Implementing an interface for virtual input devices into the MGSim simulator

Koen Putman [1] (Author) Raphael Poss [2] (Supervisor)

[1]LIACS - Leiden University

[2]UvA - University of Amsterdam

February 21, 2017

Presentation overview

1. Introduction
2. Requirements and prior work
3. Interface design
4. Implementation in MGSim
5. Results
6. Demonstration
7. Conclusion

Introduction

- The MGSim simulator
  - Configurable and extensible
  - Used for research
  - Used for education
    - Simple infrastructure
    - No direct interaction with a running simulation
    - Virtual graphical output interface
- The idea
  - Providing an interface to access external devices
    - Joystick/controller, Mice, Touch devices
  - Allows students to create interactive programs
  - Teach students about memory mapped I/O
  - Unpredictable source of I/O data

Requirements

- Provides access to features of SDL 2.0
- Should resemble actual hardware
- Component implementation
- Documentation/examples
- Minimise input latency
- Deterministic record/replay

Existing frameworks

- Simple DirectMedia Layer (SDL)
- DirectInput
- XInput
- Linux input devices
- X Input Device Extension Library (Xinput)
- Kivy

Framework feature overview

Framework	Joystick
SDL	Many
DirectInput	Many
XInput (Microsoft)	4
Linux kernel API	Many
XInput (X11)	No
Kivy	Many

SDL features cross platform access to:

- As many joysticks as the platform allows

Framework feature overview

Framework	Joystick	Keyboard/mouse
SDL	Many	Unified
DirectInput	Many	Individual
XInput (Microsoft)	4	No
Linux kernel API	Many	Individual
XInput (X11)	No	Unified
Kivy	Many	No

SDL features cross platform access to:

- As many joysticks as the platform allows
- Unified keyboard and mouse

Framework feature overview

Framework	Joystick	Keyboard/mouse	Touch
SDL	Many	Unified	Multi
DirectInput	Many	Individual	No
XInput (Microsoft)	4	No	No
Linux kernel API	Many	Individual	Multi
XInput (X11)	No	Unified	Multi
Kivy	Many	No	Multi

SDL features cross platform access to:

- As many joysticks as the platform allows
- Unified keyboard and mouse
- Multi touch and gestures

Framework feature overview

Framework	Joystick	Keyboard/mouse	Touch	Events
SDL	Many	Unified	Multi	Unified
DirectInput	Many	Individual	No	Per device
XInput (Microsoft)	4	No	No	No
Linux kernel API	Many	Individual	Multi	Per device
XInput (X11)	No	Unified	Multi	Unified
Kivy	Many	No	Multi	Per widget

SDL features cross platform access to:

- As many joysticks as the platform allows
- Unified keyboard and mouse
- Multi touch and gestures
- A unified event queue

Framework feature overview

Framework	Joystick	Keyboard/mouse	Touch	Events	State access
SDL	Many	Unified	Multi	Unified	Yes
DirectInput	Many	Individual	No	Per device	Yes
XInput (Microsoft)	4	No	No	No	Yes
Linux kernel API	Many	Individual	Multi	Per device	Yes
XInput (X11)	No	Unified	Multi	Unified	Yes
Kivy	Many	No	Multi	Per widget	No

SDL features cross platform access to:

- As many joysticks as the platform allows
- Unified keyboard and mouse
- Multi touch and gestures
- A unified event queue
- Direct access to device state

Design overview

- Uses packet based MMIO network
  - Own address space
  - Read/write requests
  - Supports sending interrupts
- Design is a mock-up
- Address space is divided into sections
  - Based on bits of the address
  - Major sections on bit 11+
  - First section subdivides on bit 10
  - First subdivision subdivides on bit 9

Control section

- Only allows 8-bit operations
- About the interface
- Only writeable section
- Controls features
- Controls event queue

Address	Reading	Writing
0	Device type	Enables or disables device
1	Events activated	Enables or disables events
2	Interrupts activated	Enables or disables interrupts
3	Interrupt channel	Sets the interrupt channel
4	Event queue size	Pops the event queue

Device information section

- Only allows 32-bit read operations
- Describes device layout
- Each entry corresponds to a state access section
- Every value contains 4 8-bit values
  - Amount of items in that section
  - Access width for the section
  - Amount of bits per value
  - Amount of values per item

Bits	25-32	17-24	9-16	0-8
Value	6	2	16	1
Meaning	6 axes	16-bit	16 bits	1 per axis

Table: Example for Xbox 360 controller axes

Event access

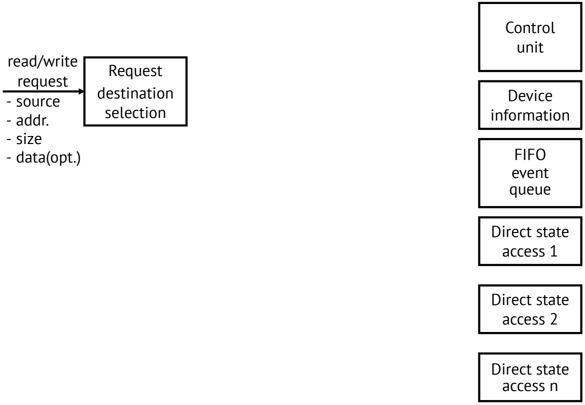
- Only allows 32-bit read operations
- Access the front of the FIFO queue
- Implementation defined events
- Selective chunk copying

Direct state access

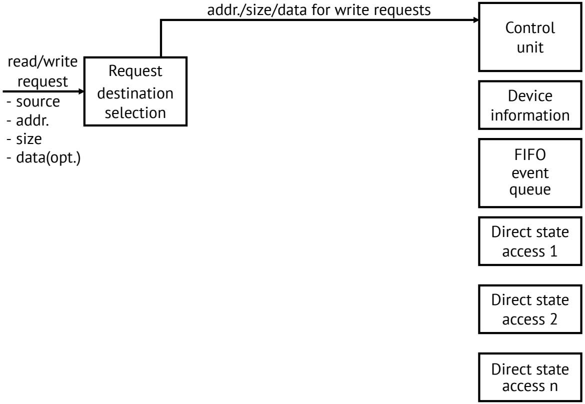
- Rest of the sections
- Access width is variable
- Direct access to state of parts
- Implementation defined

Architecture model

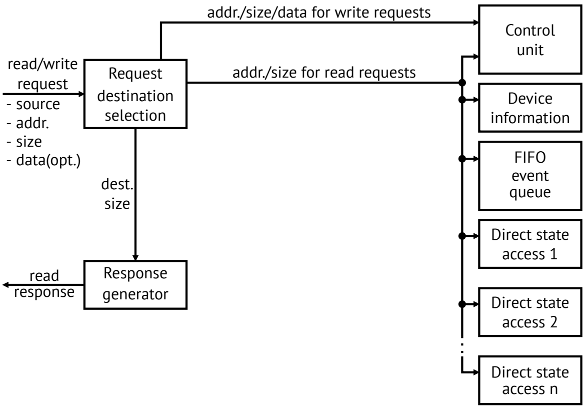
Architecture model



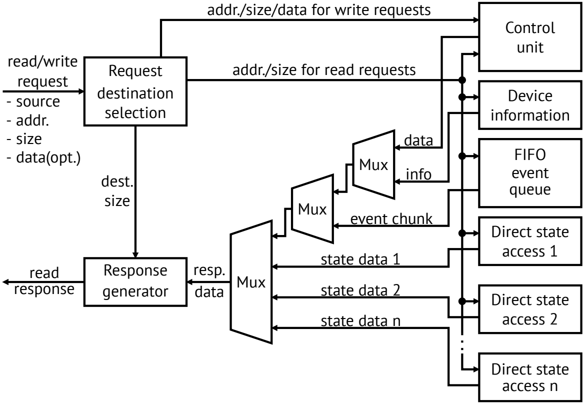
Architecture model



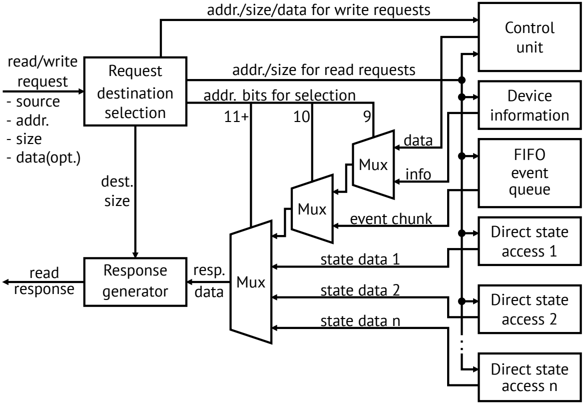
Architecture model



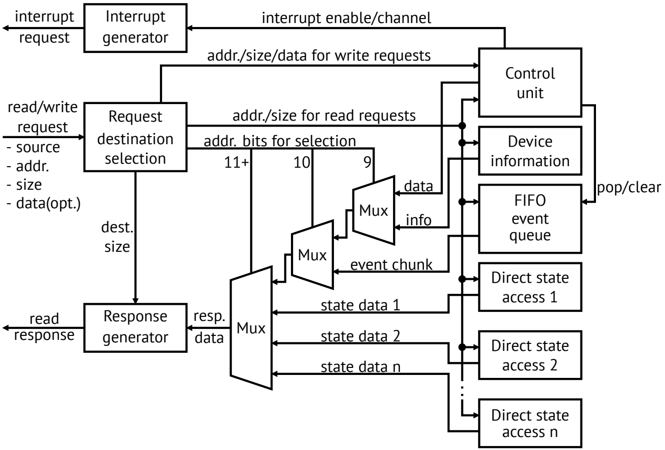
Architecture model



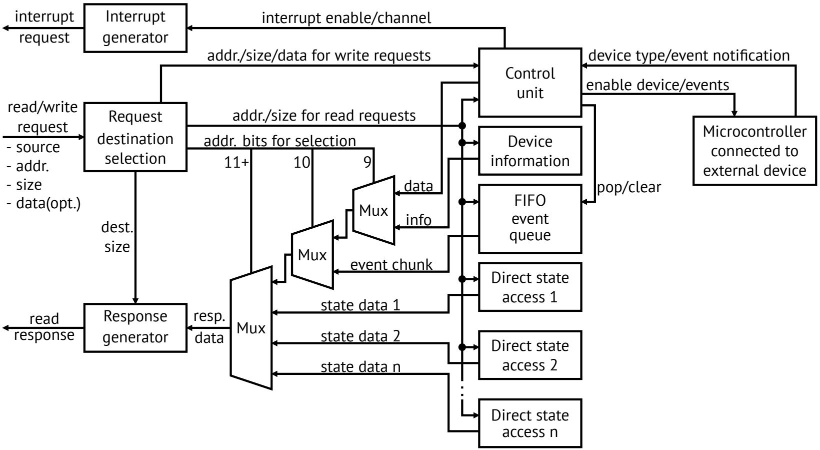
Architecture model



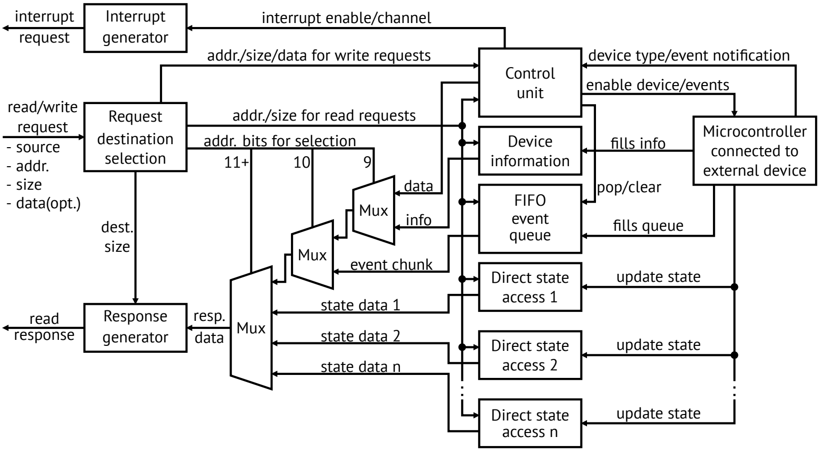
Architecture model



Architecture model



Architecture model



Design decisions

- Access width
- Event queue popping
- Extensibility

Implementation overview

- Path to our final implementation
  - Proof of concept using UART
  - Connecting MGSim to external devices
  - Updating the UART
  - Creating a component that implements our interface

Universal asynchronous receiver/transmitter (UART)

- Allows systems to communicate serially
- No elaborate synchronisation
- Transmits packets of individual bits
- Writing transmits, reading receives
- Usually have FIFO to prevent data loss
- Common on microcontrollers

Component modifications

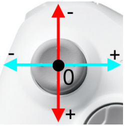
- UART component features
  - Supports reading from file descriptors
  - FIFO for both transmitting and receiving
- Connecting it to a joystick
  - Use the Linux Joystick API
  - Event byte queue is emptied into FIFO
- Capabilities and limitations
  - Transmits simple joystick events
  - Only works on Linux

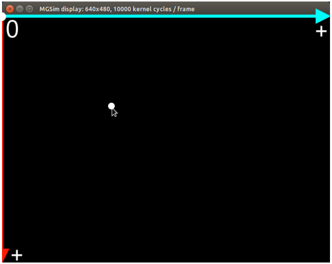
SDLInputManager

- Heavily modified DisplayManager
- Support for joysticks, mouse, touch devices
- Client based model
    - A client implements an interface
    - Clients can register for a device
    - Events are dispatched to clients
    - Access to device layout information
    - Access to joystick/mouse state
- State data types based on joystick
- Custom event structures

Joystick data type: Axes

- Signed 16-bit values
- Represents an absolute position
- Used for clearly bounded sources
  - Sliders and triggers
  - Joy- and analogue sticks
  - Mouse pointer position





Joystick data type: Buttons

- Single bits in a byte
- Binary state, pressed or released
- Used for joystick and mouse buttons

Joystick data type: Hats

- Lower 4 bits of a byte
- A bit for every main direction
- Used for directional pads

Joystick data type: Balls

- Two signed 16-bit values
- Relative movement on 2 axes
- Used for trackballs and mouse movement

Event structure

- SDL event structures were not optimal
- A new structure for each device type
  - Optimised for our case
  - Converted from SDL events
  - Better than one structure for all
- Touch events
  - SDL uses floating point values
  - We convert them to fixed point

SDLInputManager implementation

- Client management
  - Only one mouse and one touch client
  - Only one client per joystick
- Event loop
  - Configurable checking frequency
  - Events are converted and dispatched
- Device layout information
  - Amount of data sources of every type
- State updates
  - Only available for joystick/mouse
  - Filled using SDL function calls

Modifying the UART for the new system

- Added a new mode
- Events are queued similarly
- Slower, 10 v.s. 8 byte events
- Supports more features
- Platform independent

The JoyInput component

- Implements our interface design
- Configurable to access joystick, mouse, or touch devices
- Uses all features of the manager
- Allows recording and replaying sessions

SDLInputManager interaction

- Registration happens on interface activation
- Local device info and state are updated
- Events it receives are
  - used to keep the state up to date
  - added to the queue when appropriate

Request handling

- Write requests
  - Easy to validate
  - Handled with switch statement
  - Handles component state changes
- Read requests
  - Handled with nested switch statements
  - Requests are validated per section
  - Section details
    - Control section is straightforward
    - Device information is mostly static
    - Event access uses a pointer
    - Direct state access converts address into index
  - Ensures correct response endianness

Replay functionality

- Saved and replayed at request level
- Stored in plain text file
- Requests are verified on playback
- Does not stall system
- Interrupts are not supported

Performance comparison

- Comparing UART and JoyInput
- Testing joystick event copying speed

```
uint8_t buff[n];       //n = event size
if (uart[5] & 1){      //Timing starts on uart[5] receive
  buff[0] = uart[0];   //Copy a byte from the UART
  ...
  buff[n-1] = uart[0];//Timing ends on receive
}
```

```
uint32_t buff[3];
if (joydev[4]){            //Timing starts on joydev[4] receive
  buff[0] = joydevev[0];
  buff[1] = joydevev[1]; //only copied chunk for partial events
  buff[2] = joydevev[2]; //Timing ends on receive
  joydev[4] = 1;         //Pop the event queue
}
```

Comparison results

Configuration	Original cycles
JoyInput (partial event)	22
JoyInput (full event)	24
UART using joystick API	49
UART using SDL	44

- Initial results based on response
    - JoyInput performs as expected
    - UART requires investigation
- All tests store some data to stack

Comparison results

Configuration	Original cycles	Corrected cycles
JoyInput (partial event)	22	36
JoyInput (full event)	24	39
UART using joystick API	49	51
UART using SDL	44	64

- Initial results based on response
    - JoyInput performs as expected
    - UART requires investigation
- All tests store some data to stack
- Corrected results for last store
    - Results match expectations

Other tests

- Difference in latency between APIs
  - Both APIs connected to same joystick
  - No difference
- Input latency
  - Based on perception not measurement
  - Should not be a problem

Live demonstration

A showcase of some example programs.

What works

- Can connect to external devices
- Provides device information
- Provides event system
- Provides state access
- Replay can be saved and replayed

Future work

- Design and component
  - Stall processor during replay
  - Alternative replay type
  - Variable width for direct state access
- External device interaction
  - Touch input handling improvements
  - Adding relative mouse mode
  - Handling device dis- and reconnection
  - Support force feedback
  - Support SDL event generation with no active window

Questions?

Contact: Koen Putman <koen@putman.pw>

Thesis/slides available on http://putman.pw/

Code available on GitHub:
MGSim branch: https://github.com/Fleppensteyn/mgsim
Examples: https://github.com/Fleppensteyn/joyinput-examples