SysAlloc:
A Hardware Manager for Dynamic Memory Allocation in Heterogeneous Systems

Zeping Xue, David Thomas
zeping.xue10@imperial.ac.uk

FPL 2015, London
4 Sept 2015
Single Processor System
int *ptr;
ptr = malloc(size);
1. Search for suitable memory block
2. Record the memory allocation

```c
int *ptr;
ptr = malloc(size);
```
Multi-Processors (Homogeneous) System

Memory Mapped Bus

Allocator  Allocator  Allocator

Memory
What about Heterogeneous Systems?

![Diagram showing a Memory Mapped Bus connecting Nios II, MicroBlaze, FPGA Accelerator 1, FPGA Accelerator 2, Allocator, and Memory.](image)
What about Heterogeneous Systems?

FPL 2015
Fleming et al.
“PushPush”

Memory Mapped Bus

FPGA Accelerator 1

FPGA Accelerator 2

Allocator

Memory
Proposed Solution: Hardware Allocator

Memory Mapped Bus

Nios II  MicroBlaze  ARM  Memory

FPGA Accelerator 1  FPGA Accelerator 2  SysAlloc
Having memory management in heterogeneous systems...

- Improve Memory Efficiency
- Reduce Design Effort
- Make HLS More Capable
Related Work

1. Managing Shared Distributed BRAMs in NoCs [1][2][3]

2. Crossbar connected MPSoC with shared BRAMs [4]

3. HLS DMMs [5][6]

[6] Diamantopoulos et al. 2015
Related Work

1. Managing Shared Distributed BRAMs in NoCs [1][2][3]

2. Crossbar connected MPSoC with shared BRAMs [4]

3. HLS DMMs [5][6]

<table>
<thead>
<tr>
<th>Connection</th>
<th>SysAlloc</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>NoC</td>
<td>NoC</td>
<td>NoC</td>
<td>Crossbar</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory</th>
<th>On/Off-Chip</th>
<th>Shared Distributed On-Chip BRAMs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Scalability</th>
<th>Unbounded</th>
<th>Platform-Dependent</th>
<th>Limited by resource</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Clock Rate</th>
<th>150MHz</th>
<th>Platform-Dependent</th>
<th>140MHz</th>
</tr>
</thead>
</table>

[6] Diamantopoulos et al. 2015
Our Proposed Hardware Allocator

Scalable
- Functionality-wise, having arbitrary number of clients

Flexible
- Any range of memory can be managed

Efficient
- Clock Rate
- Resource Utilisation

SysAlloc
Route Map

- SysAlloc:
  - System Integration
  - Allocator Algorithm
- Evaluation Results
- Conclusion
SysAlloc System Integration
SysAlloc System Integration

Client 0
M O

Client 1
M ●

Client n
M O

Memory Mapped Bus

M Master Interface
S Slave Interface

● Token with value 1
O Token with value 0

read token

found token

write request

read status

status is done

read result
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
Allocator Access Examples

Software (ARM core) Access

```c
int* SysMalloc(int size){
    int token, status, *pointer;
    token = 0;
    status = 0;

    while(token == 0){
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while(status == 0){
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

Memory Map

<table>
<thead>
<tr>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000004</td>
</tr>
<tr>
<td>......</td>
</tr>
<tr>
<td>SysA_TOKEN</td>
</tr>
<tr>
<td>SysA_CMD</td>
</tr>
<tr>
<td>SysA_STAT</td>
</tr>
<tr>
<td>SysA_RESULT</td>
</tr>
</tbody>
</table>

18
int* SysMalloc(int size) {

    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
Allocator Access Examples

Software (ARM core) Access

```c
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td>0x00000008</td>
<td>0</td>
</tr>
<tr>
<td>0x0000000C</td>
<td>size</td>
</tr>
<tr>
<td>0x00000010</td>
<td>0</td>
</tr>
</tbody>
</table>

20
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}

Memory Map

<table>
<thead>
<tr>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000004</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>SysA_TOKEN</td>
</tr>
<tr>
<td>SysA_CMD</td>
</tr>
<tr>
<td>SysA_STAT</td>
</tr>
<tr>
<td>SysA_RESULT</td>
</tr>
</tbody>
</table>
int* SysMalloc(int size) {

    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}

---

**Memory Map**

<table>
<thead>
<tr>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000004</td>
</tr>
<tr>
<td>0x00000008</td>
</tr>
</tbody>
</table>

**Storage**

<table>
<thead>
<tr>
<th>SysA_TOKEN</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysA_CMD</td>
<td>size</td>
</tr>
<tr>
<td>SysA_STAT</td>
<td>0</td>
</tr>
<tr>
<td>SysA_RESULT</td>
<td>0</td>
</tr>
</tbody>
</table>
Allocator Access Examples

Software (ARM core) Access

```c
int* SysMalloc(int size){

    int token, status, *pointer;
    token = 0;
    status = 0;

    while(token == 0){
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while(status == 0){
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

Memory Map

<table>
<thead>
<tr>
<th></th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>SysA_TOKEN</td>
<td>0</td>
</tr>
<tr>
<td>SysA_CMD</td>
<td>size</td>
</tr>
<tr>
<td>SysA_STAT</td>
<td>0</td>
</tr>
<tr>
<td>SysA_RESULT</td>
<td>0</td>
</tr>
</tbody>
</table>

23
## Allocator Access Examples

### Software (ARM core) Access

```c
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

### Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>SysA_TOKEN</td>
<td>0</td>
</tr>
<tr>
<td>SysA_CMD</td>
<td>size</td>
</tr>
<tr>
<td>SysA_STAT</td>
<td>1</td>
</tr>
<tr>
<td>SysA_RESULT</td>
<td>0x0000A100</td>
</tr>
</tbody>
</table>

---

24
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
int* SysMalloc(int size) {

    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```c
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

### Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td></td>
</tr>
<tr>
<td>0x00000004</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x0000A100</td>
<td>SysA_TOKEN</td>
</tr>
<tr>
<td>1</td>
<td>SysA_CMD</td>
</tr>
<tr>
<td>0</td>
<td>SysA_STAT</td>
</tr>
<tr>
<td>0x0000A100</td>
<td>SysA_RESULT</td>
</tr>
</tbody>
</table>
Allocator Access Examples

Software (ARM core) Access

```c
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while(token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while(status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

Memory Map

<table>
<thead>
<tr>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
</tr>
<tr>
<td>0x00000004</td>
</tr>
<tr>
<td>.......</td>
</tr>
<tr>
<td>SysA_TOKEN</td>
</tr>
<tr>
<td>SysA_CMD</td>
</tr>
<tr>
<td>SysA_STAT</td>
</tr>
<tr>
<td>SysA_RESULT</td>
</tr>
</tbody>
</table>
Software (ARM core) Access

```c
int* SysMalloc(int size) {
    int token, status, *pointer;
    token = 0;
    status = 0;

    while (token == 0) {
        token = AXI_READ(SysA_TOKEN);
    }

    AXI_WRITE(SysA_CMD, size);

    while (status == 0) {
        status = AXI_READ(SysA_STAT);
    }

    pointer = AXI_READ(SysA_RESULT);

    return pointer;
}
```

HLS Access (Vivado HLS)

```c
int SysMalloc(int size, int *PortM) {
    int token, status, result;
    token = 0;
    status = 0;

    while (token == 0) {
        token = PortM[SysA_TOKEN];
    }

    PortM[SysA_CMD] = size;

    while (status == 0) {
        status = PortM[SysA_STAT];
    }

    result = PortM[SysA_RESULT];

    return result + PortM;
}
```
SysAlloc Allocator Algorithm
Background of SysAlloc Allocator Algorithm

Background of SysAlloc Allocator Algorithm

Background of SysAlloc Allocator Algorithm

<table>
<thead>
<tr>
<th>level</th>
<th>size</th>
<th>availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l = 0$</td>
<td>$2^3$</td>
<td>①</td>
</tr>
<tr>
<td>$l = 1$</td>
<td>$2^2$</td>
<td>② ⊗ ③</td>
</tr>
<tr>
<td>$l = 2$</td>
<td>$2^1$</td>
<td>④ ⊗ ⑤ ⊗ ⑥ ⊗ ⑦</td>
</tr>
<tr>
<td>$l = 3$</td>
<td>$2^0$</td>
<td>$b_0 \cdot b_1 \cdot b_2 \cdot b_3 \cdot b_4 \cdot b_5 \cdot b_6 \cdot b_7$</td>
</tr>
</tbody>
</table>

①: two-input or-gate for detecting contiguous 0s.

Background of SysAlloc Allocator Algorithm

Limitation!

For large bit-map, high resource utilisation and low clock rate due to long sequence of logic gates.
SysAlloc Allocator Algorithm

Our Proposed Solution
Allocation Tree

Allocation Search Binary Tree

<table>
<thead>
<tr>
<th>Node Value</th>
<th>Node Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Empty</td>
</tr>
<tr>
<td>10</td>
<td>Partially Full</td>
</tr>
<tr>
<td>11</td>
<td>Full</td>
</tr>
</tbody>
</table>
Allocation Tree Storage

Stored in memory as groups of nodes

Allocation Search Binary Tree

Bit-Map
Node Value | Representation | Node Status
---|---|---
00 | | Empty
10 | | Partially Full
11 | | Full

Bit-Map

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Allocation Tree Storage

Request size = 3 blocks

Bit-Map:
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Request size = 3 blocks
Constant FPGA Logic Utilisation
For large Bit-Maps,

\[
\text{Max Tree Depth} \quad \ll \ll \quad \log_2(\text{Bit-Map Size}) \ll \ll \quad \text{Bit-Map Size}
\]

Using the search tree is more efficient than directly searching in bit-map.
Evaluation Results
## Evaluation Set-up

<table>
<thead>
<tr>
<th>Environment</th>
<th>Zynq-7000 SoC XC7Z020 FPGA @ 150MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysAlloc Configuration</td>
<td>Manage 128MB DDR Minimum Block Size = 16B</td>
</tr>
<tr>
<td>Synthetic Clients</td>
<td>Request rate and size are configurable</td>
</tr>
</tbody>
</table>
Scalability – Pathological Case

The diagram illustrates the scalability of different allocation methods under varying numbers of active clients. The x-axis represents the number of active clients, while the y-axis shows the number of allocations per second. The legend indicates the different methods:

- SysAlloc(Client)
- SysAlloc(System)
- Ideal(Client)
- Ideal(System)
- Software(Client)
- Software(System)

The curves show how the performance degrades as the number of active clients increases. Ideal methods maintain performance better than others, especially as the number of clients grows.
Scalability – Size & Wait Period with Exponential Growth

1.36 million malloc/s
## Configuration Cases

<table>
<thead>
<tr>
<th>Case A</th>
<th>Managing 512 MB DDR with minimum block in size of 16B.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case B</td>
<td>A linked-list using 256KB on-chip memory with minimum block in size of 8B.</td>
</tr>
<tr>
<td>Case C</td>
<td>Signal processing applications need 256 1MB buffers to be managed.</td>
</tr>
</tbody>
</table>
Configuration Cases – Bit-Map Size

- **Case A**: 32M
- **Case B**: 32K
- **Case C**: 256
Configuration Cases – FPGA LEs Utilisation

- Case A: LUTs Utilisation 9.6%, Registers Utilisation 4.7%
- Case B: LUTs Utilisation 9.1%, Registers Utilisation 4.2%
- Case C: LUTs Utilisation 8.4%, Registers Utilisation 3.7%
Configuration Cases – Memory Bits Required

- **Case A**: 13.1MB
- **Case B**: 18.31KB
- **Case C**: 0.285KB
Configuration Cases – Latency

Allocation Latency with Single Client (Cycles)

<table>
<thead>
<tr>
<th>Case</th>
<th>A/DDR</th>
<th>B/DDR</th>
<th>B/BRAM</th>
<th>C/DDR</th>
<th>C/BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>549</td>
<td>363</td>
<td>122</td>
<td>271</td>
<td>125</td>
</tr>
</tbody>
</table>
Conclusion
Conclusion

Scalarable
Efficient
Flexible

SysAlloc

Peak system allocation rate = 1.36 million malloc/s, when managing 128MB DDR for 4 Clients
Future Work

Faster Allocator
Future Work

More Efficient Communication Protocol
Future Work

Garbage Collection
Future Work

HLS Access
Thank You 😊 & Questions ?

Source code can be found at https://github.com/hilx/SysAlloc