Optimizing a Domain-Specific Language for Device Drivers

Christopher L. Conway

29 April 2003
NDL: A Network Device Language

A high-level, domain-specific language that compiles to C.

A device specification is typically:
• A set of memory-mapped special-purpose registers
• Protocols for accessing registers
• A collection of device functions

NOTE: “device registers” reside in memory, accessing device registers is slow
FSM Behavior

Aspects of device behavior can be described using a finite state machine.

```plaintext
state PAGE(i : int{0..2}) { registerPage = i ; }

state STOPPED {
    goto DMA_DISABLED ;
    stop = true ;
}

state DMA_DISABLED { dmaState = DISABLED ; }

state DMA_READING { goto STARTED ; dmaState = READING ; }

state DMA_WRITING { goto STARTED ; dmaState = WRITING ; }
```
Register Definitions

State predicates control overlay registers

```python
( PAGE(0) ) => {
    write dataConfig = {
        dmaTransferWidth : {
            BYTE_WIDE=#0
            WORD_WIDE=#1
        },
        byteOrder : {
            LITTLE_ENDIAN = #0
            BIG_ENDIAN = #1
        },
        dmaAddrMode : {
            DUAL_16 = #0
            SINGLE_32 = #1
        },
        loopbackDisabled,
    }
    autoInitRemote,
    fifoThreshold : {
        FILLED_2 = #00
        FILLED_4 = #01
        FILLED_8 = #10
        FILLED_12 = #11
    },
    _ : bit
}

( PAGE(1) ) => {
    currentPage : byte
}
```
Register Definitions

Registers have sub-registers

```plaintext
[ ( PAGE(0) )=> { write dataConfig = {
  dmaTransferWidth : {
    BYTE_WIDE=#0
    WORD_WIDE=#1
  },
  byteOrder : {
    LITTLE_ENDIAN = #0
    BIG_ENDIAN = #1
  },
  dmaAddrMode : {
    DUAL_16 = #0
    SINGLE_32 = #1
  },
  loopbackDisabled,
  autoInitRemote,
  fifoThreshold : {
    FILLED_2   = #00
    FILLED_4   = #01
    FILLED_8   = #10
    FILLED_12  = #11
  },
  _ : bit
}
} ]
```
Device Functions

Device interaction is imperative; reads to and writes from registers

```javascript
function init() {
    goto STOPPED ;

    fifoThreshold = FILLED_8 ;
    autoInitRemote = false ;
    loopbackDisabled = true ;
    dmaAddrMode = DUAL_16 ;
    byteOrder = LITTLE_ENDIAN ;
    dmaTransferWidth = WORD_WIDE ;

    /* ... */

    currentPage = RX_START_PAGE ;
}
```
Intermediate Representation

Typical three-address code for expressions and assignment.

Retains control-flow structures for translation into C.

LOAD main memory → device register
STORE device register → main memory

State assertions are expanded inline:

\[
\text{goto PAGE}(n) \rightarrow \text{registerPage} = n \rightarrow \text{STORE} \text{command}[6:7], n
\]
function init() {
  goto STOPPED;

  fifoThreshold = FILLED_8;
  autoInitRemote = false;
  loopbackDisabled = true;
  dmaAddrMode = DUAL_16;
  byteOrder = LITTLE_ENDIAN;
  dmaTransferWidth = WORD_WIDE;

  /* ... */

  currentPage = RX_START_PAGE;
}

{ STORE command[5], 1
  STORE command[0], 1
  STORE command[6:7], 0
  STORE dataConfig[5:6], 2
  STORE command[6:7], 0
  STORE dataConfig[4:4], 0
  STORE command[6:7], 0
  STORE dataConfig[3:3], 1
  STORE command[6:7], 0
  STORE dataConfig[2:2], 0
  STORE command[6:7], 0
  STORE dataConfig[1:1], 0
  STORE command[6:7], 0
  STORE dataConfig[0:0], 1
  /* ... */
  STORE command[6:7], 1
  STORE currentPage[0:7], 76
}
Optimization

• The key performance metric is I/O operations - we want to minimize register read/writes

• Leave low-level considerations to the C compiler

• Leverage domain knowledge + rich semantic information

• Pitfall: Device functions are sensitive to instruction reordering
Optimization: Register Attributes

command = {
    stop: trigger except 0,
    start: trigger except 0,
    transmit: trigger except 0,
    dmaState : {
        READING = #001
        WRITING = #010
        SENDING = #011
        DISABLED = #1**
    } volatile,
    registerPage : int{0..2}
}

Register write causes an event
Register value may change between reads
Default: Register has traditional state-holding properties (idempotence)
Optimization: Idempotence

Repeated writes to idempotent register values can be pruned

```
STORE command[3:5], 4
STORE command[0], 1
STORE command[6:7], 0
STORE dataConfig[5:6], 2
STORE command[6:7], 0
STORE dataConfig[4:4], 0
STORE command[6:7], 0
STORE dataConfig[3:3], 1
STORE command[6:7], 0
STORE dataConfig[2:2], 0
STORE command[6:7], 0
STORE dataConfig[1:1], 0
STORE command[6:7], 0
STORE dataConfig[0:0], 1
/* ... */
STORE command[6:7], 1
STORE currentPage[0:7], 76
```
Optimization: Sub-registers

Orthogonal writes to the same register can be consolidated.

```plaintext
STORE command[3:5], 4
STORE command[0], 1
STORE command[6:7], 0
STORE dataConfig[5:6], 2
STORE dataConfig[4:4], 0
STORE dataConfig[3:3], 1
STORE dataConfig[2:2], 0
STORE dataConfig[1:1], 0
STORE dataConfig[0:0], 1
/* ... */
STORE command[6:7], 1
STORE currentPage[0:7], 76
```

```plaintext
STORE command[0,3:7], 0x11
STORE dataConfig[0:6], 0x49
/* ... */
STORE command[6:7], 1
STORE currentPage[0:7], 76
```
Optimization: Sub-registers 2

Partial register writes require read-modify-write = 2 I/O ops

- LOAD tmp1, command
- tmp2 = (tmp1 & 0x06) | 0x11
- STORE command, tmp2
- LOAD tmp3, dataConfig
- tmp4 = (tmp3 & 0x80) | 0x49
- STORE dataConfig, tmp4

Use knowledge of don’t-cares and write-neutral values:

- command[1] = stop: trigger except 0
- command[2] = transmit: trigger except 0

- STORE command, 0x11
- STORE dataConfig, 0x49
More Optimization

• Device functions are inlined - relatively few functions calls + no recursion make this practical

• Multiple reads from state-holding registers could be cached

• Apply all techniques using DFA methods (currently works within basic blocks)

• Effectiveness: TBD