Implementing A Read/Write Memory Module

Inputs:
• 2 Address bits: A0 and A1
• 1 “chip select” (CS) bit
• 1 read/write bit (1 = read; 0 = write)
• 1 clock signal (CLK)

Input or Output:
• Data bit (connected to the “data bus”)
Implementing A Read/Write Memory Module

With 2 address bits, how many memory elements can we address?

How could we implement each memory element?
Implementing A Read/Write Memory Module

With 2 address bits, how many memory elements can we address?

• 4 1-bit elements

How could we implement each memory element?

• With a D flip-flop
Memory Module Specification

When chip select is low:
• No memory elements change state
• The memory does not drive the data bus
Memory Module Specification

When chip select is high:

• If R/W is high:
  – Drive the data bus with the value that is stored in the element specified by A1, A0

• If R/W is low:
  – Store the value that is on the data bus in the element specified by A1, A0
Memory Timing Diagram

Q2
A1
A0
R/W
CS
CLK
D
Memory Timing Diagram

Q2
A1
A0
R/W
CS
CLK

Data bus not driven

D
Memory Timing Diagram

Memory element 2 is initially in a high state.

Q2
A1
A0
R/W
CS
CLK
D

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch
Memory Timing Diagram

What happens next?
Memory Timing Diagram

Q2
A1
A0
R/W
CS
CLK
D

Chip is selected
Memory Timing Diagram

Q2

A1

A0

R/W

CS

CLK

D

Address memory element 2
Memory Timing Diagram

Specify a write operation

Data bus is driven low (by another device)
Memory Timing Diagram

Q2__________
A1
A0__________
R/W_________
CS
CLK

Clock goes low
Memory Timing Diagram

Memory element 2 changes state to low
**Memory Timing Diagram**

**Setup time**: all inputs must be valid during this time

- Q2
- A1
- A0
- R/W
- CS
- CLK
- D
Memory Timing Diagram

**Hold time**: all inputs must continue to be valid
Memory Timing Diagram II

Q2   ____________
    
A1   ______________
    
A0   ______________
    
R/W  ______________
    
CS   ______________
    
CLK  
    
D    ______________
Memory Timing Diagram II

Q2

- __________

A1

- 

A0

- __________

R/W

- 

CS

- __________

CLK

D

- __________

Data bus is not driven
Memory Timing Diagram II

What happens next?
Memory Timing Diagram II

On chip select – drive data bus from Q2
What happens now?
Memory Timing Diagram II

Data bus returns to a non-driven state
Next Time

• Memory implementation
• Microprocessor architecture
  – Processor components
  – Timing and control
• The Atmel microprocessor

• Reading: make sure you have read 3.1-3.2
Last Time

• Arithmetic
  – Inversion
  – Addition
  – Subtraction
  – Multiplication

• Memory chip specification
Today

• Sequential counter design
• Memory design
• Processor components
Administrivia

• Homework 1:
  – All but question 7 due today at 5:00
  – Question 7 is due Friday at 5:00

• Homework 2:
  – Available on the website by the end of the day
Counter Design

What is the fundamental problem with our ripple counter?
Counter Design

What is the fundamental problem with our ripple counter?

• The state of the different output lines does not change at the same time
  – There is a small delay as the carry moves from one flip-flop to the next
Synchronous Counter Design

We would like to change things such that all of the outputs change on the downward edge of the clock

• What is the fundamental insight in our fix?
Synchronous Counter Design

What is the fundamental insight in our fix?

• We want the clock to be delivered to all of the flip-flops (not just the first flip-flop)

Key: we will gate the clock signal arriving to each of the flip-flops
Gating the Clock

Under what conditions do we want X1 to change state?
– (X1 is the 2\textsuperscript{nd} bit)
Gating the Clock

Under what conditions do we want X1 to change state?

• When X0 is high
• (and when the clock transitions from high to low)
Gating the Clock

Under what conditions do we want X2 to change state?
Gating the Clock

Under what conditions do we want X2 to change state?

• When both X0 and X1 are high
• (and when the clock transitions from high to low)
Gating the Clock

What do these observations say about our design?
Last Time(s)

- Memory specification
- Synchronous counter design
Today

• Memory module design
• Microprocessor design
Administrivia

• Project 1 due Tuesday
  – Demo must be done by class time
  – Group report: due @5:00. Only one copy required. Will accept hardcopy, M$Word, etc
  – Personal report: due @5:00. Hand-in using blackboard. Raw text only!

• Homework 1: solutions

• Homework 2: is available from the web site
Synchronous Counter Design

CLK0 always follows CLK
Synchronous Counter Design

CLK1 follows CLK when X0 is high
Synchronous Counter Design

CLK2 follows CLK when X0 and X1 are high.
Synchronous Counter Design

AND gate introduces a delay that is equivalent to the other delays

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch
Recall: A Read/Write Memory Module

Inputs:
- 2 Address bits: A0 and A1
- 1 “chip select” (CS) bit
- 1 read/write bit (1 = read; 0 = write)
- 1 clock signal (CLK)

Input or Output:
- Data bit (connected to the “data bus”)

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch 43
Recall: Memory Module Specification

When chip select is low:

- No memory elements change state
- The memory does not drive the data bus
Recall: Memory Module Specification

When chip select is high:

• If R/W is high:
  – Drive the data bus with the value that is stored in the element specified by A1, A0

• If R/W is low:
  – Store the value that is on the data bus in the element specified by A1, A0
Memory Implementation

How do we build one?
Memory Implementation

How do we build one?

1. Focus on the behavior of a single bit of memory

2. Replicate this bit and add control/selection logic
Memory Implementation

Behavior of a single bit of memory
• Input specifies whether the memory should stay the same or copy the value from the data bus
• Copy happens on downward clock edge

What does this circuit look like?
Memory Cell

“cell control line”

0 = No change
1 = Copy data
Memory Cell

Nothing happens until the downward side of the clock

0 = No change
1 = Copy data
Memory Cell

What happens?

1->0

0=No change
1=Copy data

ANDREW H. Fagg: Embedded Real-Time Systems: Computer Arch
Memory Cell

D receives the input from Q: so no change

0 = No change
1 = Copy data

D \rightarrow Q

1 \rightarrow 0
Memory Cell

How about now?

1->0

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch
Memory Cell

D receives the input from the data bus

$1 \rightarrow 0$

0=No change
1=Copy data
Memory Implementation

Now: consider a memory will 4 cells

- For a specific cell – how do we decide to change its state?
Memory Implementation

For a specific cell – how do we decide to change its state?

- The address must match,
- It must be a write operation,
- And the chip must be selected
Memory Implementation

Assume for the instant that the address matches:

- What is the truth table for the function from R/W and CS to “cell control line” (CCL)?

<table>
<thead>
<tr>
<th>R/W</th>
<th>CS</th>
<th>CCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Memory Implementation

Assume for the instant that the address matches:

- What is the truth table for the function from R/W and CS to “cell control line” (CCL)?

<table>
<thead>
<tr>
<th>R/W</th>
<th>CS</th>
<th>CCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Memory Implementation

Assume for the instant that the address matches:

- What is the truth table for the function from R/W and CS to “cell control line” (CCL)?

<table>
<thead>
<tr>
<th>R/W</th>
<th>CS</th>
<th>CCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Memory Implementation

How can we address the memory cell (simply)?
Memory Implementation

How can we address the memory cell (simply)?
• Use a demultiplexer!
Memory Implementation

How can we address the memory cell (simply)?

• Use a demultiplexer!

What does the complete circuit look like?
Last Time

• Synchronous counter
• Memory design
Today

• More on memory:
  – Implementing our memory “read” operation
  – Expanding memory
• Microcontroller design
Administrivia

- Lab 1 due Tuesday:
  - Demos in class
  - Reports by 5:00
- Homework 1:
  - Available late today
- Homework 2:
  - Due in 1 week
Memory Implementation

If address does not match, then CCL is 0
• This means no change to cell state!
Memory Implementation

On to the “reading” side of the memory…

• Under what conditions will a particular memory cell drive the data bus?
Memory Implementation

Under what conditions will a particular memory cell drive the data bus?

• The cell must be addressed
• It must be a “read” operation
• The chip must be selected
Memory Implementation

Assume for the instant that the address matches:

- What is the truth table for the function from R/W and CS to “drive bus” (DB)?

<table>
<thead>
<tr>
<th>R/W</th>
<th>CS</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Memory Implementation

Assume for the instant that the address matches:

• What is the truth table for the function from R/W and CS to “drive bus” (DB)?

<table>
<thead>
<tr>
<th>R/W</th>
<th>CS</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Memory Implementation

Assume for the instant that the address matches:

- What is the truth table for the function from R/W and CS to “drive bus” (DB)?

<table>
<thead>
<tr>
<th>R/W</th>
<th>CS</th>
<th>DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Memory Implementation

How do we make use of this “drive bus” signal?
Memory Implementation

How do we make use of this “drive bus” signal?

• As the “select” input of a tristate buffer
Memory Implementation

Finally: how do we select which memory cell will write to the bus?
Memory Implementation

Finally: how do we select which memory cell will write to the bus?

• Use a multiplexer!

How do all of the pieces fit together?
Memory Implementation

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch 76
Memory Implementation

When:
• Address match,
• CS is high, and
• R/W is high
Moving Beyond the Bit

In modern processors, the data bus is at least 8 lines wide. How do we handle this in our implementation?
Moving Beyond the Bit

How do we handle this in our implementation?

• For each address – there will be 8 flip-flops (one for each line)
• These are independent from each other – but they do share CS, R/W, and address circuitry
Our Memory Chip

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch
A Memory Abstraction

• We think of memory as an array of elements
• Each element contains a value
• It is most common for the values to be 8-bits wide (so a byte)

| 0x32 | 0xF1 | 0x11 | 0x67 | ...... | 0x7B |
| 0    | 1    | 2    | 3    | ...... |      |

\[2^M - 1\]
Memory Expansion

Given two memory chips with M address bits and N data bits:

• How do we construct a memory system that is twice the size of the two chips?
Memory Expansion

Start with two memories:

<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>......</td>
<td>$2^{M-1}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>......</td>
<td>$2^{M-1}$</td>
</tr>
</tbody>
</table>
Memory Expansion

Start with two memories:

```
<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2^{M-1}</td>
<td></td>
</tr>
</tbody>
</table>
```

And append them together:

```
<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2^{M-1}</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2^{M}</td>
<td>2^{M+1}</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2^{M+2}</td>
<td>2^{M+3}</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>0x32</th>
<th>0xF1</th>
<th>0x11</th>
<th>0x67</th>
<th>......</th>
<th>0x7B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2^{M+1}</td>
<td>2^{M+1}</td>
</tr>
</tbody>
</table>
```
Memory Expansion

How do we build one?

• Note that we have one extra address bit
Chip select for the composite memory
“Low order” address bits are used in the same way as before.
The “high order” bit selects which chip we will use.
Only one chip ever drives the data bus at one time.
Memory Implementation

Final Notes:

- We can construct large memories by cascading demultiplexers
- But: at every level, we induce an additional delay in the response of the memory unit
- This delay is critical in our computers
Types of Memory

Our design: Random Access Memory (RAM)

• Computer can change state of this memory at any time
• Once power is lost, we lose the contents of the memory
• This will be our data storage on our microcontrollers
Types of Memory

• Read Only Memory (ROM)
  – Computer \textit{cannot} arbitrarily change state of this memory
  – When power is lost, the contents are maintained
Types of Memory

Erasable/Programmable ROM (EPROM)

• State can be changed under very specific conditions (usually not when connected to a computer)

• Our microcontrollers have an Electrically Erasable/Programmable ROM (EEPROM) for program storage
Components of a Computer

• Microprocessor
• Memory
  – Both RAM and ROM
• Input/Output devices
• Clock
Components of a Microprocessor

• Registers (fast-access memory)
  – General purpose: used for data storage
  – Special purpose: used to control the behavior of the microprocessor and/or the devices connected to it

• Instruction decoder
  – Instructions are the primitive “actions” that the microprocessor can perform
  – Load/store to/from memory, AND, ADD, jump…
Components of a Microprocessor

- Arithmetic Logical Unit (ALU)
- Memory control logic
- Timers
  - Including timing mechanisms for instruction fetch and execution
- Interrupt processor
Common Special-Purpose Registers

- **Program counter (PC):** address of memory from which the next instruction will be fetched
- **Status register (SR):** stores basic state of the processor
- **Instruction register:** stores the recently fetched instruction
Common Special-Purpose Registers

• Stack pointer (SP): address in memory of the top of “the stack”
  – On a subroutine call, will store the return address and relevant data
  – When the subroutine “returns” (ends), the PC address is read back from the stack
An Example: the Atmel Mega8

Andrew H. Fagg: Embedded Real-Time Systems: Computer Arch
Atmel Mega8

8-bit data bus

- Primary mechanism for data exchange
Atmel Mega8

32 general purpose registers

- 8 bits wide
- 3 pairs of registers can be combined to give us 16 bit registers
Atmel Mega8

Special purpose registers

• Control of the internals of the processor
Atmel Mega8

Random Access Memory (RAM)
• 1 KByte in size
Atmel Mega8

Random Access Memory (RAM)
• 1 KByte in size

Note: in high-end processors, RAM is a separate component
Atmel Mega8

Flash (EEPROM)
- Program storage
- 8 KByte in size
Atmel Mega8

Flash (EEPROM)
- In this and many microcontrollers, program and data storage is separate
- Not the case in our general purpose computers
Atmel Mega8

EEPROM

- Permanent data storage
Arithmetic Logical Unit

- Data inputs from registers
- Control inputs not shown (derived from instruction decoder)
Machine-Level Programs

Machine-level programs are stored as sequences of machine instructions
• Stored in program memory
• Execution is generally sequential (instructions are executed in order)
• But – with occasional “jumps” to other locations in memory
Types of Instructions

• Memory operations: transfer data values between memory and the internal registers
• Mathematical operations: ADD, SUBTRACT, MULT, AND, etc.
• Tests: value == 0, value > 0, etc.
• Program flow: jump to a new location, jump conditionally (e.g., if the last test was true)
Atmel Mega8: Decoding Instructions

Program counter

- Address of currently executing instruction
Atmel Mega8: Decoding Instructions

Instruction register
- Stores the machine-level instruction currently being executed
Atmel Mega8

Instruction decoder

• Translates current instruction into control signals for the rest of the processor
Atmel Mega8

Status register
• Many machine instructions affect the state of this register
Mega8 Status Register

Interrupt enable

• If ‘1’, the currently executing program can be interrupted by another event (e.g., a byte arriving through the serial port)
Mega8 Status Register

Half carry flag
• Set if an arithmetic operation resulted in a carry from the first nybble to the next
Mega8 Status Register

Two’s complement overflow flag
- Set if an arithmetic operation resulted in an overflow in two’s complement (e.g., incrementing an 8-bit number whose value is 127)
Mega8 Status Register

Negative flag
- Set if an arithmetic operation resulted in a negative value
Mega8 Status Register

Zero flag

- Set if an arithmetic operation resulted in a value of zero
Mega8 Status Register

- **Carry flag**
  - Set if an arithmetic operation resulted in a carry (with an unsigned value)
Some Mega8 Memory Operations

**LDS Rd, k**
- Load SRAM memory location k into register Rd
- \( \text{Rd} \leftarrow (k) \)

**STS Rd, k**
- Store value of Rd into SRAM location k
- \( (k) \leftarrow \text{Rd} \)

We refer to this as “Assembly Language”
Some Mega8 Memory Operations

**LD Rd, Ry**
- Load SRAM memory location indicated by Ry into register Rd
- Rd <- (Ry)

**ST Rd, Ry**
- Store value of Rd into SRAM location indicated by the value of Ry
- (Ry) <- Rd
Some Mega8 Arithmetic and Logical Instructions

**ADD Rd, Rr**
- Rd and Rr are registers
- Operation: Rd <- Rd + Rr
- Also affects status register (zero, carry, etc.)

**ADC Rd, Rr**
- Add with carry
- Rd <- Rd + Rr + C
Some Mega8 Arithmetic and Logical Instructions

**NEG Rd**: take the two’s complement of Rd
**AND Rd, Rr**: bit-wise AND with a register
**ANDI Rd, K**: bit-wise AND with a constant
**EOR Rd, Rr**: bit-wise XOR
**INC Rd**: increment Rd
**MUL Rd, Rr**: multiply Rd and Rr (unsigned)
**MULS Rd, Rd**: multiply (signed)
Some Mega8 Test Instructions

**CP Rd, Rr**
- Compare Rd with Rr
- Alters the status register

**TST Rd**
- Test for zero or minus
- Alters the status register
Some Program Flow Instructions

**RJMP** k

- Change the program counter by k+1
- PC <- PC + k + 1

**BRCS** k

- Branch if carry set
- If C==1 then PC <- PC + k + 1
Connecting Assembly Language to C

- Our C compiler is responsible for translating our code into Assembly Language.
- Today, we rarely program in Assembly Language.
  - Embedded systems are a common exception.
  - Also: it is useful in some cases to view the assembly code generated by the compiler.
An Example

A C code snippet:

if(A > B) {
    D += A;
}

An Example

A C code snippet:

```c
if(A > B) {
    D += A;
}
```

The Assembly:

```
LDS R1 (A)
LDS R2 (B)
CP R1, R2
BRMI 3
LDS R3 (D)
ADD R3, R1
STS (D), R3
```
An Example

A C code snippet:

```c
if(A > B) {
    D += A;
}
```

Branch if greater than or equal to will jump ahead 3 instructions if true

The Assembly:

```assembly
LDS R1 (A)
LDS R2 (B)
CP R2, R1
BRGE 3
LDS R3 (D)
ADD R3, R1
STS (D), R3
```