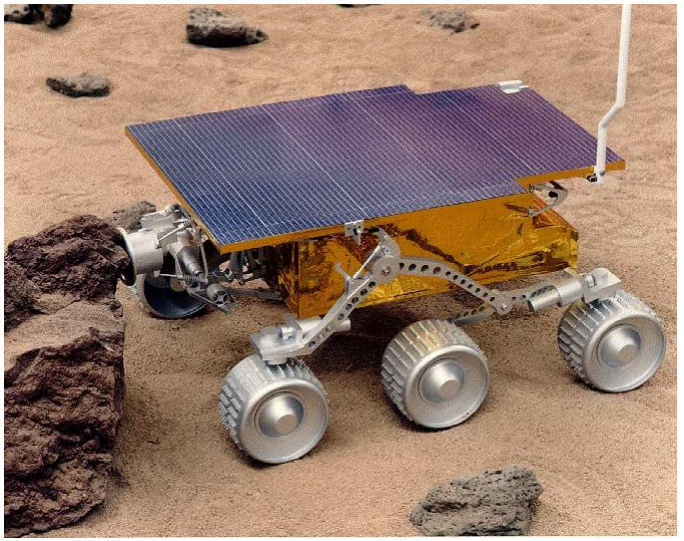
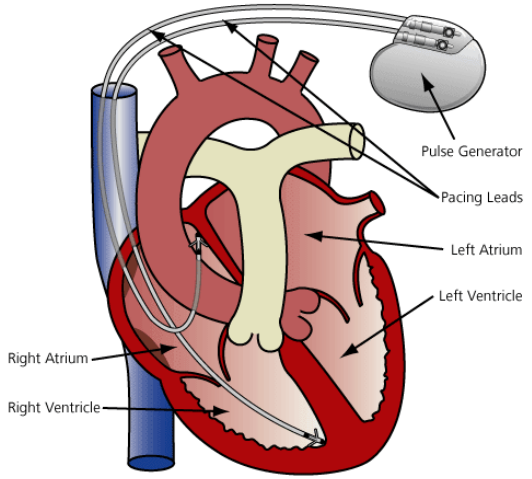
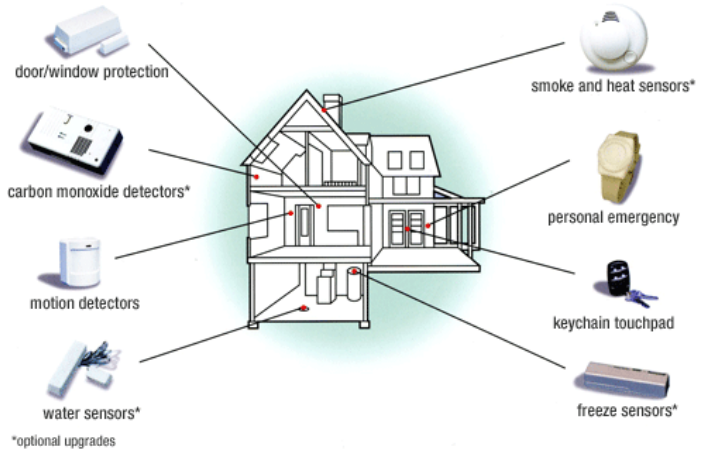
Correctness Proofs for Device Drivers in Embedded Systems

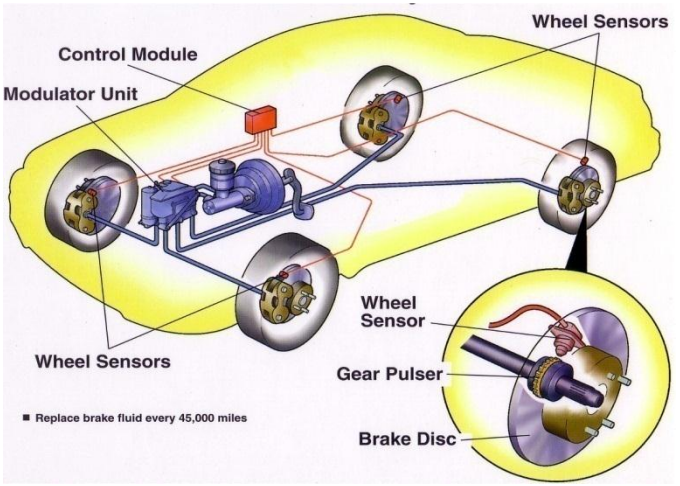
Jianjun Duan, John Regehr
School of Computing
University of Utah
Oct. 7, 2010

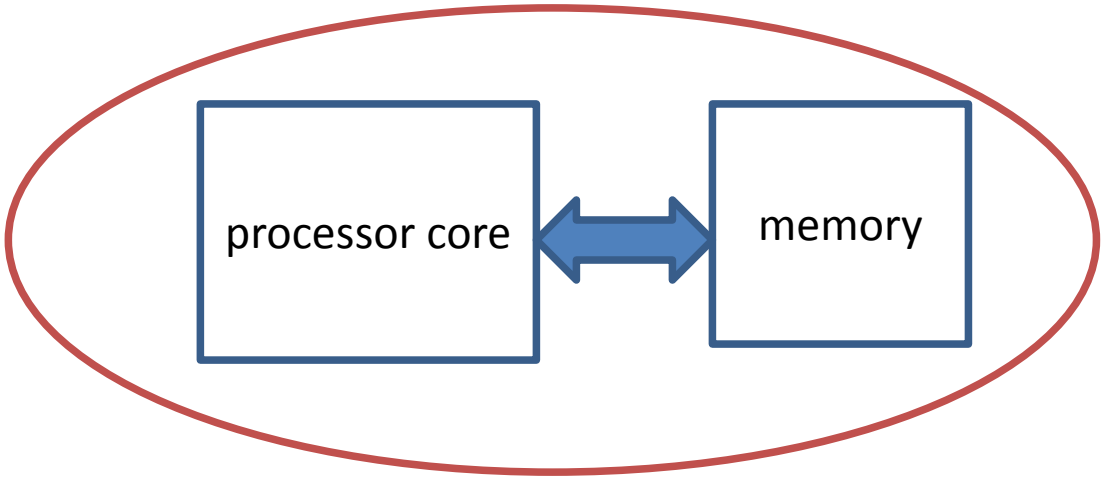
Embedded systems





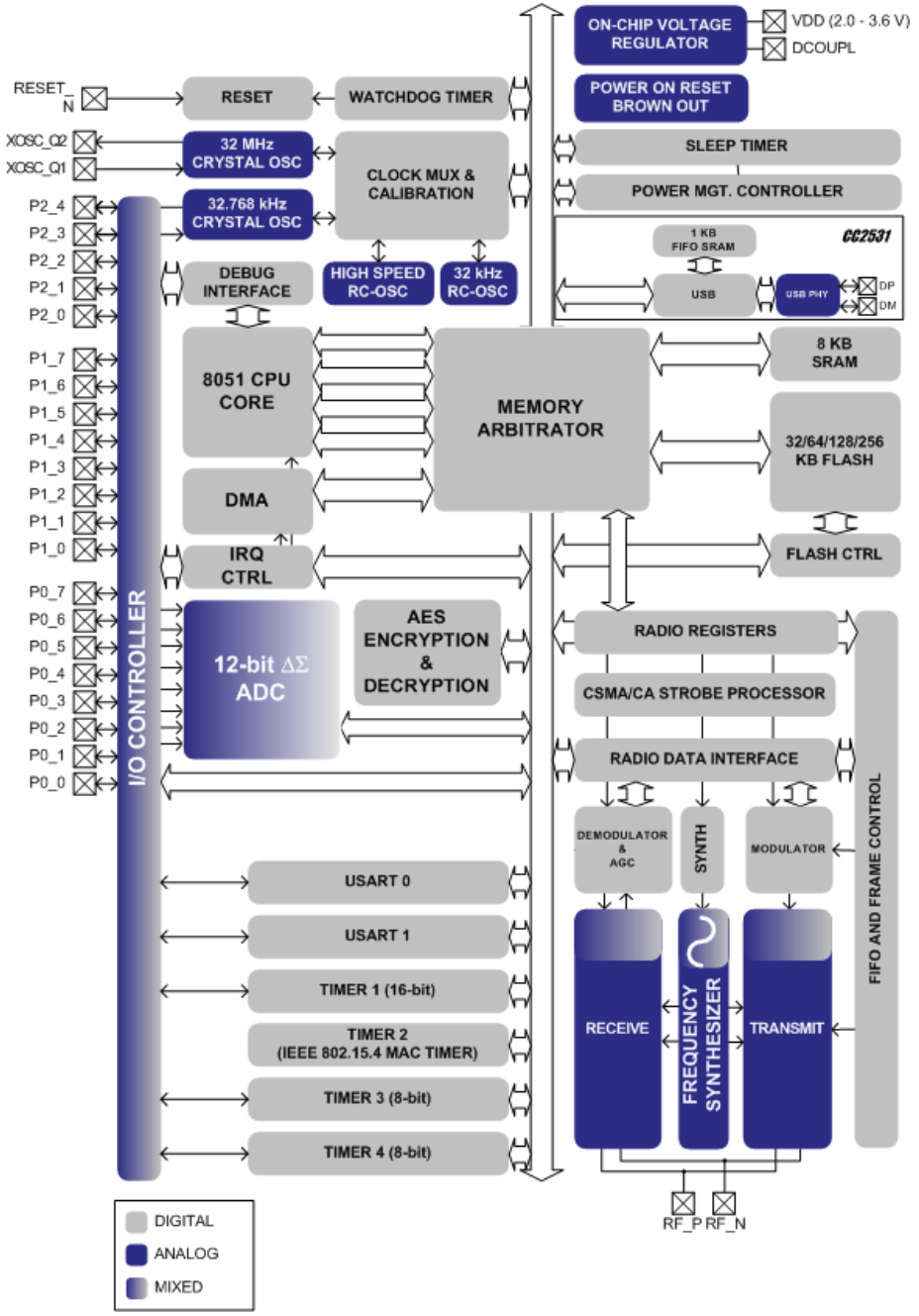


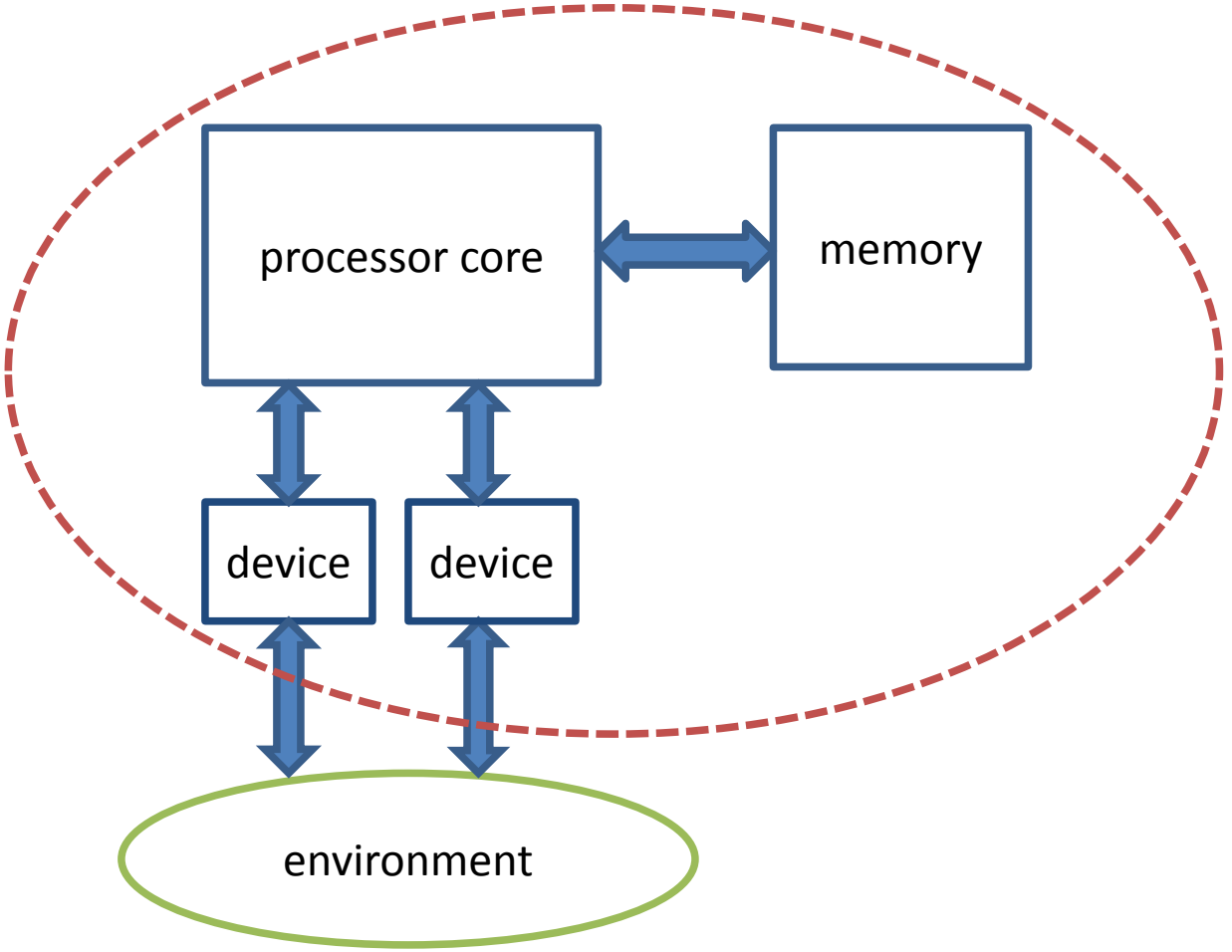




Cambridge
ARM model

Fox (2003)

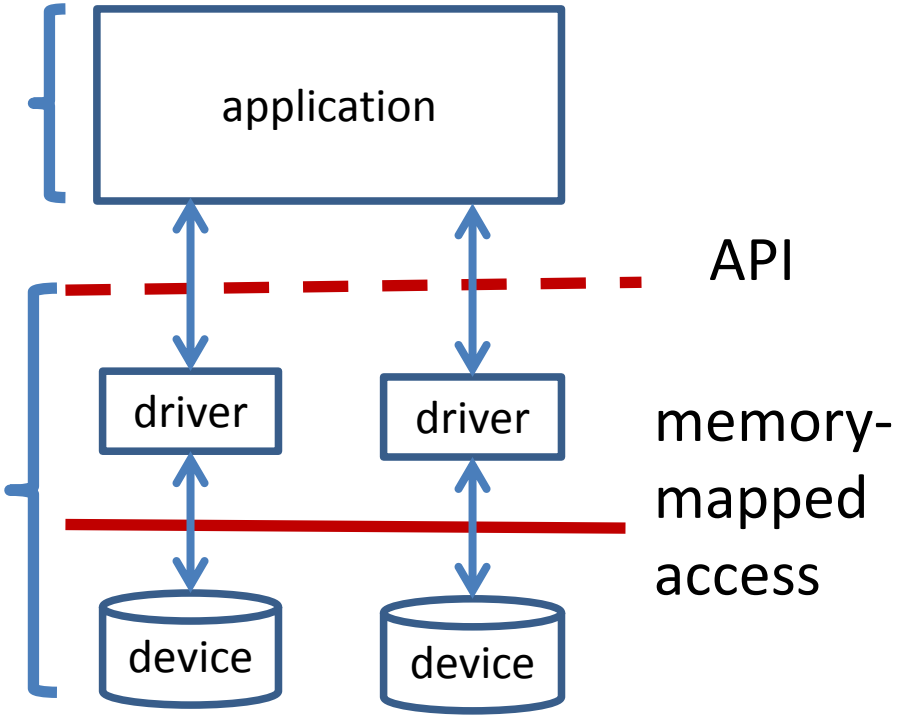

ON-CHIP VOLTAGE REGULATOR
VDD (2.0 - 3.6 V)
DCOUPL
POWER ON RESET BROWN OUT
RESET_N
RESET
WATCHDOG TIMER
XOSC_Q2
XOSC_Q1
32 MHz CRYSTAL OSC
CLOCK MUX & CALIBRATION
SLEEP TIMER
POWER MGT. CONTROLLER
P2_4
P2_3
P2_2
P2_1
P2_0
32.768 kHz CRYSTAL OSC
1 KB FIFO SRAM
CC2531
USB
USB PHY
DP
DM
DEBUG INTERFACE
HIGH SPEED RC-OSC
32 kHz RC-OSC
P1_7
P1_6
P1_5
P1_4
P1_3
P1_2
P1_1
P1_0
8051 CPU CORE
MEMORY ARBITRATOR
8 KB SRAM
32/64/128/256 KB FLASH
DMA
IRQ CTRL
FLASH CTRL
P0_7
P0_6
P0_5
P0_4
P0_3
P0_2
P0_1
P0_0
I/O CONTROLLER
12-bit ΔΣ ADC
AES ENCRYPTION & DECRYPTION
RADIO REGISTERS
CSMA/CA STROBE PROCESSOR
RADIO DATA INTERFACE
DEMODULATOR & AGC
SYNTH
MODULATOR
FIFO AND FRAME CONTROL
USART 0
USART 1
TIMER 1 (16-bit)
TIMER 2 (IEEE 802.15.4 MAC TIMER)
TIMER 3 (8-bit)
TIMER 4 (8-bit)
RECEIVE
FREQUENCY SYNTHESIZER
TRANSMIT
RF_P
RF_N
DIGITAL
ANALOG
MIXED


processor core
memory
device
device
environment

Layered proof

- application correctness
- automation and scalability

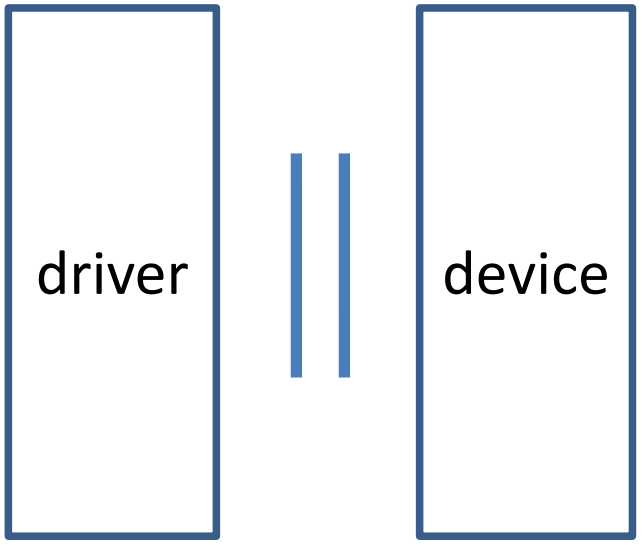
- functional correctness of device drivers
- reasoning about timing
- not much automation



Our work

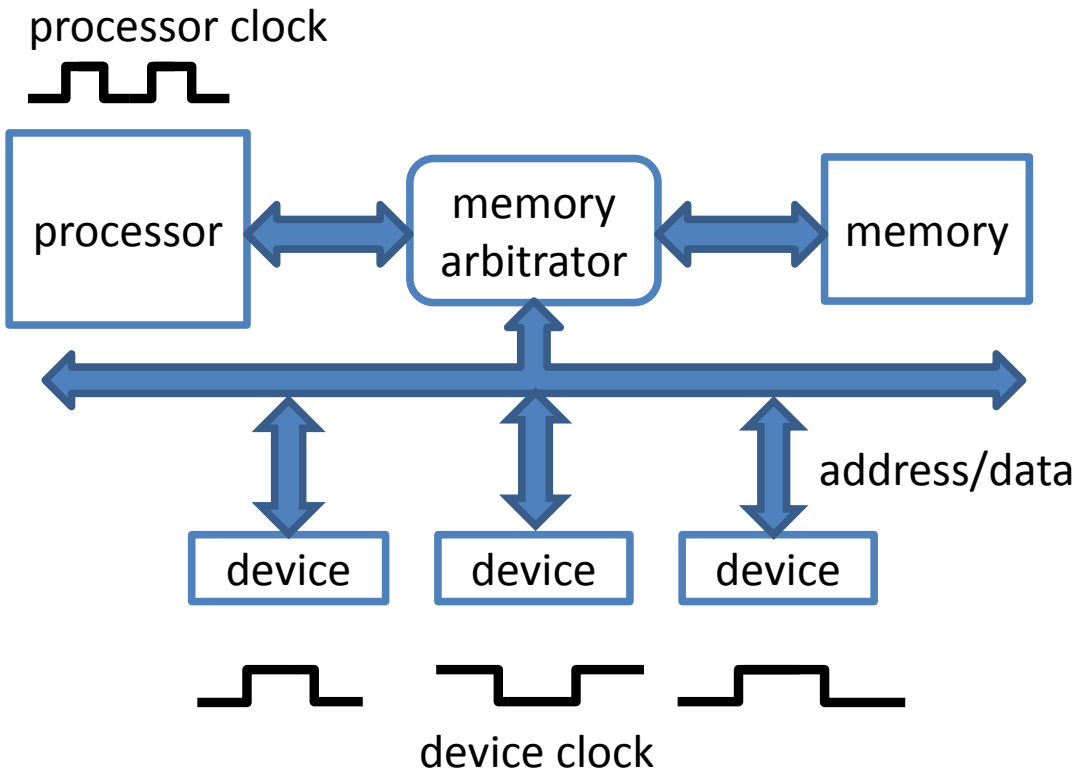
- Abstract device model to be plugged into an instruction set architecture model
- A realistic serial port (UART) model
- Strong properties including timing constraints
- Full correctness proof for a UART driver
- Implementation based on ARM v6 model in HOL4

Related work



- Alkassar et al. (2007, 2008), Monniaux (2007)
- Difficult to reason about timing property

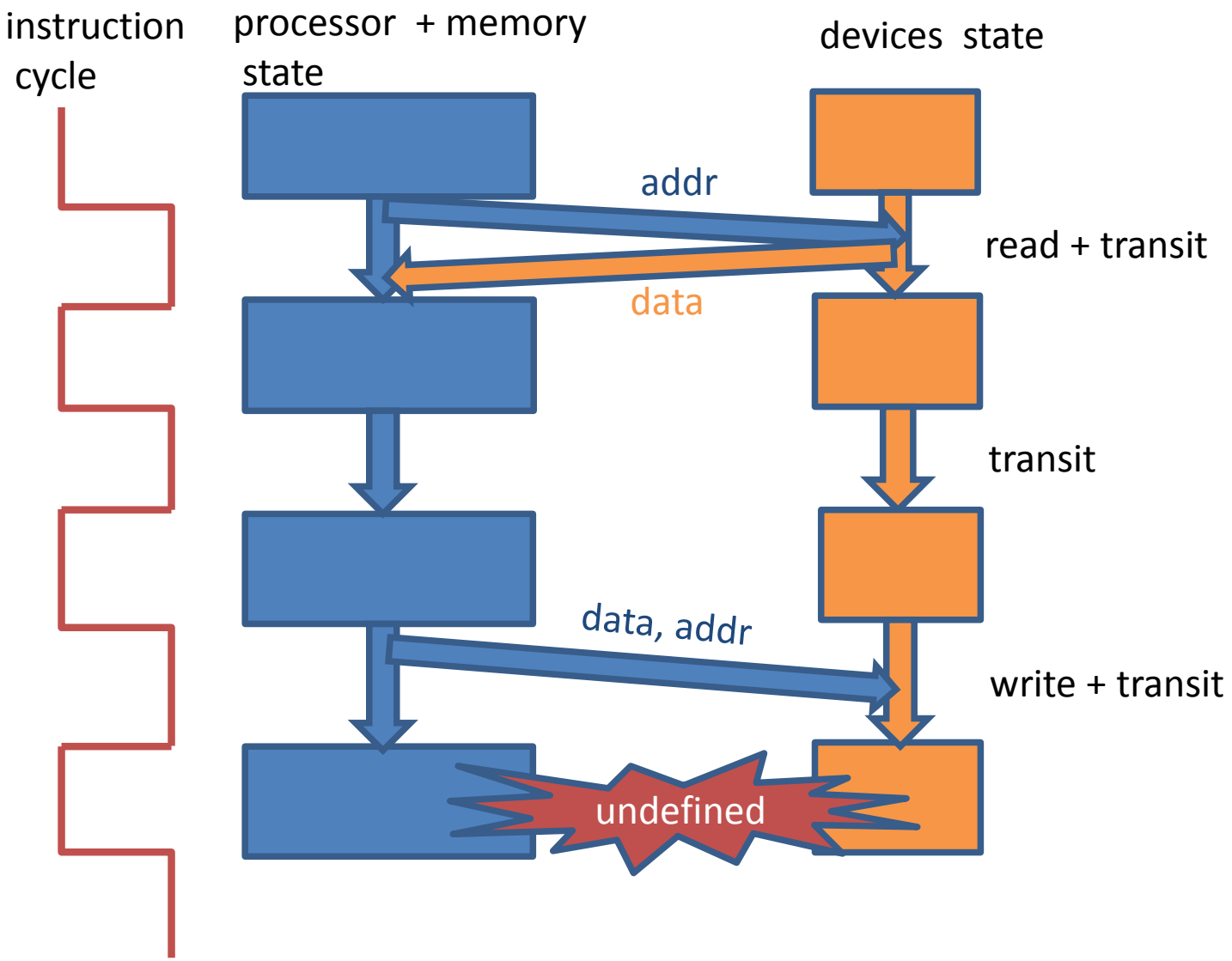
System with devices



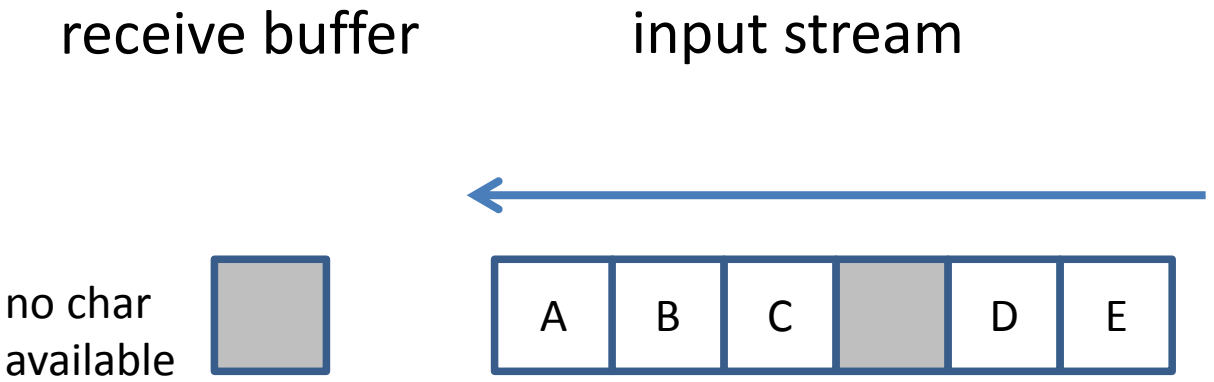
Device model

- Set of memory-mapped device registers : $addr \longrightarrow bool$
- Read effect : $(addr * \tau) \longrightarrow (word * bool * \tau)$
- Write effect : $(addr * word * \tau) \longrightarrow (bool * \tau)$
- Autonomous transition : $\tau \longrightarrow \tau$
- Device state must be well-formed : $\tau \longrightarrow bool$

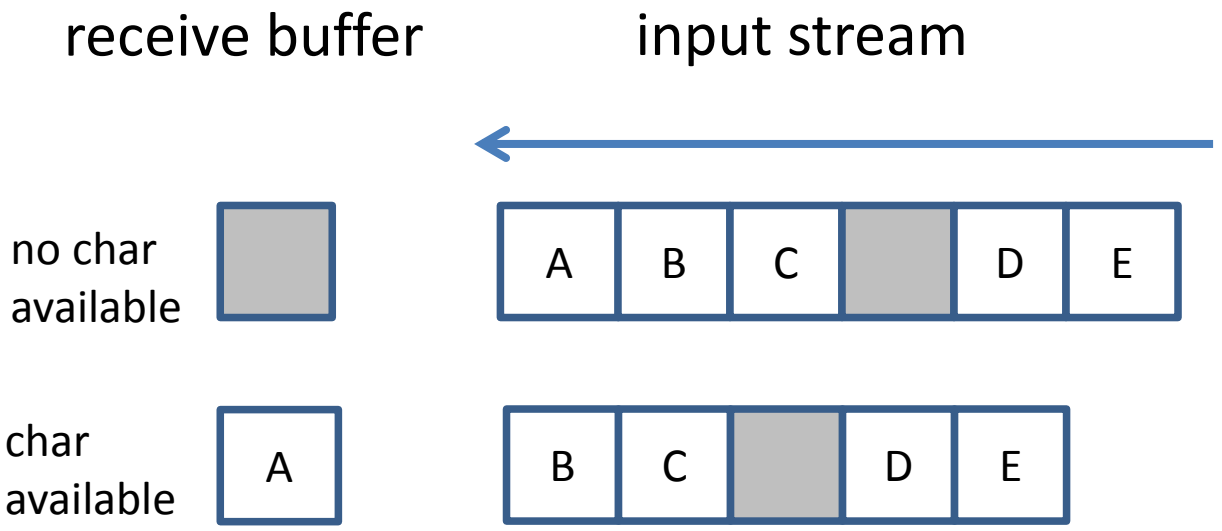
Parallel state transition



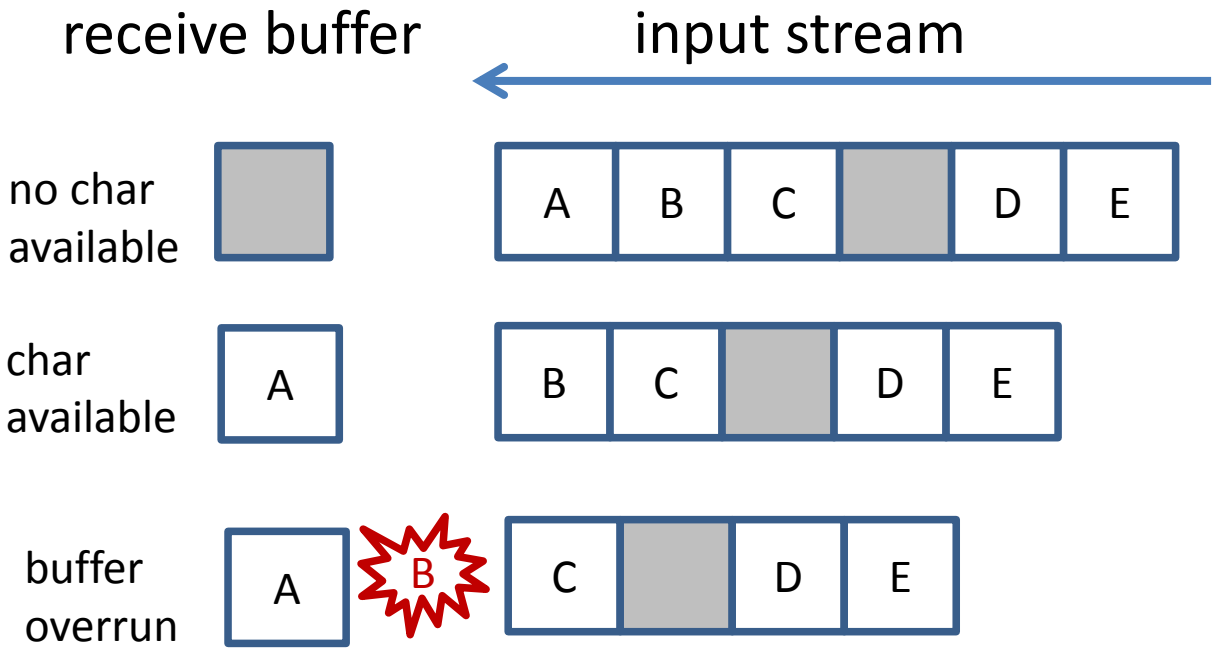
UART input stream



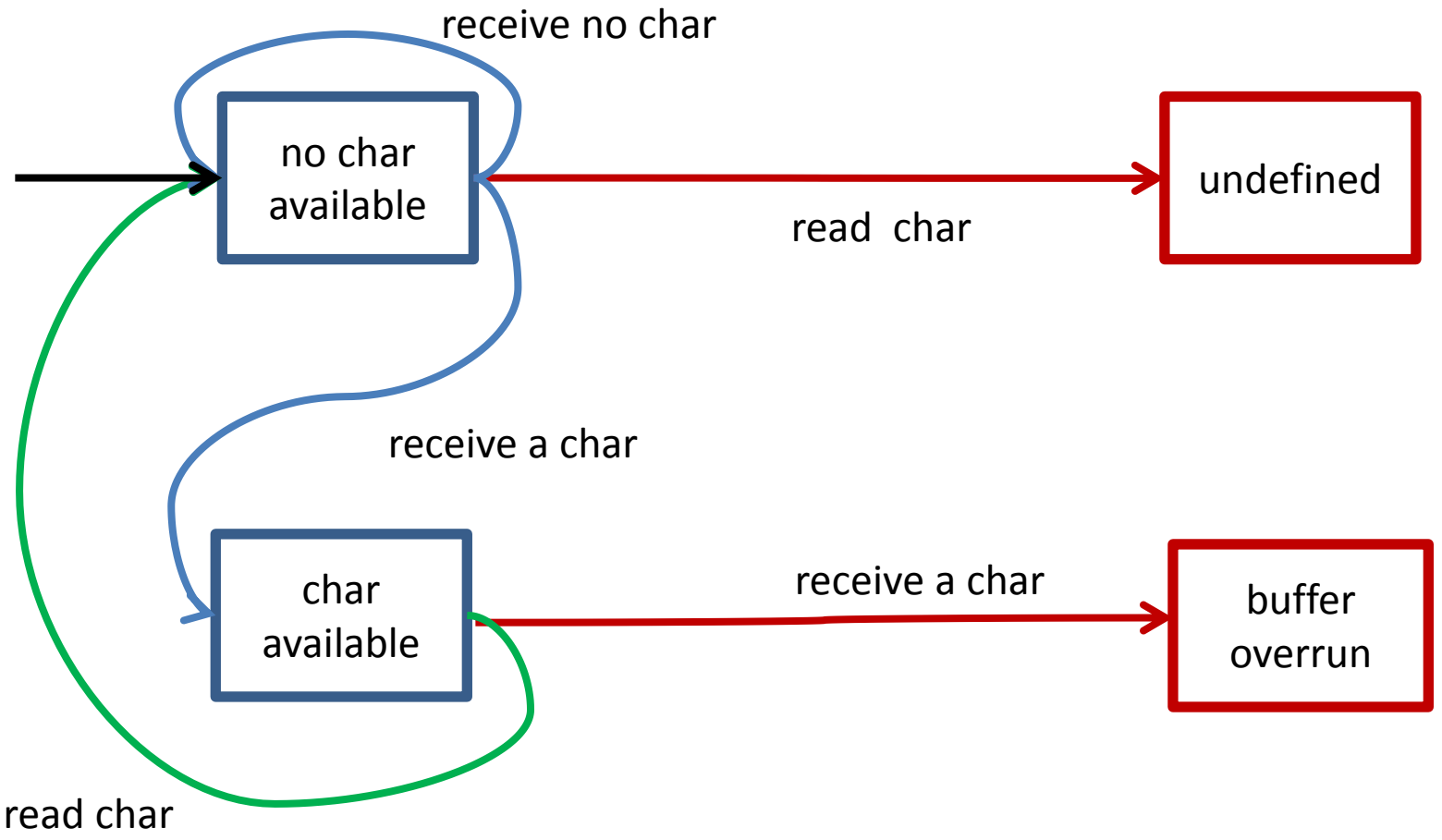
UART input stream



UART input stream



UART receive state machine



Can the driver avoid buffer overrun?

REGISTER DESCRIPTION

Table 74: UART0 Register Map

Name	Description	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	Access	Reset Value*	Address
U0RBR	Receiver Buffer Register	MSB			READ DATA				LSB	RO	un-defined	0xE000C000 DLAB = 0
U0THR	Transmit Holding Register	MSB			WRITE DATA				LSB	WO	NA	0xE000C000 DLAB = 0
U0IER	Interrupt Enable Register	0	0	0	0	0	Enable Rx Line Status Interrupt	Enable THRE Interrupt	Enable Rx Data Available Interrupt	R/W	0	0xE000C004 DLAB = 0
U0IIR	Interrupt ID Register	FIFOs Enabled		0	0	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE000C008
U0FCR	FIFO Control Register	Rx Trigger		Reserved		-	Tx FIFO Reset	Rx FIFO Reset	FIFO Enable	WO	0	0xE000C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0	0xE000C00C
U0LSR	Line Status Register	Rx FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000C014
U0SCR	Scratch Pad Register	MSB							LSB	R/W	0	0xE000C01C
U0DLL	Divisor Latch LSB	MSB							LSB	R/W	0x01	0xE000C000 DLAB = 1
U0DLM	Divisor Latch MSB	MSB							LSB	R/W	0	0xE000C004 DLAB = 1

*Reset Value refers to the data stored in used bits only. It does not include reserved bits content.

UART0 contains ten 8-bit registers as shown in Table 74. The Divisor Latch Access Bit (DLAB) is contained in U0LCR7 and enables access to the Divisor Latches.

UART0 Receiver Buffer Register (U0RBR - 0xE000C000 when DLAB = 0, Read Only)

The U0RBR is the top byte of the UART0 Rx FIFO. The top byte of the Rx FIFO contains the oldest character received and can be read via the bus interface. The LSB (bit 0) represents the "oldest" received data bit. If the character received is less than 8 bits, the unused MSBs are padded with zeroes.

The Divisor Latch Access Bit (DLAB) in U0LCR must be zero in order to access the U0RBR. The U0RBR is always Read Only.

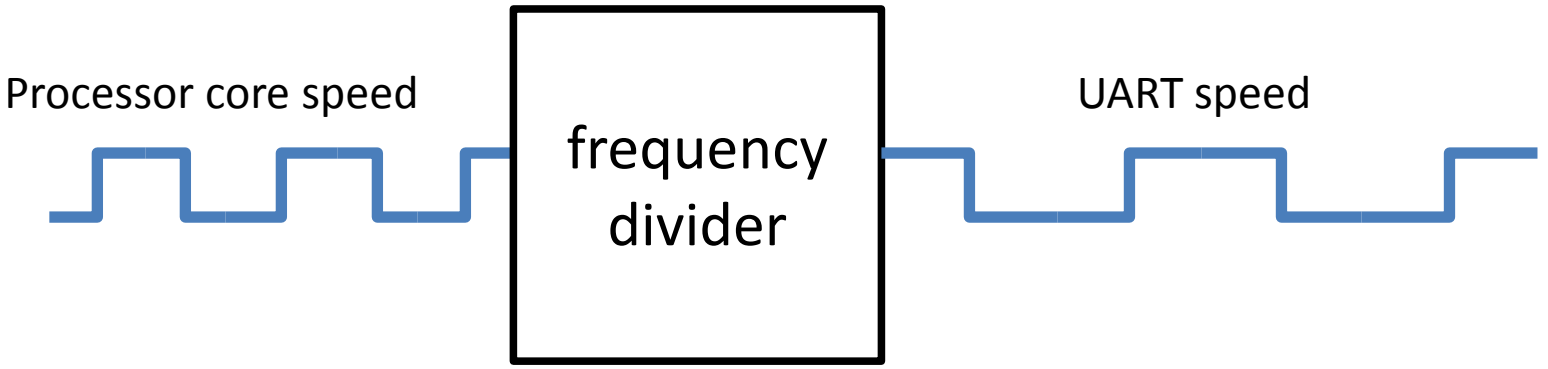
- LPC2129 is based on ARM7TDMI-S
- 306 page manual
- 12 pages for UART0

LPC2129 UART0 model

- Functional model at character level
- Side effect and undefined behavior of memory-mapped access of registers
- Speed of UART relative to the processor

  allows timing properties to be expressed
- Buffer size = 1
- No interrupt support

Modeling UART speed



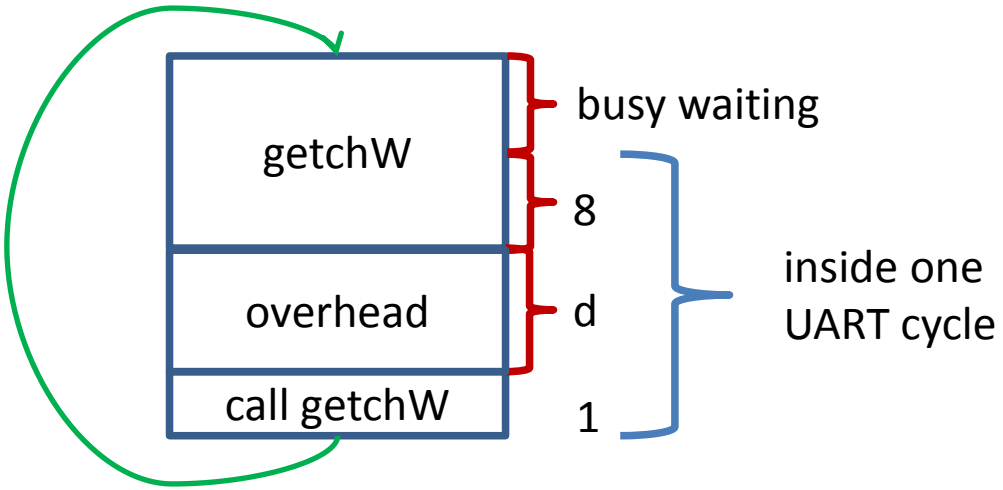
```
<getchW>:
ldr    r2,  uart0rbr
ldrb   r3,  [r2, #20]
tst    r3,  #1
beq    <getchW>
ldrb   r0,  [r2]
mov    pc,  lr
```

```
<getch>:
ldr      r2, uart0rbr
ldrb     r3, [r2, #20]
tst      r3, #1
ldrneb   r3, [r2]
mvn      r0, #0
andne    r0, r3, #255
mov      pc, lr
```

```
<putch>:
ldr    r2, uart0rbr
ldrb   r3, [r2, #20]
tst    r3, #32
beq    <putch>
and    r0, r0, #225
strb   r0, [r2]
mov    pc, lr
```

Compiled from open-source C code

Receive a string using getchW



UART speed divider > 9 + d

Correctness of getchW

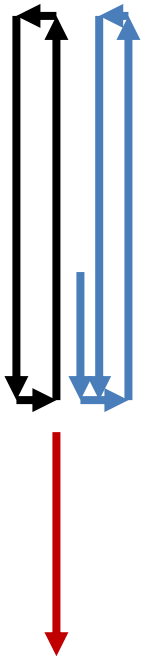
- UART speed divider > 9 + d
- Pre: pc points to getchW, receive buffer accessible, no char available
- Post: getchW returns, reads the first char from the input queue, no receive action for d + 1 cycles
- Invariant: no buffer overrun, safety property is observed

Correctness of getch and putch

- putch appends the character to the string already sent out in the output queue.
- getch reads a character from the input queue or returns an error code.
- Safety invariant: UART configuration is not changed, memory safety is observed, no undefined behavior

Proof method

- Busy waiting until char is available (not a static point)
- Loop exit (char available)
- Copy receive buffer and return

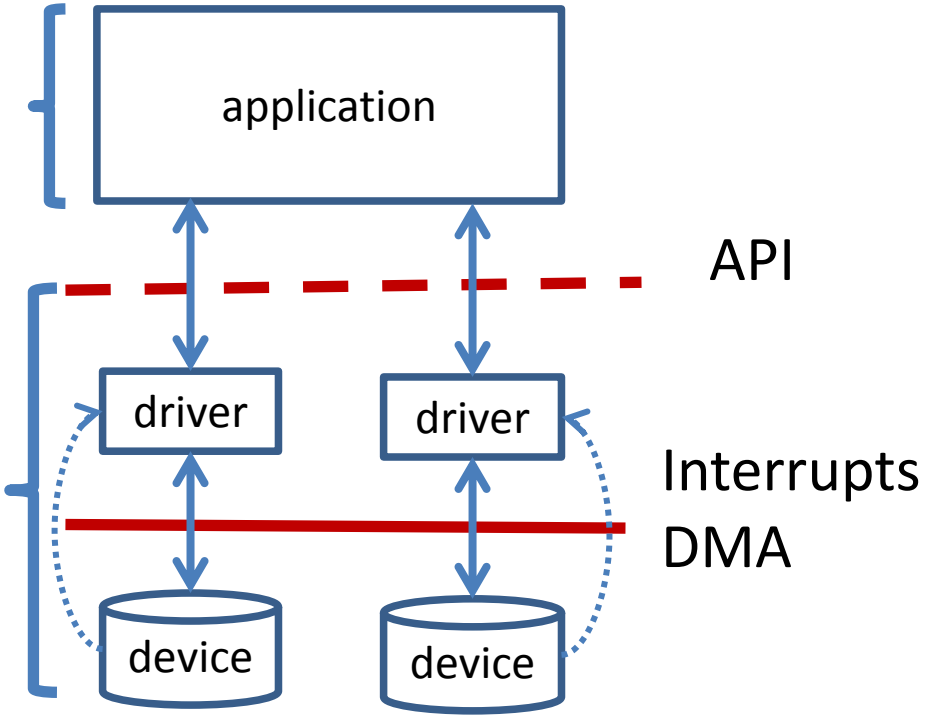


```
<getchW>:
ldr    r2,  uart0rbr
ldrb   r3,  [r2, #20]
tst    r3,  #1
beq    <getchW>
ldrb   r0,  [r2]
mov    pc,  lr
```

Layered proof

- application correctness
- automation and scalability

- functional correctness of device drivers
- reasoning about timing
- not much automation



Contributions

- A framework for proving correctness of device drivers in embedded systems
- A realistic UART model to work with the ARM model in HOL4
- Full correctness of character level receive and send functions in a realistic UART driver, including timing constraints