Module 3

Lecture: ARM Cortex M Architecture Digital Logic
Introduction to Digital Logic

You will learn in this module

- Fundamentals of digital logic
  - Digital versus analog
  - Gate-level view
    - NOT
    - AND
    - OR
    - XOR or EOR
  - Addition
  - Introduction to the processor
Digital versus analog

We use electric signals to represent information

- Analog
  - Voltage is analogous to signal
  - Minimum and maximum
  - Continuous relationship
  - Physical interaction with the real world

- Digital
  - $n$ digital signals represent an $n$-bit integer
  - Minimum and maximum
  - Discontinuous relationship, $2^n$ possible values

- Conversion between real world and computer
  - Output: Digital to Analog Converter (DAC)
  - Input: Analog to Digital Converter (ADC)
Gate-level view of digital NOT gate

<table>
<thead>
<tr>
<th>A</th>
<th>p-type</th>
<th>n-type</th>
<th>~A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0V</td>
<td>active</td>
<td>off</td>
<td>3.3V</td>
</tr>
<tr>
<td>3.3V</td>
<td>off</td>
<td>active</td>
<td>0V</td>
</tr>
</tbody>
</table>
Gate-level view of digital AND gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&amp;B</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>
Gate-level view of digital OR gate

A | B | A|B

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Addition - 1-bit Adder

\[
\begin{array}{c c c c c}
A & B & C_{in} & A+B+C_{in} & C_{out} & S_{out} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 2 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 2 & 1 & 0 \\
1 & 1 & 0 & 2 & 1 & 0 \\
1 & 1 & 1 & 3 & 1 & 1 \\
\end{array}
\]
Addition - 8-bit Adder
Introduction to Digital Logic

Summary

- Digital Logic
  - CMOS is built with NMOS and PMOS transistors
  - AND OR NOT XOR building blocks
  - Complex logic like the MSP432 is created
    - With a systems-level approach
    - By combining simpler building blocks
  - Software will use digital logic
    - To process data
    - To make decisions
You will learn in this module

- Cortex M Architecture
  - Buses
  - CISC versus RISC
  - Registers
  - Memory
  - Addressing modes
ARM Cortex M Architecture

ARM Cortex-M4 processor

- Harvard versus von Neumann architecture
- Different busses for instructions and data

- ICode bus - Fetch op codes from ROM
- System bus - Data from RAM and I/O
- Dcode bus - Debugging
- PPB bus - Private peripherals
### Reduced Instruction Set Computer (RISC)

<table>
<thead>
<tr>
<th>CISC</th>
<th>RISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many instructions</td>
<td>Few instructions</td>
</tr>
<tr>
<td>Instructions have varying lengths</td>
<td>Instructions have fixed lengths</td>
</tr>
<tr>
<td>Instructions execute in varying times</td>
<td>Instructions execute in 1 or 2 bus cycles</td>
</tr>
<tr>
<td>Many instructions can access memory</td>
<td>Few instructions can access memory</td>
</tr>
<tr>
<td>In one instruction, the processor can both</td>
<td>No one instruction can both read and write memory in the same instruction</td>
</tr>
<tr>
<td>• Read memory and</td>
<td></td>
</tr>
<tr>
<td>• Write memory</td>
<td></td>
</tr>
<tr>
<td>Fewer and more specialized registers</td>
<td>Many identical general purpose registers</td>
</tr>
<tr>
<td>• Some registers contain data</td>
<td></td>
</tr>
<tr>
<td>• Others contain addresses</td>
<td></td>
</tr>
<tr>
<td>Many different types of addressing modes</td>
<td>Limited number of addressing modes</td>
</tr>
<tr>
<td></td>
<td>• Register, PC - relative</td>
</tr>
<tr>
<td></td>
<td>• Immediate</td>
</tr>
<tr>
<td></td>
<td>• Indexed</td>
</tr>
</tbody>
</table>

#### RISC machine

- Pipelining provides single cycle operation for many instructions
- Thumb-2 configuration employs both 16 and 32-bit instructions
Registers

Where are data?
- Registers
- RAM
- ROM
- I/O ports

Where are commands?
- ROM (pointed to by PC)

<table>
<thead>
<tr>
<th>Condition Code Bits</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>negative</td>
</tr>
<tr>
<td>Z</td>
<td>zero</td>
</tr>
<tr>
<td>V</td>
<td>overflow</td>
</tr>
<tr>
<td>C</td>
<td>carry</td>
</tr>
</tbody>
</table>

Result is negative
Result is zero
Signed overflow
Unsigned overflow
Memory Map

For the detailed Memory Map go to http://www.ti.com/lit/ds/symlink/msp432p401r.pdf
## Endianness

### 16-bit 1000 = 0x03E8

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000.0850</td>
<td>0x03</td>
<td>0x2000.0850</td>
<td>0x0E8</td>
</tr>
<tr>
<td>0x2000.0851</td>
<td>0xE8</td>
<td>0x2000.0851</td>
<td>0x03</td>
</tr>
</tbody>
</table>

Big Endian | Little Endian

### 32-bit 1000000 = 0x000F4240

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000.0850</td>
<td>0x00</td>
<td>0x2000.0850</td>
<td>0x40</td>
</tr>
<tr>
<td>0x2000.0851</td>
<td>0x0F</td>
<td>0x2000.0851</td>
<td>0x42</td>
</tr>
<tr>
<td>0x2000.0852</td>
<td>0x42</td>
<td>0x2000.0852</td>
<td>0x0F</td>
</tr>
<tr>
<td>0x2000.0853</td>
<td>0x40</td>
<td>0x2000.0853</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Big Endian | Little Endian

### ASCII string “Jon” = 0x4A,0x6F,0x6E,0x00

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000.0850</td>
<td>0x4A</td>
</tr>
<tr>
<td>0x2000.0851</td>
<td>0x6F</td>
</tr>
<tr>
<td>0x2000.0852</td>
<td>0x6E</td>
</tr>
<tr>
<td>0x2000.0853</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Big Endian and Little Endian
Addressing Modes: immediate

MOV R0, #100

Register
Immediate
Indexed
PC-relative
Addressing Modes: Indexed

LDR R0, [R1]

Register
Immediate
Indexed
PC-relative

EEPROM

RAM

PC 0x00000144

0x00000142
0x00000144
0x00000146
0x00000148
0x12345678

R0
0x20000000
0x20000000
0x20000004
0x20000008
0x2000000C

R1 0x20000004

12345678

6808
Addressing Modes: Indexed

```
LDR R0, [R1, #4]
```

- **Register**: R0
- **Immediate**: #4
- **Indexed**: R1
- **PC-relative**: 

Diagram:
- PC: 0x00000146
- R0: 0x87654321
- R1: 0x20000004
- EEPROM: 0x00000142, 0x00000144, 0x00000146, 0x00000148
- RAM: 0x20000000, 0x20000004, 0x20000008, 0x2000000C
Variable Access: Load/store architecture

Increment: .asmfunc
LDR R1,CountAddr ; R1=pointer to Count
LDR R0,[R1] ; R0=value of Count
ADD R0,R0,#1
STR R0,[R1]
BX LR
.endasmfunc
CountAddr .field Count,32

uint32_t Count;
void Increment(void){
    Count++;
}
ARM Cortex M Architecture

Summary

- Architecture
  - Buses
  - Registers
  - Memory
  - Addressing modes

Terms:
- RISC vs CISC
- Little vs big endian
- Address vs data
- Variables

Register
Immediate
Indexed
PC-relative

MSP432
84 I/O pins
100 total pins

14mm
Module 3

Lecture: ARM Cortex M Assembly Programming
Gate-level view of digital XOR gate

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A^B</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
ARM Cortex M Assembly Programming

You will learn in this module

- Assembly Programming
  - Logical and shift operations
  - Addition, subtraction, multiplication and divide
  - Accessing memory
  - Stack
  - Functions, parameters
  - Conditionals
  - Loops
Logic Operations

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A&amp;B</th>
<th>A</th>
<th>B</th>
<th>A^B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ \text{AND} \{Rd,\} \ Rn, \ <op2> \ ; Rd=Rn \& op2 \]
\[ \text{ORR} \{Rd,\} \ Rn, \ <op2> \ ; Rd=Rn \mid op2 \]
\[ \text{EOR} \{Rd,\} \ Rn, \ <op2> \ ; Rd=Rn \oplus op2 \]

\[ \text{ORR} \ R0,R1,R2 \]
\[ R1 \ 0001 \ 0010 \ 0011 \ 0100 \ 0101 \ 0110 \ 0111 \ 1000 \]
\[ R2 \ 1000 \ 0111 \ 0110 \ 0101 \ 0100 \ 0011 \ 0010 \ 0001 \]
\[ \text{ORR} \ 1001 \ 0111 \ 0111 \ 0101 \ 0101 \ 0111 \ 0111 \ 1001 \]
Shift Operations

**Logical Shift Right**

- `LSR Rd, Rm, Rs` ; logical shift right `Rd=Rm>>Rs` (unsigned)
- `LSR Rd, Rm, #n` ; logical shift right `Rd=Rm>>n` (unsigned)

**Arithmetic Shift Right**

- `ASR Rd, Rm, Rs` ; arithmetic shift right `Rd=Rm>>Rs` (signed)
- `ASR Rd, Rm, #n` ; arithmetic shift right `Rd=Rm>>n` (signed)

**Logical Shift Left**

- `LSL Rd, Rm, Rs` ; shift left `Rd=Rm<<Rs` (signed, unsigned)
- `LSL Rd, Rm, #n` ; shift left `Rd=Rm<<n` (signed, unsigned)
Arithmetic Operations

- **Addition/subtraction**
  - Two n-bit → n+1 bits

- **Multiplication**
  - Two n-bit → 2n bits

- **Avoid overflow**
  - Restrict input values
  - Promote to higher, perform, check, demote

- **Division**
  - Avoid divide by 0
  - Watch for dropout

- **Signed versus unsigned**
  - Either signed or unsigned, not both
  - Be careful about converting types
Addition and Subtraction

Sets the carry bit

\[ +0x4000.0000 \]
\[ 0 \]
\[ 0x4000.0000 \]
\[ 0x2000.0000 \]
\[ 0x0000.0000 \]
\[ 0xFF.FFFF \]

**Condition**

<table>
<thead>
<tr>
<th>Code Bits</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>negative</td>
</tr>
<tr>
<td>Z</td>
<td>zero</td>
</tr>
<tr>
<td>V</td>
<td>overflow</td>
</tr>
<tr>
<td>C</td>
<td>carry</td>
</tr>
</tbody>
</table>

**<op2>**

- Register
- Register, shifted
- Constant

\[ \text{ADD} \{ \text{Rd,} \} \text{ Rn, } \text{<op2>} \quad ; \text{Rd} = \text{Rn} + \text{op2} \]
\[ \text{SUB} \{ \text{Rd,} \} \text{ Rn, } \text{<op2>} \quad ; \text{Rd} = \text{Rn} - \text{op2} \]
\[ \text{CMP Rn, } \text{<op2>} \quad ; \text{Rn} - \text{op2} \]
Multiplication and Division

```c
uint32_t N, M;
// times 0.6
void Fun(void) {
    M = 3*N/5;
}
```

```assembly
.data
.align 2
N .space 4
M .space 4
.text
.align 2
Fun: .asmfunc
    LDR R3, Naddr ; R3 = &N (R3 points to N)
    LDR R1, [R3] ; R1 = N
    MOV R0, #3 ; R0 = 3
    MUL R1, R0, R1 ; R1 = 3*N
    MOV R0, #5 ; R0 = 5
    UDIV R0, R1, R0 ; R0 = 3*N/5
    LDR R2, MAddr ; R2 = &M (R2 points to M)
    STR R0, [R2] ; M = 3*N/5
    BX LR
.endasmfunc
NAddr .field N, 32
MAddr .field M, 32
```

MUL {Rd,} Rn, Rm ; Rd = Rn * Rm
UDIV {Rd,} Rn, Rm ; Rd = Rn/Rm unsigned
SDIV {Rd,} Rn, Rm ; Rd = Rn/Rm signed
Stack

**Push**
1. SP = SP - 4
2. Store at SP

**POP**
1. Read at SP
2. SP = SP + 4

**Usage**
- Temporary storage
- Local variables

```
PUSH {R0}
PUSH {R1}
PUSH {R2}
POP  {R3}
POP  {R4}
POP  {R5}
```

```
Push
1. SP = SP - 4
2. Store at SP

POP
1. Read at SP
2. SP = SP + 4
```

![Stack Diagram](image)
Function calls

```asm
.data
.align 2
M .space 4
.text
.align 2
Seed: .asmfunc
    LDR R1,MAddr ; R1=&M
    STR R0,[R1] ; set M
    BX LR
    .endasmfunc
Rand: .asmfunc
    LDR R2,MAddr ; R2=&M, address of M
    LDR R0,[R2] ; R0=M, value of M
    LDR R1,Slope
    MUL R0,R0,R1 ; R0 = 1664525*M
    LDR R1,Offst
    ADD R0,R0,R1 ; 1664525*M+1013904223
    STR R0,[R2] ; store M
    LSR R0,#24 ; 0 to 255
    BX LR
    .endasmfunc
MAddr .field M,32
Slope .field 1664525,32
Offst .field 1013904223,32
```

```
.main
    MOV R0,#1
    BL Seed

loop
    BL Rand
    LDR R1,nAddr
    STR R0,[R1]
    B loop

.endasmfunc
nAddr .field n,32
```
Conditionals

G1 <= G2  G1 > G2

Yes()  No()

LDR R3,G2Addr ; R3=&G2, address of G2
LDR R2,[R3]  ; R2=G2, value of G2
LDR R0,G1Addr ; R0=&G1, address of G1
LDR R1,[R0]  ; R1=G1, value of G1
CMP R1,R2   ; compare G1 G2
BHI isNo
isYes       BL Yes ; G1<=G2
B done
isNo        BL No
done

G1Addr .field G1,32
G2Addr .field G2,32

Instruction  Branch if
B target      ; always
BEQ target    ; equal (signed or unsigned)
BNE target    ; not equal (signed or unsigned)
BLO target    ; unsigned less than
BLS target    ; unsigned less than or equal to
BHS target    ; unsigned greater than or equal to
BHI target    ; unsigned greater than
BLT target    ; signed less than
BGE target    ; signed greater than or equal to
BGT target    ; signed greater than
BLE target    ; signed less than or equal to

if(G2<=G1){
  Yes();
} else{
  No();
}

Think of the three steps
1) bring first value into a register,
2) compare to second value,
3) conditional branch, bxx
(where xx is eq ne ls hi hs gt ge lt or le).
The branch will occur if (first is xx second).
While Loops

```assembly
while(G2>G1) {
    Body();
}

LDR R3,G2Addr ; R3=&G2, address of G2
LDR R2,[R3]   ; R2=G2,  value of G2
LDR R0,G1Addr ; R0=&G1, address of G1
LDR R1,[R0]   ; R1=G1,  value of G1

loop CMP R1,R2 ; compare G1 G2
    BLS done
    BL  Body ; G1>G2
    B   loop

done

G1Addr .field G1,32 ;unsigned 32-bit number
G2Addr .field G2,32 ;unsigned 32-bit number
```
For Loops

```assembly
MOV R4,#10
loop
CMP R4,#0
BEQ done
BL Body
SUB R4,R4,#1
B loop

done
```

```
for(i=10; i!=0; i--){
    Body(); // 10 times
}
```
ARM Cortex M Assembly Programming

Summary

- Programming
  - Accessing memory
  - Logical and shift operations
  - Addition, subtraction, multiplication and divide
  - Stack
  - Functions, parameters
  - Conditionals
  - Loops

Register
Immediate
Indexed
PC-relative
IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated