

TI-RSLK

Texas Instruments Robotics System Learning Kit



TEXAS INSTRUMENTS



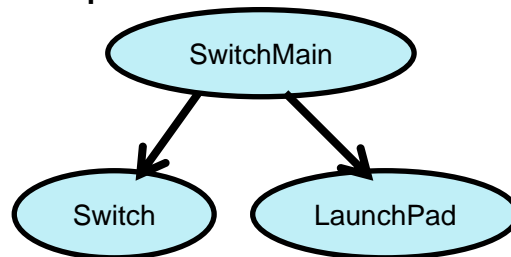
Module 4

Lecture: Software Design using MSP432 - Design

You will learn in this module

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

Call Graph

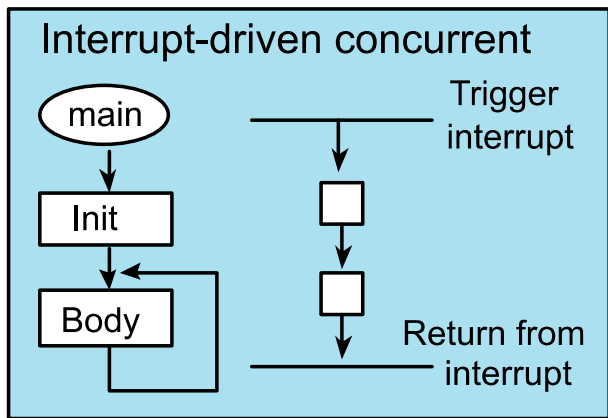


Data Flow Graph





Multi-threading





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

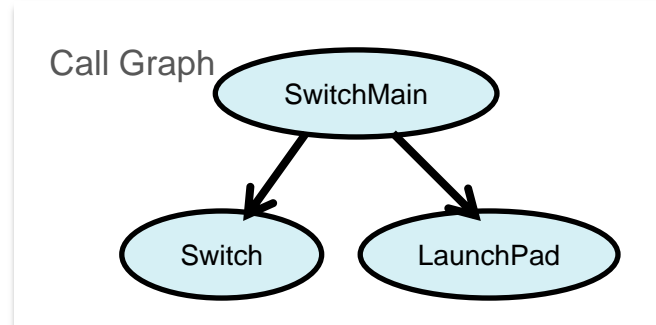
Three similar terms:

- 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



```
/**  
 * @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);
```



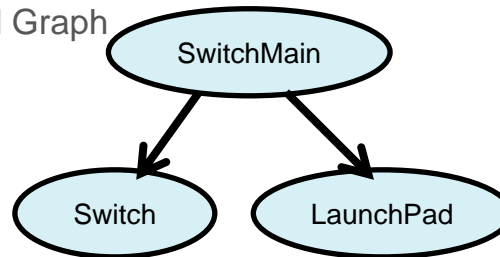
Code files

- 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"
```

```
//-----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)
}
```

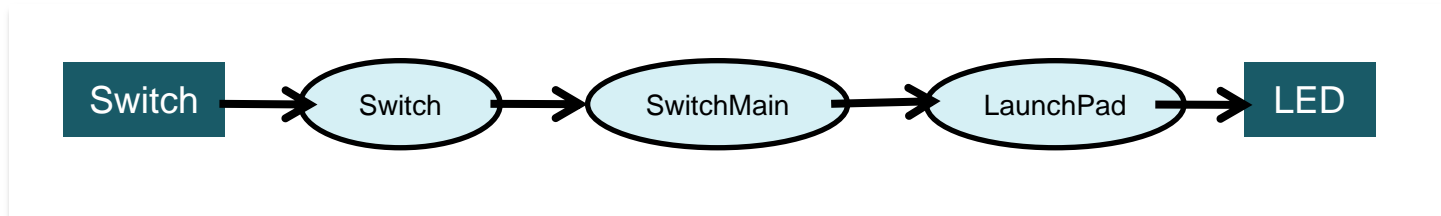
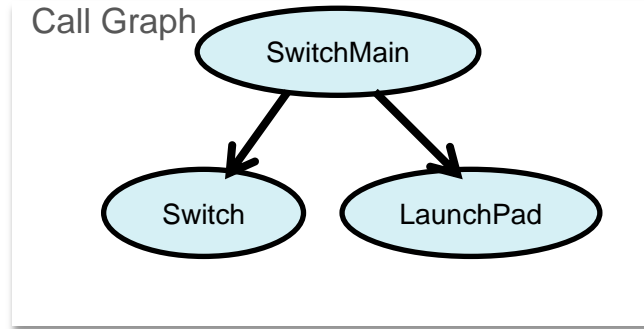
Call Graph



- ▼ Switch [Active - Debug]
 - > Includes
 - > Debug
 - > targetConfigs
 - > LaunchPad.c
 - > msp432p401r.cmd
 - > startup_msp432p401r_ccs.c
 - > Switch.c
 - > Switch.h
 - > Switchmain.c
 - > system_msp432p401r.c

Summary

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





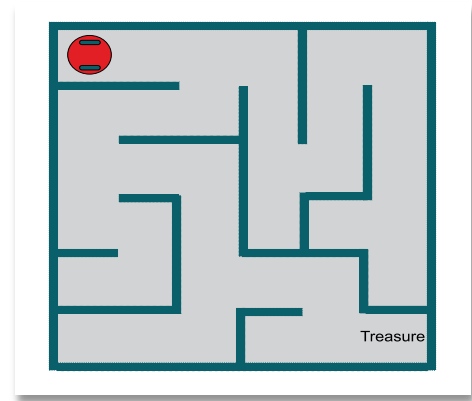
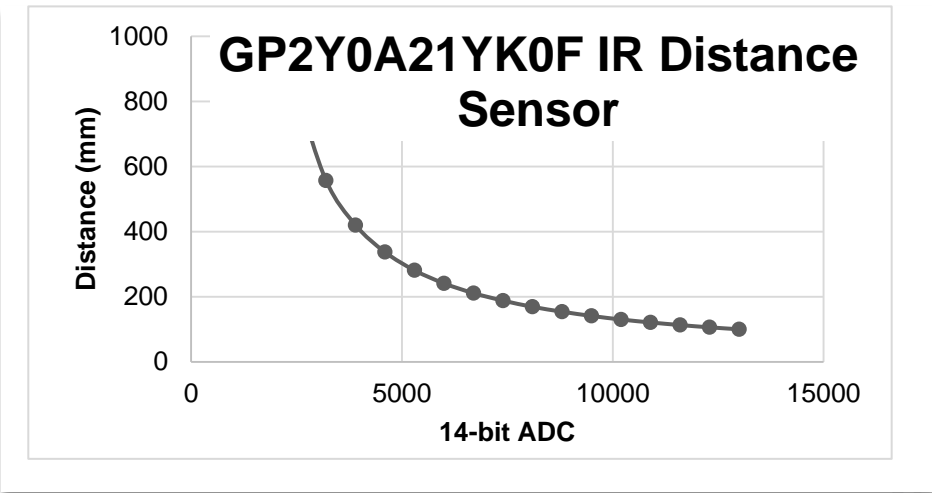
Module 4

Lecture: Software Design using MSP432 - C Programming

C Programming on the MSP432

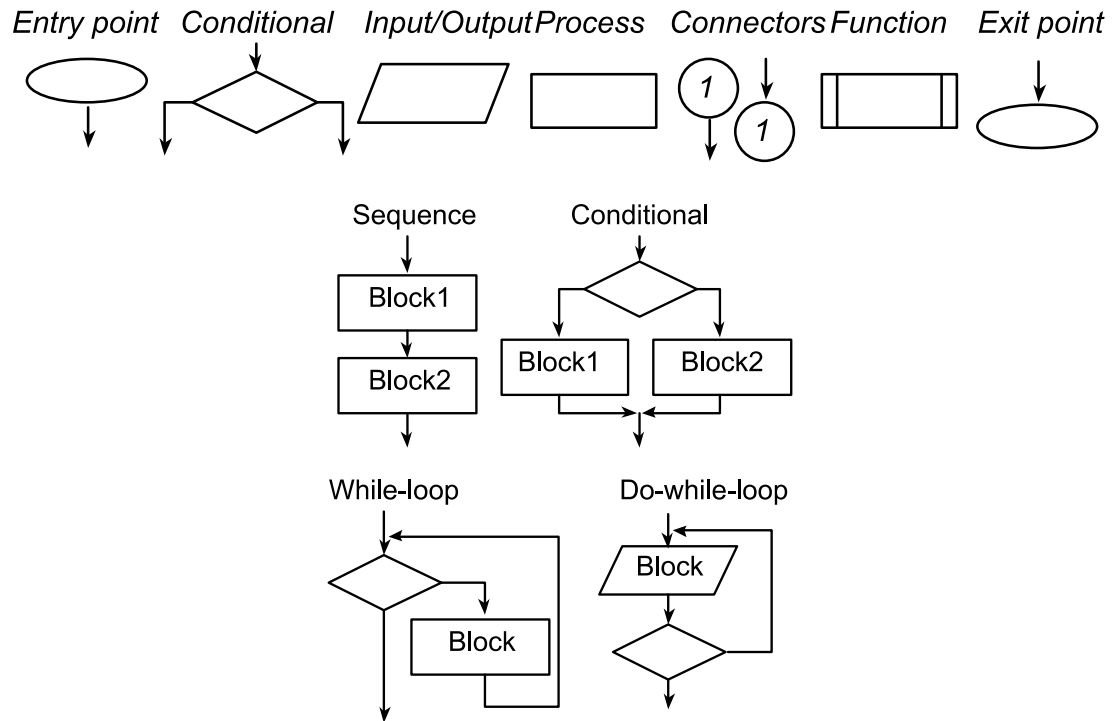
You will learn in this module

- 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

A	B	A&B	A B	A^B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

A	~A
0	1
1	0

AND

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

```
y = P1->IN&0x03; // select bits 1,0  
x = x&(~0x08); // clear bit 3  
x &= ~0x08; // clear bit 3
```

OR

- Combine
- Set bits (OR with 1)

```
z = x|y; // combine x,y  
x = x|0x08; // set bit 3  
x |= 0x08; // set bit 3
```

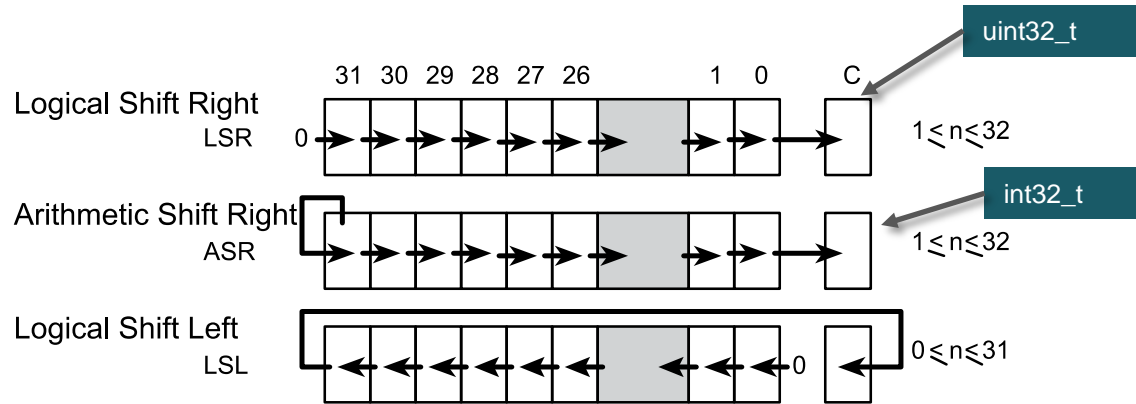
EOR

- Toggle bits (EOR with 1)

```
P1->OUT ^= 0x08; // toggle bit 3
```



Shift Operations



Unsigned (logical) shift right

- Divide by 2^n
- Align bits

```
y = x >> 3; // divide by 8
```

Signed (arithmetic) shift right

- Divide by 2^n

```
x = P1->IN&0x01; // P1.0 (0,1)
y = P2->IN&0x08; // P2.3 (0,8)
z = (x<<1) | (y>>3); // combine
```

Shift left (logical/arithmetic)

- Multiply by 2^n
- Align bits

```
y = x << 8; // multiply by 256
```

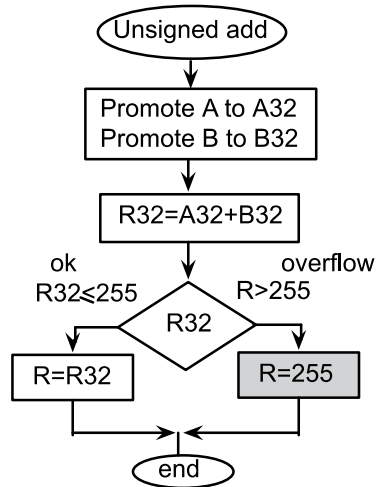


Arithmetic Operations

- Addition/subtraction
 - Two n-bit \rightarrow n+1 bits
- Multiplication
 - Two n-bit \rightarrow 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

```
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
}
```

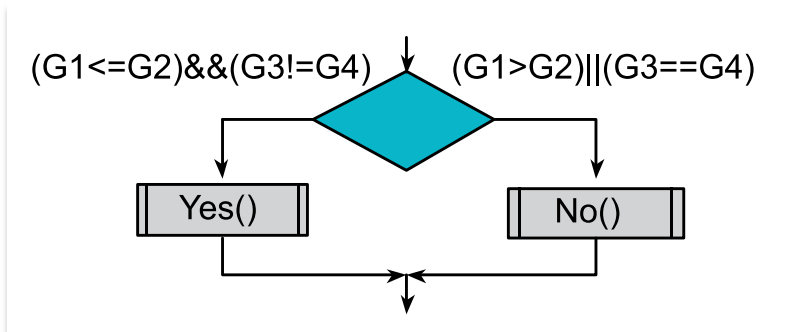
```
uint8_t   int8_t
uint16_t  int16_t
uint32_t  int32_t
```





Conditionals

- Boolean
 - Zero is false
 - Nonzero is true
 - `&&` `||` `!` are operators
- Relational
 - Compare similar types
 - Returns a Boolean
 - `>` `>=` `<` `<=` `==` `!=`
- Conditional
 - if-then
 - if-then-else



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

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

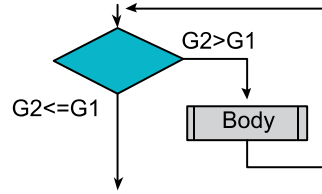
These are different `&` `&&`
These are different `|` `||`



while loops

while loop

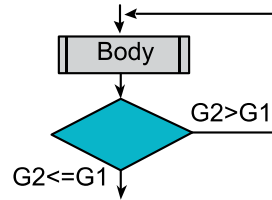
- Test first



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

do-while loop

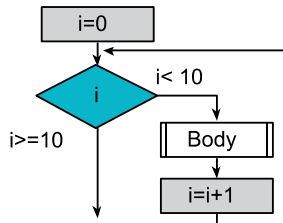
- Test last



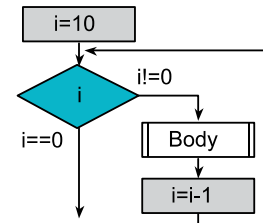
```
do{
  Body ();
} while (G2>G1);
```

for loop

- Test first



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



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

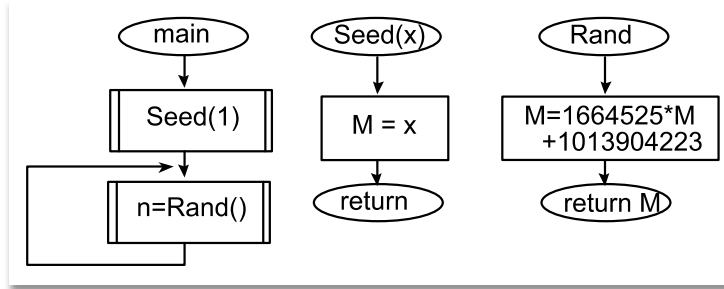


Functions

- What it does
 - Prototype
 - Header file

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

- How it works
 - Implementation
 - Code file
- Invocation
 - Calling sequence
 - Inputs: call by value/reference
 - Output: return value



```
// 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

```
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;
}
```

```
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

Can access

How does one classify I/O port registers?

- Formally: Global = public permanent
- Practically: private permanent

Does access



Examples of constants

Symbol

- #define

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

ROM

- const

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

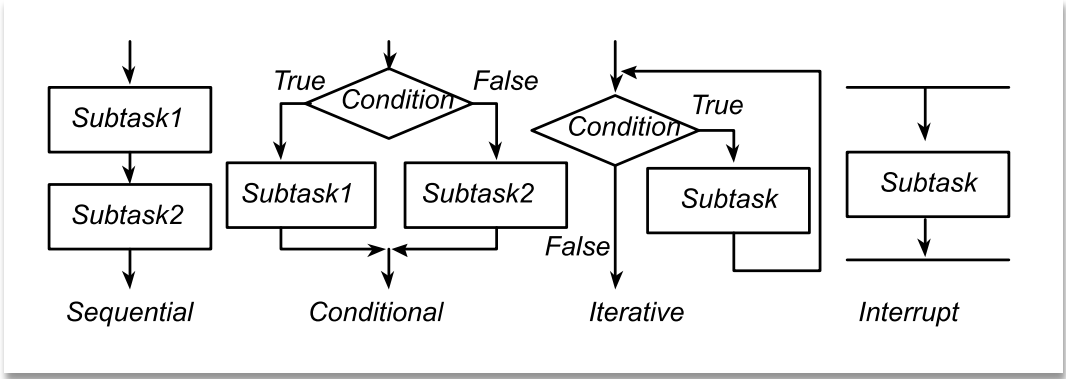
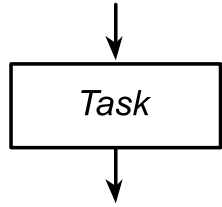
Enumerated types

- enum

```
enum scenario {
    Error = 0,
    LeftTooClose = 1,
    RightTooClose = 2,
    CenterTooClose = 4,
};
typedef enum scenario scenario_t;
```

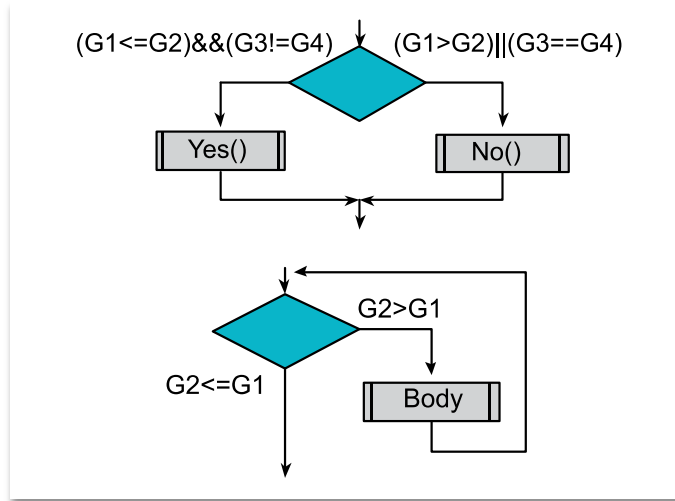
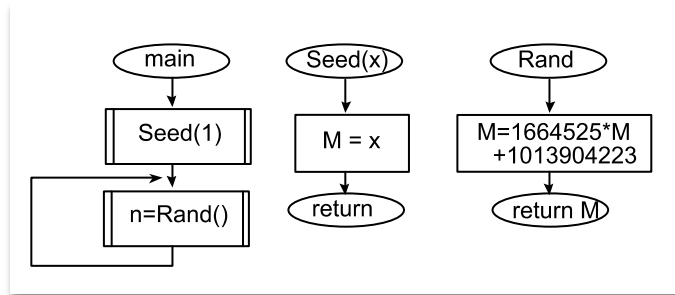
Software design, building blocks

- “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



Summary

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





Module 4

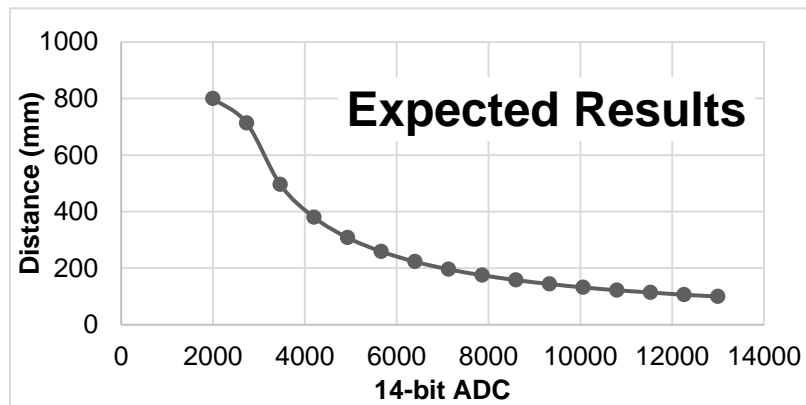
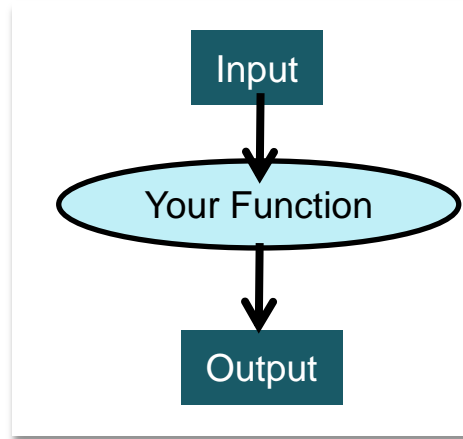
Lecture: Software Design using MSP432- Debugging



Debugging on the MSP432

You will learn in this module

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





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

Important aspects:

- Control
- Observability

```
// 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;
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){};
}
```



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

Important aspects:

- Control
- Observability

```
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:

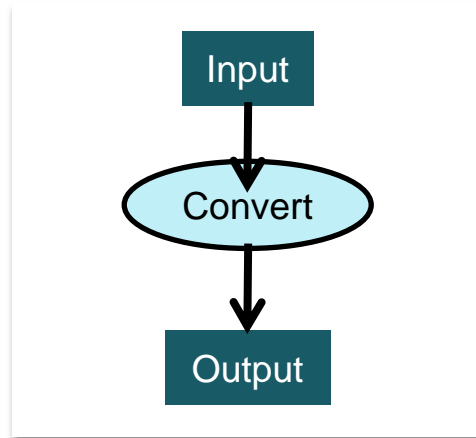
- Control
- Observability



Debugging on the MSP432

Summary

- Debugging
 - Control
 - Observability
 - Functional debugging



IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ("TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products (<http://www.ti.com/sc/docs/stdterms.htm>), [evaluation modules](#), and [samples](http://www.ti.com/sc/docs/sampterm.htm) (<http://www.ti.com/sc/docs/sampterm.htm>).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2018, Texas Instruments Incorporated