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)
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

- processor
- memory arbiter
- memory
- device
- device
- device

processor clock

address/data

device clock
Device model

- Set of memory-mapped device registers: \( \text{addr} \rightarrow \text{bool} \)
- Read effect: \((\text{addr} \times \tau) \rightarrow (\text{word} \times \text{bool} \times \tau)\)
- Write effect: \((\text{addr} \times \text{word} \times \tau) \rightarrow (\text{bool} \times \tau)\)
- Autonomous transition: \(\tau \rightarrow \tau\)
- Device state must be well-formed: \(\tau \rightarrow \text{bool}\)
Parallel state transition
UART input stream

receive buffer

input stream

no char available

A B C D E
UART input stream

receive buffer

no char available

char available

input stream

A B C D E

A B C D E

B C D E

available
UART input stream

receive buffer

no char available

char available

buffer overrun

input stream

A B C D E

A B C D E

A B C D E
UART receive state machine

Can the driver avoid buffer overrun?
**REGISTER DESCRIPTION**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Access</th>
<th>Reset Value*</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>U0RBR</td>
<td>Receiver Buffer Register</td>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RO</td>
<td>undefined</td>
<td>0xE000C000 DLAB = 0</td>
</tr>
<tr>
<td>U0THR</td>
<td>Transmit Holding Register</td>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>WO</td>
<td>NA</td>
<td>0xE000C000 DLAB = 0</td>
</tr>
<tr>
<td>U0IER</td>
<td>Interrupt Enable Register</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>RW</td>
<td>0xE000C004 DLAB = 0</td>
<td></td>
</tr>
<tr>
<td>U0IR</td>
<td>Interrupt ID Register</td>
<td>FIFOs Enabled</td>
<td>0</td>
<td>0</td>
<td>IIR3</td>
<td>IIR2</td>
<td>IIR1</td>
<td>IIR0</td>
<td>RO</td>
<td>0x01</td>
<td>0xE000C008</td>
<td></td>
</tr>
<tr>
<td>U0FCR</td>
<td>FIFO Control Register</td>
<td>Rx Trigger</td>
<td>Reserved</td>
<td>-</td>
<td>Enable Rx Frame</td>
<td>Enable TxFIFO</td>
<td>Enable Rx Error</td>
<td>Enable TxFIFO</td>
<td>RW</td>
<td>0xE000C008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>U0LOR</td>
<td>Line Control Register</td>
<td>DLAB</td>
<td>Set</td>
<td>Break</td>
<td>Stick</td>
<td>Parity</td>
<td>Even Parity</td>
<td>Parity</td>
<td>Word Length</td>
<td>RW</td>
<td>0xE000C00C</td>
<td></td>
</tr>
<tr>
<td>U0LSR</td>
<td>Line Status Register</td>
<td>Rx FIFO Error</td>
<td>TEMT</td>
<td>THRE</td>
<td>Bi</td>
<td>FE</td>
<td>PE</td>
<td>OE</td>
<td>DR</td>
<td>RO</td>
<td>0xE000C014</td>
<td></td>
</tr>
<tr>
<td>U0SCR</td>
<td>Scratch Pad Register</td>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RW</td>
<td>0xE000C01C</td>
<td></td>
</tr>
<tr>
<td>U0DLR</td>
<td>Divisor Latch LSB</td>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RW</td>
<td>0xE000C000 DLAB = 1</td>
<td></td>
</tr>
<tr>
<td>U0DLM</td>
<td>Divisor Latch MSB</td>
<td>MSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RW</td>
<td>0xE000C004 DLAB = 1</td>
<td></td>
</tr>
</tbody>
</table>

*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 U0LOR 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 U0LOR 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

Processor core speed

frequency divider

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

API
Interrupts
DMA
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