

# EE445M/EE380L.12

## Embedded and Real-Time Systems/ Real-Time Operating Systems

### Lecture 2: Software Development, Design, Implementation, Debugging

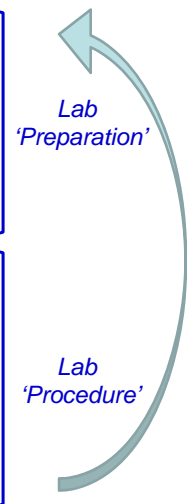
Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

1

## Development Process

1. Analyze (Ask)
  - Requirements, specifications
2. Design (Think)
  - Dataflow/call graphs, flowcharts, OO
3. Implement (Do)
  - Code, coding styles, documentation
4. Debug and test (Check)
  - Verification (correctness)
  - Validation (performance)



Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

2

## Development Tools

- Editor
- Version/Revision Control
- (Cross-)Compiler
- Simulator
- Debugger
- Integrated Development Environment (IDE)

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

3

## Announcements

- Labs
  - Week of 1/28: TA demos
  - Week of 2/4: Lab 1 Prep
  - Week of 2/11: Lab 1 Demo/Report, Lab 2 Prep
- Lab submissions
  - Preparation: software design (oral)
  - Demo: software implementation (Github)
    - Submission and repository info will be mailed
  - Report: software documentation (Canvas)

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

4

## Modular Programming

- Encapsulation (private)
  - Information hiding
  - Reduce coupling
- Abstraction (public)
  - Well-defined external interfaces
  - Separate mechanism from policy

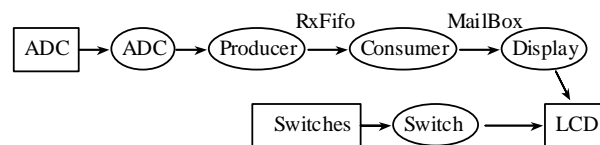
Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

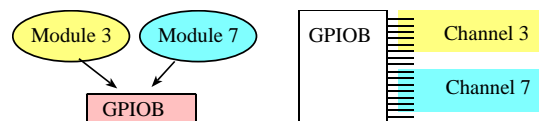
5

## Dataflow & Call Graphs

- Dataflow graphs
  - Parallel or sequential execution



- Call graphs
  - Use to identify potential conflicts



Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

6

## Modular Programming in C

- Naming conventions
  - Object name has *Module Name* and underline
    - E.g. `UART_xxx`
- Public object declarations in header file
  - Avoid public global variables
    - Otherwise declared as `extern`
- Private objects & definitions in C file
  - Private globals have `static` modifier
    - Public globals locally defined w/o `extern`

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

7

## Object-Oriented Programming (OOP)

- Elevate modules into first-class citizens
  - Encapsulation of data & code
    - Member variables and functions (*methods*)
  - Dynamic vs. static scope & lifetime
    - Blueprint (*class*) and instances (*objects*)
    - Initialization & tear-down (*constructor/destructor*)
  - Plus inheritance, polymorphism, ...
    - Further increase opportunities for reuse

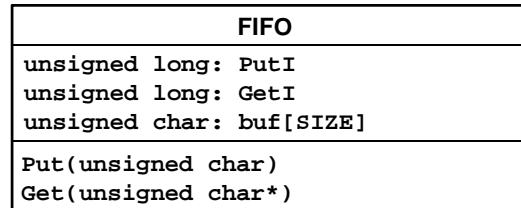
Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

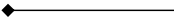
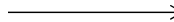
8

## Class Diagrams (UML)

- Classes & objects



- Relationships

- Membership (“has a”) 
- Inheritance (“is a”) 

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

9

## Objects in C

- “Class” declaration (header file)

```
#define AddFifo(NAME,SIZE,TYPE, SUCCESS,FAIL) \
unsigned long volatile PutI ## NAME; \
unsigned long volatile GetI ## NAME; \
TYPE static Fifo ## NAME [SIZE]; \
void NAME ## Fifo_Init(void){ \
    PutI ## NAME= GetI ## NAME = 0; \
} \
int NAME ## Fifo_Put (TYPE data){ \
    if(( PutI ## NAME - GetI ## NAME ) & ~(SIZE-1)){ \
        return(FAIL); \
    } \
    Fifo ## NAME[ PutI ## NAME &(SIZE-1)] = data; \
    PutI ## NAME ## ++; \
    return(SUCCESS); \
} \
int NAME ## Fifo_Get (TYPE *datapt){ \
    if( PutI ## NAME == GetI ## NAME ){ \
        return(FAIL); \
    } \
    *datapt = Fifo ## NAME[ GetI ## NAME &(SIZE-1)]; \
    GetI ## NAME ## ++; \
    return(SUCCESS); \
}
```

“Member” variables

“Methods”

- Object instantiation

```
AddFifo(Tx,32,unsigned char, 1,0)
```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

10

# Structured Programming

- Flowcharts

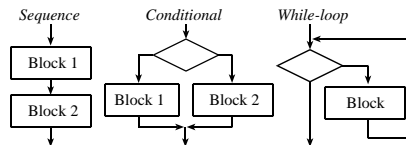


Figure 2.1. Flowchart showing the basic building blocks of structured programming.

- Parallel constructs

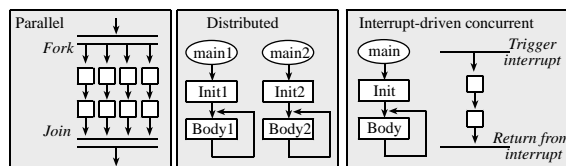


Figure 2.2. Flowchart symbols to describe parallel, distributed, and concurrent programming.

# Design Patterns

- Buffered I/O

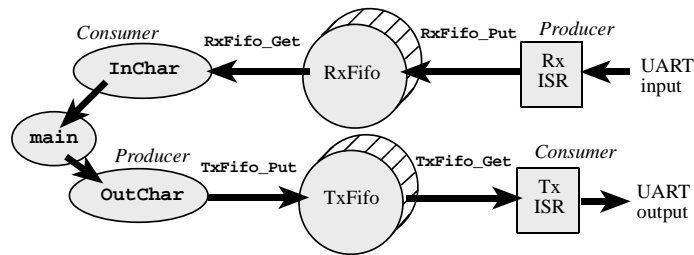
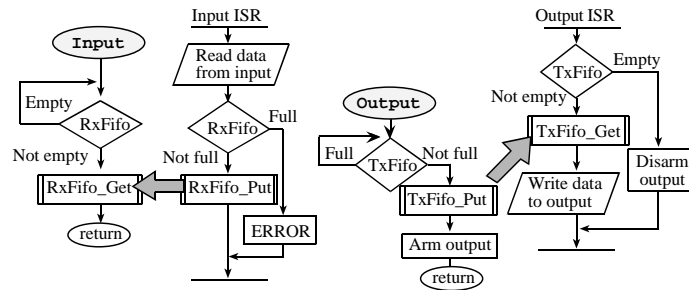


Figure 3.3. A data flow graph showing two FIFOs that buffer data between producers and consumers.

[UARTInts\\_4C123.zip](#)

# Buffered I/O

- Using interrupt-driven I/O synchronization



UARTInts\_4C123.zip

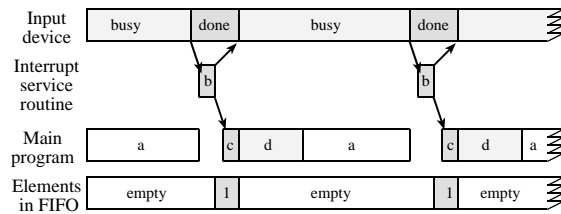
Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

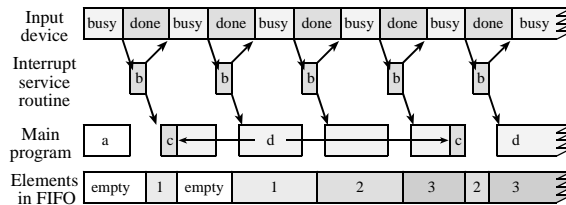
13

# Input Synchronization

- I/O bound input device (slow input)



- CPU bound input device (fast input)



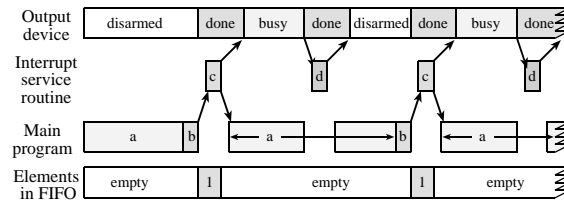
Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

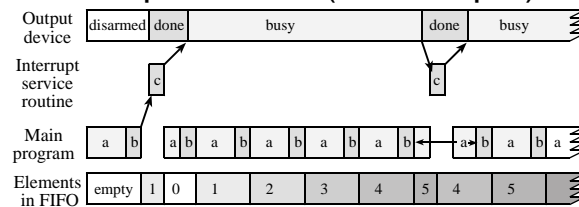
14

# Output Synchronization

- CPU bound output device (fast output)



- I/O bound output device (slow output)



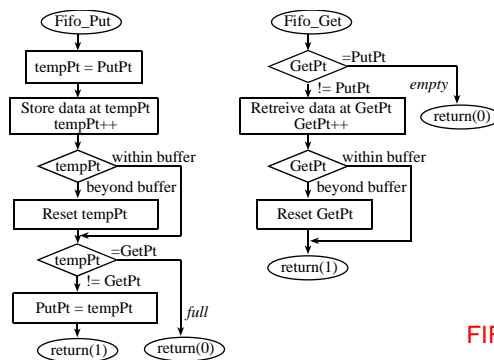
Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

15

# FIFO Implementation

- Pointer- or index-based implementation



FIFO\_4C123.zip

Figure 3.19. Flowcharts of the pointer implementation of the FIFO queue.

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

16



## Coding Style

- Find your own conventions
  - Naming
    - Meaning, type of variables & functions
  - Readability
    - Indentation, white space
  - Documentation
    - Comments for every module, function, code block

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

17

## Testing

- Unit vs. integration testing
  - Test function, module before integrating into next bigger system
- Black box vs. white box testing
  - Just inputs/outputs vs. can probe inside
  - Know what it does vs. know how it works
  - Have interfaces vs. have internals

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

18

## Debugging

1. Stabilize the system (reproducibility)
  - Creating a test routine that fixes (or stabilizes) all inputs
    - Can reproduce the exact inputs over and over again
  - Modify the program
    - Change in outputs is a function of modification and not due to a change in the input parameters
  
2. Debugging instruments (control, observability)
  - Code that is added to isolate origin of bug
    - “Rough and ready” manual methods
      - Desk-checking, dumps, printf statements
  - Intrusive vs. non-intrusive
    - Measure of the degree of perturbation caused in program behavior by an instrument

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

19

## Debugging Tools (1)

- Software debuggers
  - Breakpoints
    - Replacing the instruction with a trap
    - Can not be performed when the software is in ROM
  - Single step
    - Implicit breakpoints or periodic interrupts
  - Inspection
    - Processor state (registers), memory
  
- Hardware debuggers (local or remote via JTAG)
  - Interface with microcomputer chip itself
    - Communicates with the debugging computer
    - Ability to observe software execution in real time
  - Set breakpoints, single step
    - Ability to stop the computer and set hardware breakpoints
  - Processor, memory and I/O ports are accessible while running
    - Hardware support to break on events (e.g. memory access)

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

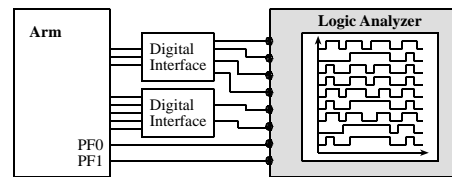
20

## Debugging Tools (2)

- Logic analyzer
  - Multiple channel digital storage scope
    - Numerous digital signals at various points in time
      - triggering to capture data at appropriate times
    - Good for real time observation of I/O signals
      - Attached to strategic I/O signals, real-time measurement
      - Attached to heart beats, profile execution
    - Massive amount of information
      - must interpret the data
    - Nonintrusive

- PC-based

<http://www.digilentinc.com/analogdiscovery/>  
 Software: <http://www.digilentinc.com/waveforms/>  
<http://www.usbee.com>  
<http://www.saleae.com>



*A logic analyzer and example output.*

Lecture 2

J. Valvano, A. Gerstlauer  
 EE445M/EE380L.12

21

## Functional Debugging

- Verification of input/output parameters
  - What data is processed at specific points
- A static process where
  - inputs are supplied,
  - the system is run, and
  - the outputs are compared against expected results.

Lecture 2

J. Valvano, A. Gerstlauer  
 EE445M/EE380L.12

22

## Functional Debugging Methods

- Intrusive methods
  - Single stepping or trace
  - Breakpoints
  - Instrumentation w/ print statements
    - Difficult in embedded systems
    - A standard output device may not be available
    - Output may be slow (relative to rest of system)
    - Output device used for normal operation
    - Send print output to special debug device
      - E.g. UART, see Lecture 1

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

23

## Debugging Instruments (1)

- Dump into array without filtering
  - Dumps strategic information into an array at run time
  - Observe the contents of the array at a later time
  - Use debugger to visualize when running

```
long DebugList[100];
unsigned int DebugCnt=0;
void RecordIt(long data){
    if(DebugCnt==100)return;
    DebugList[DebugCnt]=data;
    DebugCnt++;
}
```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

24

## Debugging Instruments (2)

- Dump into array with filtering
  - A software/hardware condition that must be true in order to place data into the array

```

if(condition){
    RecordIt(MyData);
}

```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

25

## Debugging Instruments (3)

- Monitor using output port
  - A monitor is an independent output process
    - executes very fast, so is minimally intrusive
    - small amounts of strategic information (enter/exit block)
  - Examples
    - LCD display
    - LED's on individual otherwise unused output bits

```

#define PF1 ((volatile unsigned long *)0x40025008)
#define GPIO_PORTF_DATA_R ((volatile unsigned long *)0x400253FC)
PF1 = 0x02; // atomic (specific for TM4C)
GPIO_PORTF_DATA_R |= 0x02; // not atomic
PF1 = 0x00; // atomic (specific for TM4C)
GPIO_PORTF_DATA_R &= ~0x02; // not atomic

```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

26

## Debugging Instruments (2)

- Monitor using output port
  - Measure using scope or logic analyzer
  - Again, atomicity in parallel/interrupt cases

4804	LDR	r0,[pc,#16]	;r0= 0x400063FC	GPIO_PORTC_DATA_R  = 0x20;
5800	LDR	r0,[r0,#0x00]	;r0=PORTC	
F0400020	ORR	r0,r0,#0x20	;set bit 5	
4903	LDR	r1,[pc,#12]	;r1= 0x40006000	
F8C103FC	STR	r0,[r1,#0x3FC]	;write PORTC	
4804	LDR	r0,[pc,#16]	;r0= 0x400063FC	GPIO_PORTC_DATA_R  = 0x40;
5800	LDR	r0,[r0,#0x00]	;r0=PORTC	
F0400040	ORR	r0,r0,#0x40	;set bit 6	
4903	LDR	r1,[pc,#12]	;r1= 0x40006000	
F8C103FC	STR	r0,[r1,#0x3FC]	;write PORTC	

*These subroutine have critical sections because of the read-modify-write access to a shared global.*

2020	MOVS	r0,#0x20		GPIO_PORTC5 = 0x20;
4902	LDR	r1,[pc,#8]	;r1=0x40006080	
6008	STR	r0,[r1,#0x00]	;set bit 5	
2020	MOVS	r0,#0x40		GPIO_PORTC6 = 0x40;
4902	LDR	r1,[pc,#12]	;r1=0x40006100	
6008	STR	r0,[r1,#0x00]	;set bit 6	

*These subroutines do not have critical sections because the write access is atomic (bit-specific addressing).*

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

27

## Performance Debugging

- Verification of timing behavior of system
  - When do specific events occur
  - Measure dynamic efficiency of software
    - Delta time spent in pieces of code
- A dynamic process
  - System is run, and
  - dynamic behavior compared to expected results

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

28

## Performance Analysis

- Count processor cycles
  - Very hard in modern processors
    - Many dynamic run-time effects
    - Pipeline, caches, branch predictors, out-of-order
  - See ARM Technical Reference Manual  
[https://static.docs.arm.com/ddi0439/b/DDI0439B\\_cortex\\_m4\\_r0p0\\_trm.pdf](https://static.docs.arm.com/ddi0439/b/DDI0439B_cortex_m4_r0p0_trm.pdf) (Table 3.1)
  - Need empirical measurements

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

29

## Performance Instruments (1)

- Independent counter
  - Use internal SysTick timer

```

unsigned long before, elapsed;
// ranges from 0 to NVIC_ST_RELOAD_R
unsigned long OS_Time(void) {
    return NVIC_ST_CURRENT_R; // 20ns
}
void main(void) {
    before = OS_Time(); // initialize
    // software to test, assume no interrupts
    ...
    elapsed = OS_TimeDiff(OS_Time(), before);
}

```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

30

## Performance Instruments (2)

- Dump with independent counter

```

unsigned long Tbuf[100];
unsigned int Tcnt=0;
void RecordTime(void){
    if(Tcnt==100) // Buffer full?
        return;
    Tbuf[Tcnt] = NVIC_ST_CURRENT_R;
    // 24-bit SysTick counter, 20ns
    Tcnt++;
}

```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

31

## Performance Instruments (3)

- Monitor using output port
  - Measure using scope or logic analyzer
  - Again, atomicity in parallel/interrupt cases
  - What about overhead?

```

void main(void){
    ss = 100;
    while(1){
        GPIO_PORTD_DATA_R |= 0x20;
        tt = sqrt(ss);
        GPIO_PORTD_DATA_R &= ~0x20;
    }
}

```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

32



## Profiling

- Collect the time history of strategic variables
  - Where executing, and when it is executing
  - What is the data, and when is the data these values
- Where executing, when it is executing, and what is the data

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

33

## Profiling Instruments (1)

- Dump into an array

```

unsigned long time[100]; // when
unsigned short place[100]; // where
unsigned short data[100]; // what
unsigned short n = 0;
void profile(unsigned short thePlace, unsigned short theData) {
    if(n==100) return;
    time[n] = OS_Time(); // current time
    place[n]= thePlace;
    data[n] = theData;
    n++;
}
unsigned short sqrt(unsigned short s) {
    unsigned short t,oldt;
    t=0; // secant method
    profile(0,t);
    if(s>0) {
        profile(1,t);
        t=32; // initial guess 2.0
        do{
            profile(2,t);
            oldt=t; // from the last
            t=((t*t+16*s)/t)/2;}
        while(t!=oldt);}
    profile(3,t);
    return t;
}

```

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

34

## Profiling Instruments (2)

- Profile using an output port
  - Output 1,2,3,... or 1,2,4,... (better)

```

unsigned int sqrt(unsigned int s){
    unsigned int t,oldt;
    GPIO_PORTC4 = 0x10;
    t=0; // secant method
    if(s>0) {
        GPIO_PORTC5 = 0x20;
        t=32; // initial guess 2.0
        do{
            GPIO_PORTC6 = 0x40;
            oldt=t; // from the last
            t=((t*t+16*s)/t)/2;
            GPIO_PORTC6 = 0;
        }
        while(t!=oldt);
        GPIO_PORTC5 = 0;
    }
    GPIO_PORTC4 = 0;
    return t;
}

```

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

35

## Profiling Instruments (3)

- Thread profiling using output port
  - When is which thread running?
  - Set bit on enter, clear bit on exit

```

GPIO_PORTC4 = 0x10; // Thread 1
RxFifo_Put(data);
GPIO_PORTC4 = 0;

```

```

GPIO_PORTC5 = 0x20; // Thread 2
TxFifo_Get(&data);
GPIO_PORTC5 = 0;

```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

36

## CPU Bound or I/O Bound?

- Measure FIFO size versus time
  - When is it I/O bound? When is it CPU bound?

```
unsigned short TxFifo_Size(void){
    if(TxPutPt<TxGetPt){
        return(TxPutPt+TXFIFOSIZE-TxGetPt);
    }
    else{
        return(TxPutPt-TxGetPt);
    }
}
```

- Collect a histogram of FIFO sizes

```
unsigned long histogram[TXFIFOSIZE];
void Collect(void){
    histogram[TxFifo_Size()]++;
}
```

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

37

## Debugging Real-Time Systems

- Events that are observable in real time
  - The input and output signals of the system
    - Observe using logic analyzer
  - Dumps
    - Record in real time, observe later off line
  - Extra output pins
    - Heart beats, monitors, profiling (logic analyzer)

Lecture 2

J. Valvano, A. Gerstlauer  
EE445M/EE380L.12

38

## Debugging Style

- Develop your own unique style
  - Place all print statements in a unique column
    - Specific pattern in their names
  - Test a run time global flag
    - Leaves a copy of the code in the final system
    - Simplifies “on-site” customer support
  - Use conditional compilation (`#ifdef DEBUG`)
    - Performance and effectiveness
  - For safety-critical systems leave instruments in