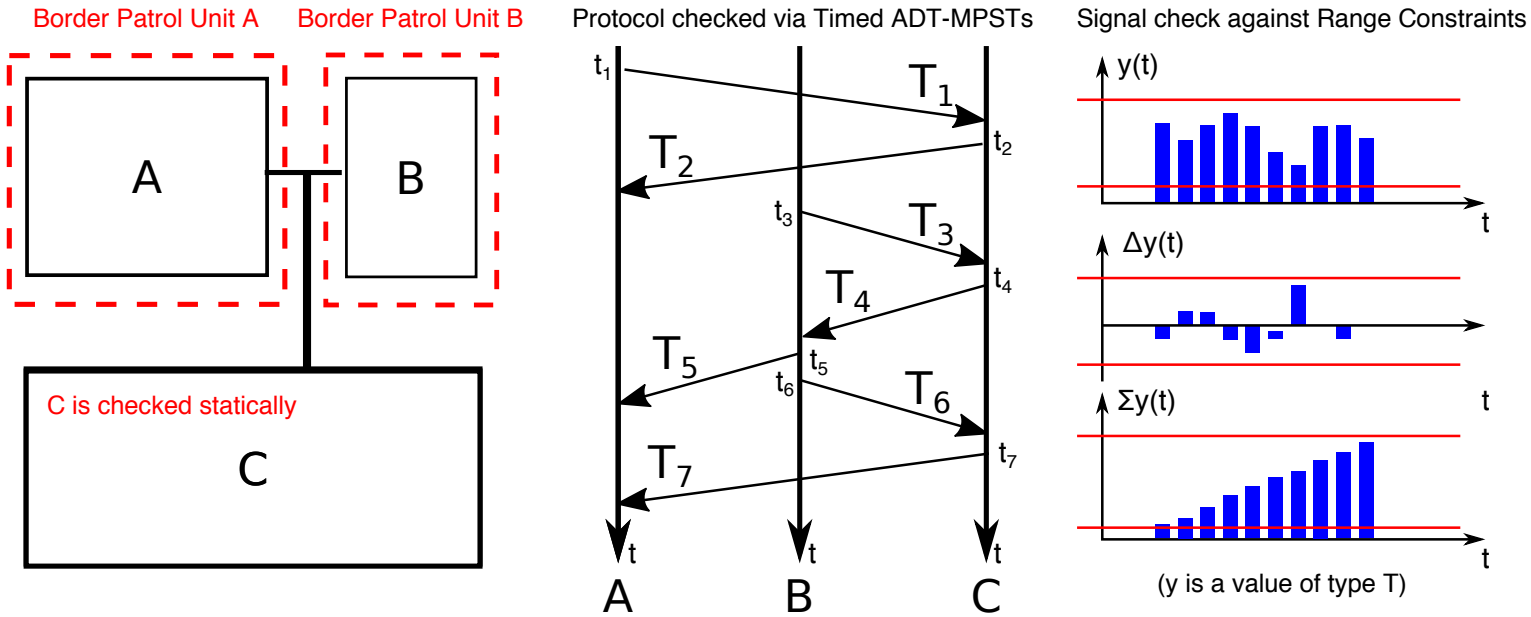
Aim

Use **State-of-the-art** techniques from programming language research to develop a **framework** for **System-on-Chip** development that provides guarantees about the **structure & behaviour** of SoCs at **design & runtime**.

Type-Systems help us:

- Reason about software programs; and
- Describe (and check) a program's structural and behavioural properties:
  - multi-party session types
  - dependent types
  - subtyping
  - hybrid typing

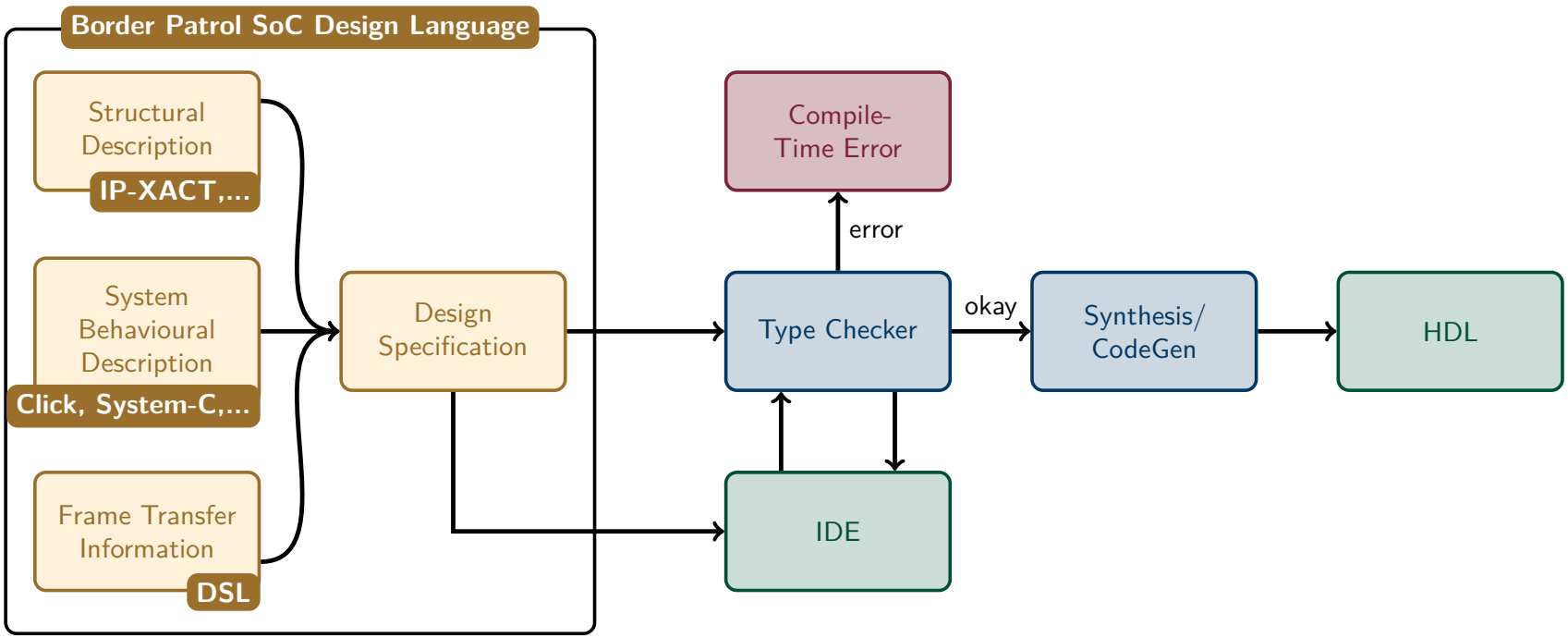
The Idea: Patrolling Hardware Borders



Using type-systems we aim to:

- create machine-checkable interface specifications that describe a SoCs structure and behaviour;
- ensure components interact in known ways using expected values; and
- monitor behaviour of third-party components during runtime against the provided specifications.

Impact on Design Process



- Use existing standards for SoC specification.
- Extend with State-of-the-Art features to describe behaviour.
- Facilitate integration with existing design environments.

Why Dependent Types?

What if types can: depend on values; and be computed?

```
data Nat = Z | S Nat

data Vect : Nat -> Type -> Type where
  Nil : Vect Z elem
  Add : elem -> Vect len elem -> Vect (S len) elem

data ListType = Strings | Numbers

ListOf : Nat -> ListType -> Type
ListOf len Strings = Vect len String
ListOf len Numbers = Vect len Int
```

- All code examples are in **Idris** `www.idris-lang.org`

Why Dependent Types?

More Precise Descriptions of Programs

With more **information** at the type level we can:

- describe **more precise** properties of our software programs.
- **help** programmers to write correct programs.
- **correctness by construction.**

```
(++) : Vect m a -> Vect n a -> Vect (m + n) a
(++) Nil       ys = ys
(++) (Add x xs)  ys = Add x $ xs ++ ys
```

Why Session Types?

Session Types allow us to:

- describe **interactions** between components;
- **separately** from their **implementation**.

Example: TCP Handshake

Implementation

1. $A \to B : (\mathrm{Syn}, x)$
2. $B \to A : (\mathrm{SynAck}, y, x + 1)$
3. $A \to B : (\mathrm{Ack}, y + 1, x + 1)$

Session Type

1. $\mathrm{A} \to \mathrm{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat} \rangle$.
2. $\mathrm{B} \to \mathrm{A} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle$.
3. $\mathrm{A} \to \mathrm{B} : k\langle \mathtt{TCPMsg}, \mathtt{Nat}, \mathtt{Nat} \rangle$. **end**

Why Session & Dependent Types?

Example Session Description

```
Handshake : Session [A,B] [(A,B)] ()
Handshake = do
  activateAll
  chan <- channel A B
  startup chan
  (_,x)   <- send chan A B (TCPMsg SYN, Nat)
  (_,y,_) <- send chan B A (TCPMsg SYNACK, Nat, Next x)
  send chan A B (TCPMsg ACK, Next y, Next x)
  shutdown chan A
  deactivateAll
  end
```

Jan de Muijnck-Hughes et al. 'Type-Driven Development of Communicating Systems using Idris.'. Presented at SPLS in Nov. 2016. Nov. 2016

Why Session & Dependent Types?

Implementation

```
myHandshakeServer : Nat -> Network Handshake B ()
myHandshakeServer y = do
   Result ssock <- newServerSocket           | Error err => fail err
   Result bsock <- bind ssock Nothing 54321  | Error err => fail err
   Result lsock <- listen bsock              | Error err => fail err
   Result (asock,addr) <- accept lsock       | Error err => fail err

   Result (msg, x) <- recv asock | Error err => fail err
   io $ putStrLn (unwords ["Received:", show msg])

   Success <- send asock (MkSynAck, y, (S x ** Refl)) | Error err => fail err
   io $ putStrLn (unwords ["Sent: SYNACK", show (S x)])

   Result (msg1, msg2, msg3) <- recv asock | Error err => fail err
   io $ putStrLn (unwords ["Received:", show msg1, show msg2, show msg3])

   unaccept asock
   theyClose lsock

main : IO ()
main = run (myHandshakeServer 5)
```

Jan de Muijnck-Hughes et al. 'Type-Driven Development of Communicating Systems using Idris.'. Presented at SPLS in Nov. 2016. Nov. 2016

Runtime Type Checking

From Session Types to Finite State Machines

- **Multiparty Session Types** can be compiled into FSMs:
  - semantic correspondence between MPSTs and communicating FSMs;
  - projections on each end point can be used as run-time protocol checkers.
- **Dependent Types** can add value checking to the FSMs:
  - dependent types can express bounds of values through linear inequalities;
  - can trivially be extended to time derivatives and integrals

Pierre-Malo Deniélou et al. 'Multiparty Session Types Meet Communicating Automata.'. In: *ESOP*. vol. 12. Springer. 2012, pp. 194–213

Subtyping

Compatibility between interfaces

- **Interface compatibility**
  - IP cores can use different interface standards
  - Requires glue logic to connect different interfaces
  - What if this glue logic could be inferred from the types?
  - This is possible through **subtyping**
- **Subtyping**
  - Structural subtyping: the structure of two types determines whether or not one is a subtype of the other
  - In particular, **algebraic data types** (sum and product types)
  - If actual structural subtyping is not a priori possible, provide a **proof of equivalence**
  - Thus, proofs are formalisations of glue logic

Benjamin C Pierce. *Types and programming languages*. MIT press, 2002

Border Patrol

In practice:

- Create a new *contract* language, integrated in e.g. SDSoC
  - Familiar syntax (SystemVerilog, IP-XACT,...)
  - But compiles to our type descriptions
- All modules in a design (IP cores) come with a contract in this language
- Trusted modules are checked at design time
- Untrusted modules are checked at run time
- Note that this approach extends to software

Border Patrol

Using Border Patrol we look to:

- increase safety and security of smart devices; and
- help safeguard critical infrastructure.

How does it benefit you?

- **Academic Research**:
  - Advances in type-theory, programming language design and compilation for HW/SW for System-on-Chip.
- **Industrial Research**:
  - Novel approach to SoC Design: Safer, more secure, more productive.
- **Consumers**:
  - Guarantees of Safety & Security; Checkable adherence to specifications.

Thank you!

BORDER PATROL
always on guard

https://border-patrol.github.io/

Want to learn more about type systems?

Learn Haskell!

Future Learn

Categories Courses Programs Degrees

Sign in

ONLINE COURSE

Functional Programming in Haskell: Supercharge Your Coding

University of Glasgow

Get an introduction to Haskell, the increasingly popular functional programming language, with this University of Glasgow course.

Join free

Upgrade - £59

+ shipping

What's the difference?