Module 4

Lecture: Software Design using MSP432 - Design
Software Design using MSP432

You will learn in this module

- Software Design
  - Call graph
  - Data Flow Graph
  - Successive refinement
  - Abstraction (functions)
  - Modular design (header/code files)
Multi-threading

Interrupt-driven concurrent

- main
  - Init
  - Body

Trigger interrupt

Return from interrupt
System Design

What does being in a state mean?
• List state parameters

What is the starting state of the system?
• Define the initial state

What information do we need to collect?
• List the input data

What information do we need to generate?
• List the output data

How do we move from one state to another?
• Actions we could do

What is the desired ending state?
• Define the ultimate goal
Successive Refinement

- Start with a task
  - Clear and unambiguous description: requirements, specifications
- Decompose the task into a set of simpler subtasks (components)
  - Subtasks are decomposed into even simpler sub-subtasks
  - Each subtask is simpler than the task itself
- Make design decisions
  - Document decisions and subtask requirements
- Ultimately, subtask is so simple, it can be implemented
  - Implementation
  - Testing
  - Documentation
- Combine components to build system
  - Interfaces are key

Three similar terms:
- Successive Refinement
- Stepwise Refinement
- Systematic Decomposition
Header files

- Why do we have header/code files?
  - Complexity abstraction
  - Separate what it does (header) from how it works
  - Automatic documentation (doxygen)

- What is in a header file?
  - Prototypes for public functions
  - Comments on what it does/how to use it
  - Code to make it load once
  - Shared structure

- What is not in a header file?
  - Function definitions
  - Variables
  - Anything private

```c
/**
 * @file Switch.h
 /**
 * Input from positive logic switch
 * interfaced to GPIO Port 1 bit 5.
 * @param none
 * @return 0x20 if pressed; 0x00 if not pressed
 * @brief Switch input
 */
uint32_t Switch_Input(void);
```
Code files

- What is in a code file?
  - Implementations for public functions
  - Variables
  - Private functions
  - Comments how it works
  - Comments on how it was tested
  - Comments on how it can be changed

- What is not in a code file?
  - References to private data/functions in other files

#include <stdint.h>
#include "Switch.h"
#include "../inc/LaunchPad.h"

```c
//--------------Switch_Input--------------
// Read and return P1.5
// Input: none
// Output: 0x20 if P1.5 is high
//        0x00 if P1.5 is low
uint32_t Switch_Input(void)
{
    // read P1.5 input
    return (P1->IN&0x20);
    // return 0x20(if pressed)
    // or 0(if not pressed)
}
```

Call Graph

![Call Graph Diagram](image-url)
Software Design using MSP432

Summary

- Software design
  - Successive refinement
  - doxygen
  - Header/code files
  - Abstraction
Module 4

Lecture: Software Design using MSP432 - C Programming
C Programming on the MSP432

You will learn in this module

- Basics of C programming
  - Logic/shift operations
  - Arithmetic calculations
  - Conditionals
  - Loops
  - Functions
  - Variables
  - Constants

- Algorithm development (lab)
  - GP2Y0A21YK0F IR distance sensor
  - Where in the world am I?
Flowcharts
Logic Operations

**AND**
- Select bits (AND with 1)
- Clear bits (AND with 0)

**OR**
- Combine
- Set bits (OR with 1)

**EOR**
- Toggle bits (EOR with 1)

<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></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>~A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- \( y = \text{P1->IN} & 0x03; \) // select bits 1,0
- \( x = x & (\sim 0x08); \) // clear bit 3
- \( x &= \sim 0x08; \) // clear bit 3

- \( z = x | y; \) // combine x,y
- \( x = x | 0x08; \) // set bit 3
- \( x |= 0x08; \) // set bit 3

- \( \text{P1->OUT} ^= 0x08; \) // toggle bit 3
Shift Operations

Unsigned (logical) shift right
- Divide by $2^n$
- Align bits

Signed (arithmetic) shift right
- Divide by $2^n$

Shift left (logical/arithmetic)
- Multiply by $2^n$
- Align bits

\[
y = x \gg 3; \quad \text{// divide by 8}
\]

\[
x = \text{P1} \rightarrow \text{IN} \& 0x01; \quad \text{// P1.0 (0,1)}
\]

\[
y = \text{P2} \rightarrow \text{IN} \& 0x08; \quad \text{// P2.3 (0,8)}
\]

\[
z = (x << 1) | (y \gg 3); \quad \text{// combine}
\]

\[
y = x \ll 8; \quad \text{// multiply by 256}
\]
Arithmetic Operations

- Addition/subtraction
  - Two n-bit → n+1 bits
- Multiplication
  - Two n-bit → 2n bits
- Division
  - Avoid divide by 0
  - Watch for dropout
- Avoid overflow
  - Restrict input values
  - Promote to higher, perform, check, demote
- Signed versus unsigned
  - Either signed or unsigned, not both
  - Be careful about converting types

```c
uint8_t Add(uint8_t A, uint8_t B){
    uint32_t A32,B32,R32;
    A32 = A; B32=B; // promotion
    R32 = A+B;       // 32-bit addition
    if(R32>255){
        R32 = 255; // ceiling
    }
    return R32;     // demotion
}
```

Diagram:
- Unsigned add
  - Promote A to A32
  - Promote B to B32
  - R32 = A32 + B32
  - R32 ≥ 255: Overflow
  - R32 ≤ 255: OK
- R = R32
- R = 255
- End
Conditionals

- **Boolean**
  - Zero is false
  - Nonzero is true
  - && || ! are operators

- **Relational**
  - Compare similar types
  - Returns a Boolean
  - > >= < <= == !=

- **Conditional**
  - if-then
  - if-then-else

```c
if((G1<=G2)&&(G3!=G4)){
  Yes();
}else{
  No();
}
```

```c
if(P1->IN&0x80){
  Something(); // if P1.7 is high
}
```

These are different & &&
These are different | ||
while loops

while loop
• Test first

do-while loop
• Test last

for loop
• Test first

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

```c
do{
    Body();
} while(G2>G1);
```

```c
for(i=10; i!=0; i--){
    Body();
}
```

```c
for(i=0; i<10; i++){  
    Body();
}  
```

```
```
Functions

- What it does
  - Prototype
  - Header file

- How it works
  - Implementation
  - Code file

- Invocation
  - Calling sequence
  - Inputs: call by value/reference
  - Output: return value

```c
// random.h
void Seed(uint32_t x);
uint8_t Rand(void);

// random.c
uint32_t static M;
void Seed(uint32_t x){
  M = x;
}
uint8_t Rand(void){
  M=1664525*M+1013904223;
  return M>>24;
}

// main.c
uint8_t n;
void main(void){
  Seed(1);
  while(1){
    n = Rand();
  }
}
```
Examples of variables

- **Global**
  - Public scope
  - Permanent allocation
  - Bad style

- **Static**
  - Private scope to file
  - Permanent allocation
  - Sharing: ISR ↔ Functions

- **Local - Automatic**
  - Private scope,
  - Dynamic allocation

- **Static local**
  - Private scope to function
  - Permanent allocation

```c
uint32_t static M;
void Seed(uint32_t x)
{
    M = x;
}
uint8_t Rand(void)
{
    uint32_t n;
    uint32_t static count=0;
    count++;
    M=1664525*M+1013904223;
    n = M>>24;
    return (uint8_t)n;
}
```

```c
uint8_t global;
void main(void)
{
    uint8_t n;
    Seed(1);
    while(1)
    {    
        n = Rand();
    }
}
```
Variables

Scope => from where can it be accessed
- Private means restricted, need to know basis
  - More protection, simpler systems
- Public means any software can access it
  - Difficult to debug, hidden complexity

Allocation => when is it created & destroyed
- Dynamic allocation using registers or stack
- Permanent allocation assigned a block of memory

Type
- Signed/unsigned
- Precision: 8, 16, 32 bits

Can you convert between types?
- `uint8_t` → `uint16_t`, `int16_t`, `uint32_t`, `int32_t`
- `int8_t` → `int16_t`, `int32_t`
- `uint16_t` → `uint32_t`, `int32_t`
- `int16_t` → `int32_t`

How does one classify I/O port registers?
- Formally: Global = public permanent
- Practically: private permanent
Examples of constants

Symbol
- `#define`

ROM
- `const`

Enumerated types
- `enum`

```c
#define IRSlope 1195172
#define IROffset -1058

int32_t const ADCBuffer[16] = 
{2000, 2733, 3466, 4199, 4932, 5665, 6398, 7131, 7864, 8597, 9330, 10063, 10796, 11529, 12262, 12995};

typedef enum scenario scenario_t;
```

```c
enum scenario {
    Error = 0,
    LeftTooClose = 1,
    RightTooClose = 2,
    CenterTooClose = 4,
};
```
Software design, building blocks

- “do A then do B” → sequential
- “do A and B in either order” → sequential (parallel)
- “if A, then do B” → conditional
- “for each A, do B” → iterative
- “do A until B” → iterative
- “repeat A over & over forever” → iterative (condition always true)
- “on external event do B” → interrupt
- “every t msec do B” → interrupt
C Programming using MSP432

Summary

- Review C programming
  - Logic/shift operations
  - Arithmetic calculations
  - Functions
  - Conditionals
  - Variables
  - Constants

```
main
  Seed(1)
  n = Rand()
  M = x
  return

Seed(x)
  M = 1664525 * M + 1013904223
  return M

Rand

(G1 <= G2) && (G3 != G4)
  Yes()
  No()

(G1 > G2) || (G3 == G4)

G2 > G1
  Body
G2 <= G1
```
Module 4

Lecture: Software Design using MSP432 - Debugging
Debugging on the MSP432

You will learn in this module

- Debugging
  - Control (step, breakpoints)
  - Observing variables
  - Functional debugging

![Graph showing expected results of a function with input and output data points.](Expected Results.png)
Debugging

- Functional Debugging
  - Known inputs
  - Expected outputs

- Stabilization
  - Fix input values, fix timing of input
  - Repeated testing shows changes in software

- Test cases
  - Near the extremes and in the middle
  - Most typical of how clients will properly use the system
  - Most typical of how clients will improperly use the system
  - That differ by one
  - You know your system will find difficult (corner cases)
  - Using a random number generator

```
// Program 4_1 used to test the Convert function
int32_t const ADCBuffer[16]={2000, 2733, 3466, 4199, 4932, 5665, 6398, 7131, 7864, 8597, 9330, 10063, 10796, 11529, 12262, 12995};
int32_t const DistanceBuffer[16]={800, 713, 496, 380, 308, 259, 223, 196, 175, 158, 144, 132, 122, 114, 106, 100};
void Program4_1(void){
  int i;
  int32_t adc,distance,errors,diff = 0;
  errors = 0;
  for(i=0; i<16; i++){
    adc = ADCBuffer[i];
    distance = Convert(adc); // call to your function
    diff = distance-DistanceBuffer[i];
    if((diff<-1)||(diff>1)){
      errors++;
    }
  }
  while(1){};
```

Important aspects:
- Control
- Observability
Debugging (Control)

- Test cases
  - Get data from arrays (rather than actual inputs devices)

- Single step
  - Step, step over, step in, step out

- Breakpoints
  - Set using debugger

- Special test main
  - Establish exact scenario you wish to test
  - Stabilization

```c
int32_t errors;
void Program4_2(void){
  scenario_t result,truth;
  int i,j,k;
  int32_t left, right, center; // sensor readings
  errors = 0;
  for(i=0; i<18; i++){
    left = CornerCases[i];
    for(j=0; j<18; j++){
      center = CornerCases[j];
      for(k=0; k<18; k++){
        right = CornerCases[k];
        result = Classify(left,center,right);
        truth = Solution(left,center,right);
        if(result != truth){
          errors++;
        }
      }
    }
  }
  while(1){
  }
}
```
Debugging (Observability)

- Debugger monitor windows
  - Globals
  - Locals
  - I/O registers

- Dump
  - Save results in RAM or ROM

- Output to UART
  - Observe with terminal program like PuTTY or TExaSdisplay

- Hardware Monitors
  - Lights, sounds
  - Nokia 5110 LCD display

Important aspects:
- Control
- Observability
Summary

- Debugging
  - Control
  - Observability
  - Functional debugging
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