Timing is Everything –
Why Embedded Systems Demand Early Teaching of Structured Time-Oriented Programming

Frank Vahid*
Department of Computer Science and Engineering
University of California, Riverside, USA
*Also with the Center for Embedded Computer Systems, UC Irvine
http://www.cs.ucr.edu/~vahid

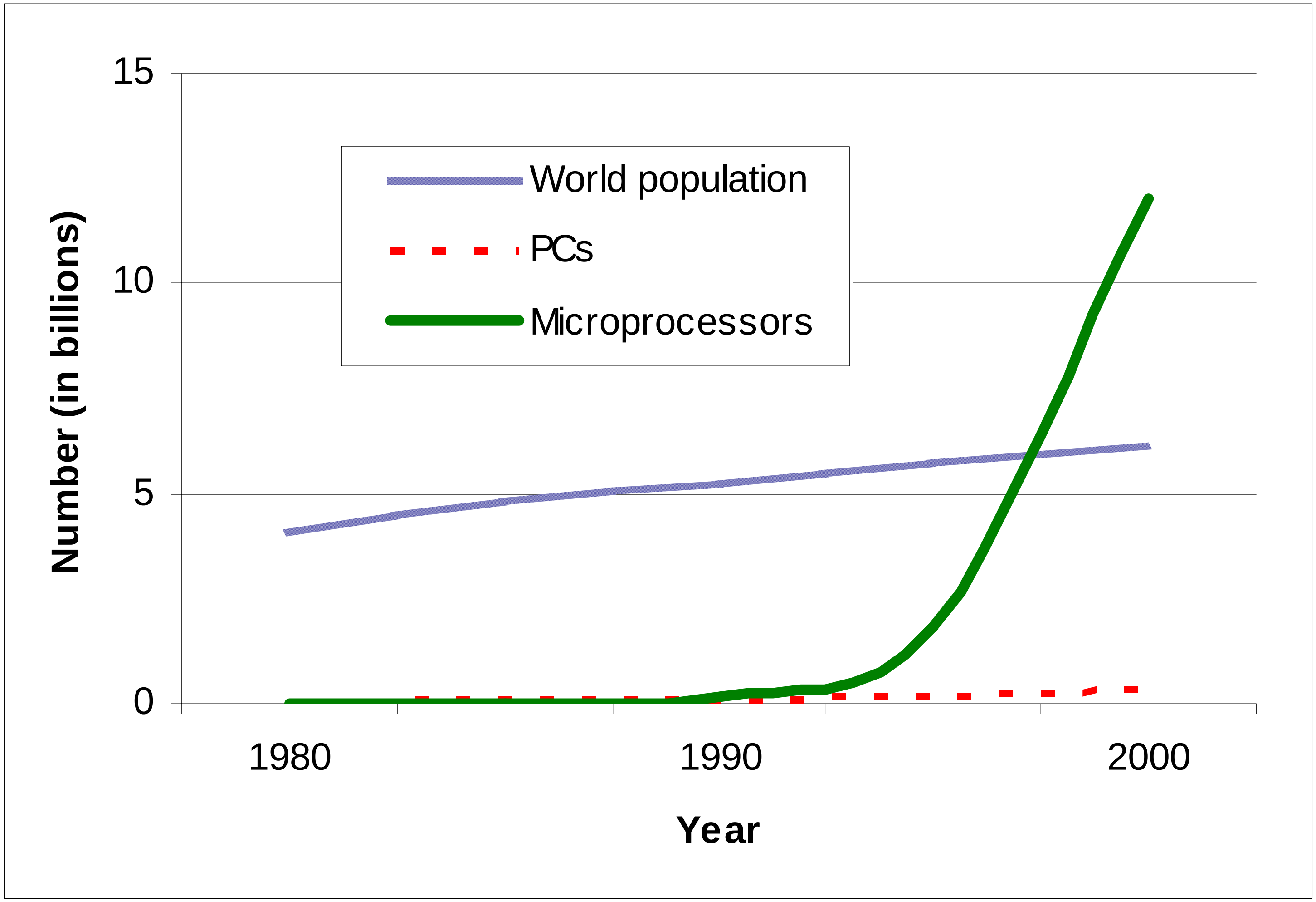
Collaborator: Tony Givargis, UC Irvine

This work was supported in part by the National Science Foundation (CNS-0614957)

Embedded Systems

- Ubiquitous

Embedded Systems

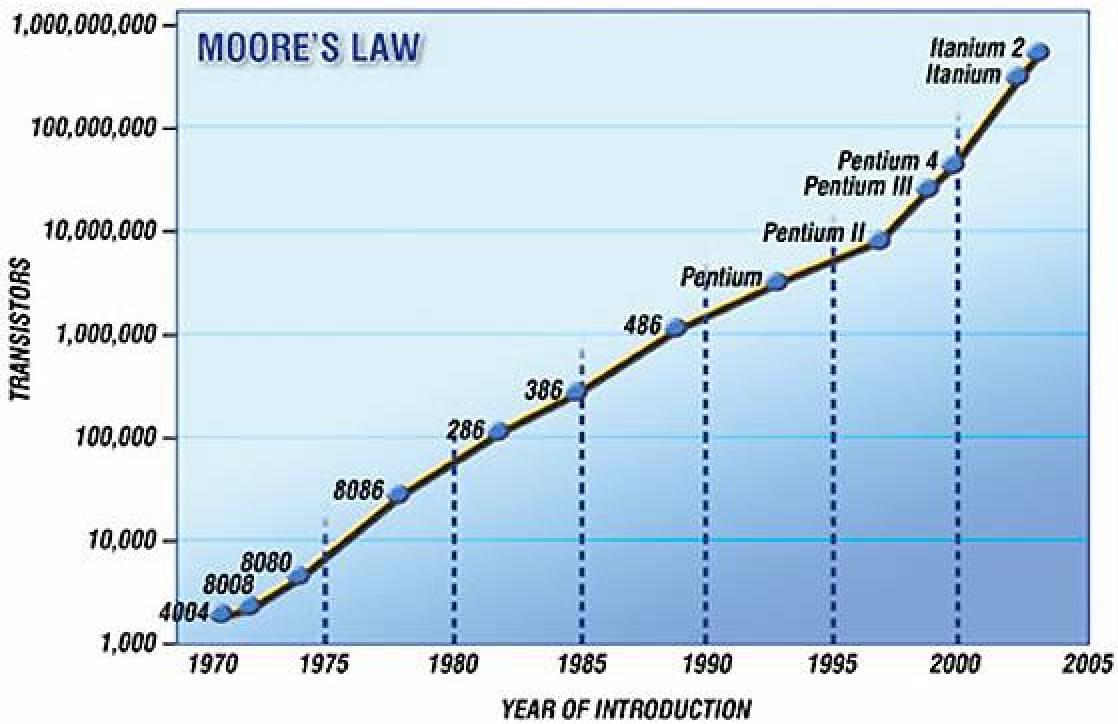


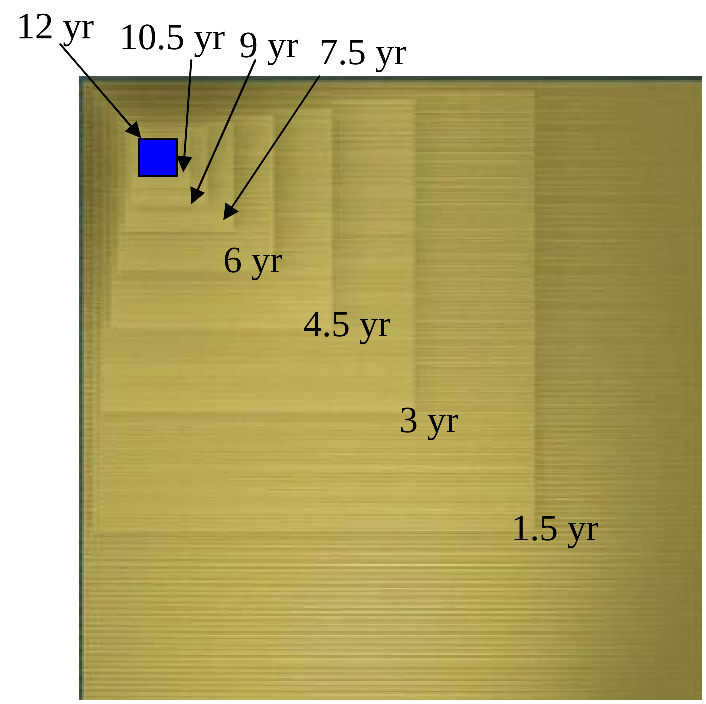
Source: Study of Worldwide trends and R&D programmes in Embedded Systems by FAST GmbH. ftp://ftp.cordis.europa.eu/pub/ist/docs/embedded/final-study-181105_en.pdf

Moore's Law

Corollary – Shrinking

2x / 18 months





Courtesy of Joe Kahn

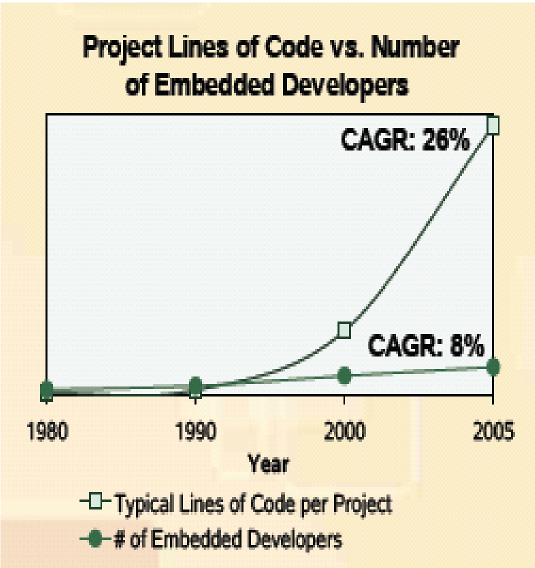
Processors can go where no processor has gone before

Literally...

http://www.templehealth.org

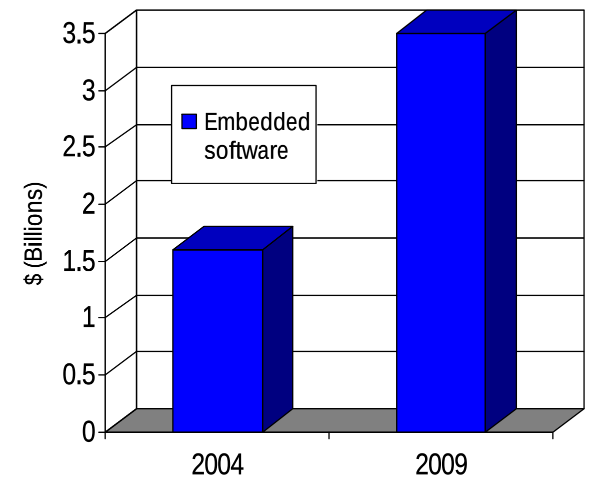
Of the approximately 150,000 U.S. patents granted per year, roughly 10,000-20,000 are embedded systems related

Embedded Systems – Software is Key



Source: Venture Development Corp, 2004

New NSF program for 2009: Cyber-Physical Systems. $30,000,000



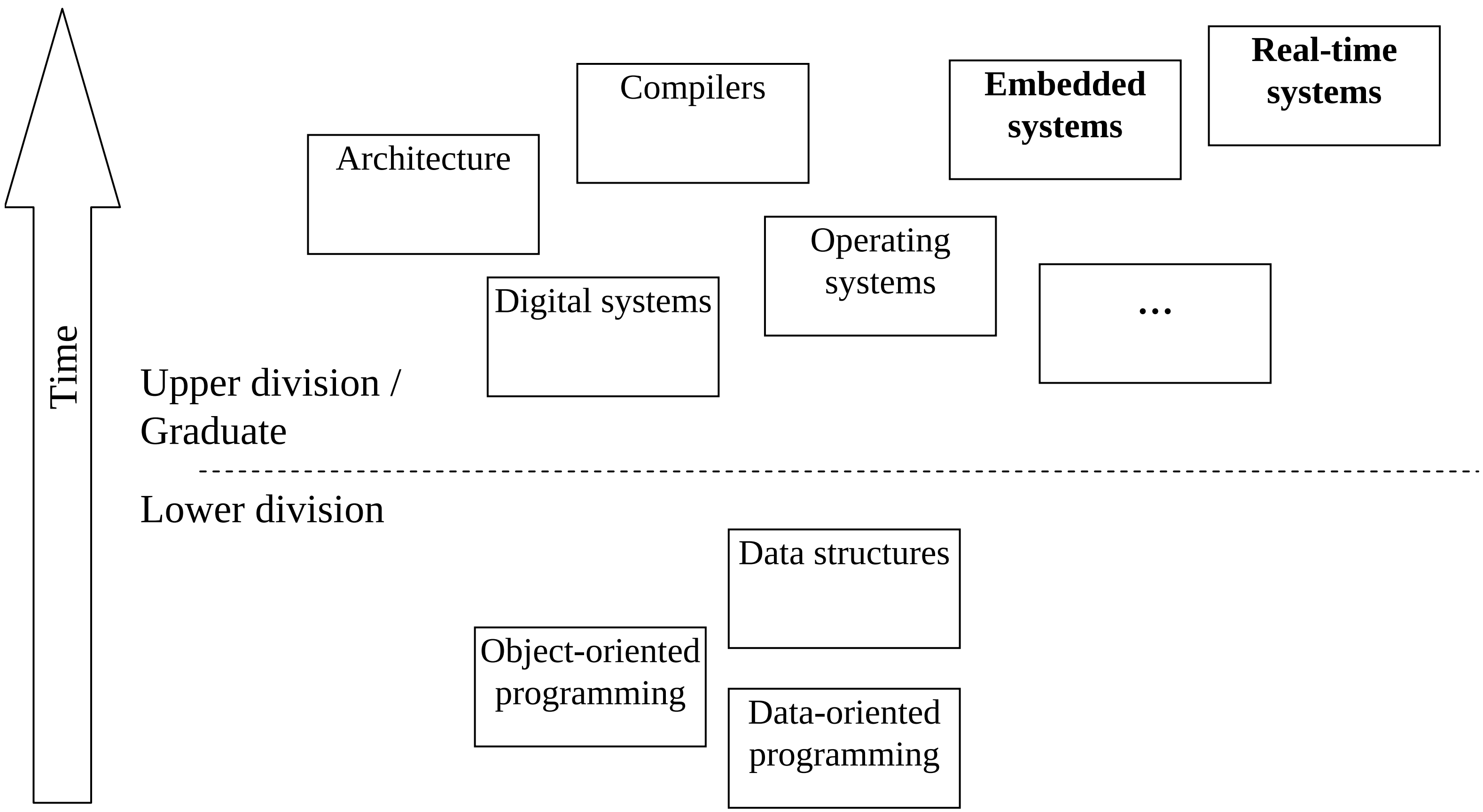
Embedded systems market: 14% annual growth (16% for emb sw) vs. 8% for PC hw and sw

- Software is growing
- Software is hard
- *Are we teaching embedded systems software development properly?*

...All arguments of the study show that requirements in embedded software are more important than those in hardware, and that the market of the future will be largely software driven. In short, **software** is the key to innovation.

Source: Study of Worldwide trends and R&D programmes in Embedded Systems by FAST GmbH. ftp://ftp.cordis.europa.eu/pub/ist/docs/embedded/final-study-181105_en.pdf

Current ES Courses Typically Senior-Level



Current ES Courses – Details, Details, Details

- Very practical courses – Learn components, get them working, build systems
  - Microcontroller, timer, UART, A2D, LCD, buttons, keypads, stepper motors, ...

```
// -----------------------------------------------------------
// configure output ports
// -----------------------------------------------------------
ADCON0 = 0x00;    /* disable A/D converter */
CM1CON0 = 0x00;
CM2CON0 = 0x00;  /* disable comparators */
ANSELH = 0x00;
ANSEL = 0x00;       /* configure pins as digital chanels */
TRISA = 0x08;       /* all bits output except RA3. */
TRISB = 0xF0;       /* Port B inputs*/
RABPU = 1;
WPUB4 = 1;      // enable weak pull ups on RB4
 IOCB4 = 1;      // enable interrupt on change for RB4
TRISC = 0x00;   /* PORTC all set to outputs */
PORTA = 0x00;
PORTB = 0x00;
PORTC = 0x00;   /* initialize ports */
// -----------------------------------------------------------
// Timer0 setup
// -----------------------------------------------------------
CLRWDT();   // turn off watch dog timer
OPTION = 0x07;    // setup prescaler
TMR0 = PRELOAD; // preload timer
T0IE = 1; //enable timer0 interrupts
// -----------------------------------------------------------
// Setup button interrupts
// -----------------------------------------------------------
RABIE = 1; //Enable change on PORTB interrupts
GIE = 1;         //global interupts enabled
```

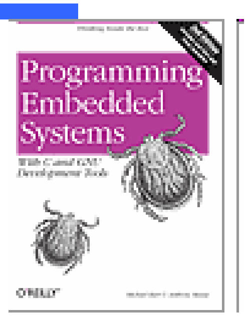
Inside Book #1

Others are similar

Details, Details, Details

Microcontroller Technology, Peter Spasov

Details, Details, Details



1. Introduction What Is an Embedded System? Variations on the Theme C: The Least Common Denominator A Few Words About Hardware.
2. Your First Embedded Program Hello, World! Das Blinkenlights The Role of the Infinite Loop.
3. Compiling, Linking, and Locating The Build Process Compiling Linking Locating Building das Blinkenlights.
4. Downloading and Debugging When in ROM - Remote Debuggers Emulators Simulators and Other Tools.
5. Getting to Know the Hardware Understand the Big Picture Examine the Landscape Learn How to Communicate Get to Know the Processor Study the External Peripherals Initialize the Hardware.
6. Memory Types of Memory Memory Testing Validating Memory Contents Working with Flash Memory.
7. Peripherals Control and Status Registers The Device Driver Philosophy A Simple Timer Driver Das Blinkenlights, Revisited
8. Operating Systems History and Purpose A Decent Embedded Operating System Real-Time Characteristics Selection Process
9. Putting It All Together Application Overview Flashing the LED Printing "Hello, World!" Working with Serial Ports The Zilog 85230 Serial Controller.
10. Optimizing Your Code Increasing Code Efficiency Decreasing Code Size Reducing Memory Usage Limiting the Impact of C++

ACM Curricula

Computing Curricula 2005

The Overview Report

covering undergraduate degree programs in

Computer Engineering
Computer Science
Information Systems
Information Technology
Software Engineering

A volume of the *Computing Curricula Series*

The Joint Task Force for Computing Curricula 2005

A cooperative project of
The Association for Computing Machinery (ACM)
The Association for Information Systems (AIS)
The Computer Society (IEEE-CS)

30 September 2005

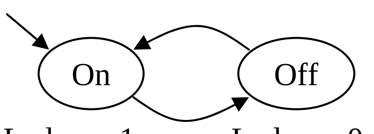
...“Currently, a dominant area within *computing engineering* is **embedded systems**, the development of devices that have software and hardware embedded in them. For example, devices such as cell phones, digital audio players, digital video recorders, alarm systems, x-ray machines, and laser surgical tools all require integration of hardware and embedded software and all are the result of computer engineering.” *(emphasis added)*

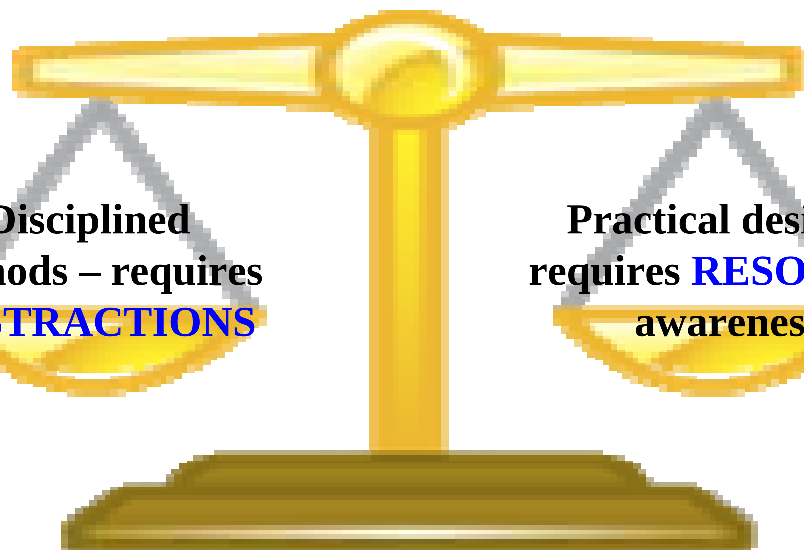
Table 3.1: Comparative weight of computing topics across the fi

Knowledge Area	CE		CS	
	min	max	min	max
Programming Fundamentals	4	4	4	5
Integrative Programming	0	2	1	3
Algorithms and Complexity	2	4	4	5
Computer Architecture and Organization	5	5	2	4
Operating Systems Principles & Design	2	5	3	5
Operating Systems Configuration & Use	2	3	2	4
Net Centric Principles and Design	1	3	2	4
Net Centric Use and configuration	1	2	2	3
Platform technologies	0	1	0	2
Theory of Programming Languages	1	2	3	5
Human-Computer Interaction	2	5	2	4
Graphics and Visualization	1	3	1	5
Intelligent Systems (AI)	1	3	2	5
Information Management (DB) Theory	1	3	2	5
Information Management (DB) Practice	1	2	1	4
Scientific computing (Numerical mthds)	0	2	0	5
Legal / Professional / Ethics / Society	2	5	2	4
Information Systems Development	0	2	0	2
Analysis of Business Requirements	0	1	0	1
E-business	0	0	0	0
Analysis of Technical Requirements	2	5	2	4
Engineering Foundations for SW	1	2	1	2
Engineering Economics for SW	1	3	0	1
Software Modeling and Analysis	1	3	2	3
Software Design	2	4	3	5
Software Verification and Validation	1	3	1	2
Software Evolution (maintenance)	1	3	1	1
Software Process	1	1	1	2
Software Quality	1	2	1	2
Comp Systems Engineering	5	5	1	2
Digital logic	5	5	2	3
Embedded Systems	2	5	0	3
Distributed Systems	3	5	1	3
Security: issues and principles	2	3	1	4
Security: implementation and mgt	1	2	1	3
Systems administration	1	2	1	1
Management of Info Systems Org.	0	0	0	0
Systems integration	1	4	1	2
Digital media development	0	2	0	1
Technical support	0	1	0	1

No mention of embedded systems under computer science topic.

ES Education Goals

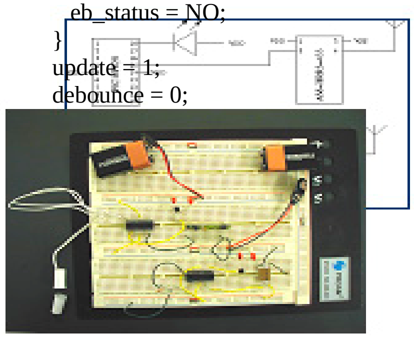




Disciplined methods – requires ABSTRACTIONS

Practical design – requires RESOURCE awareness

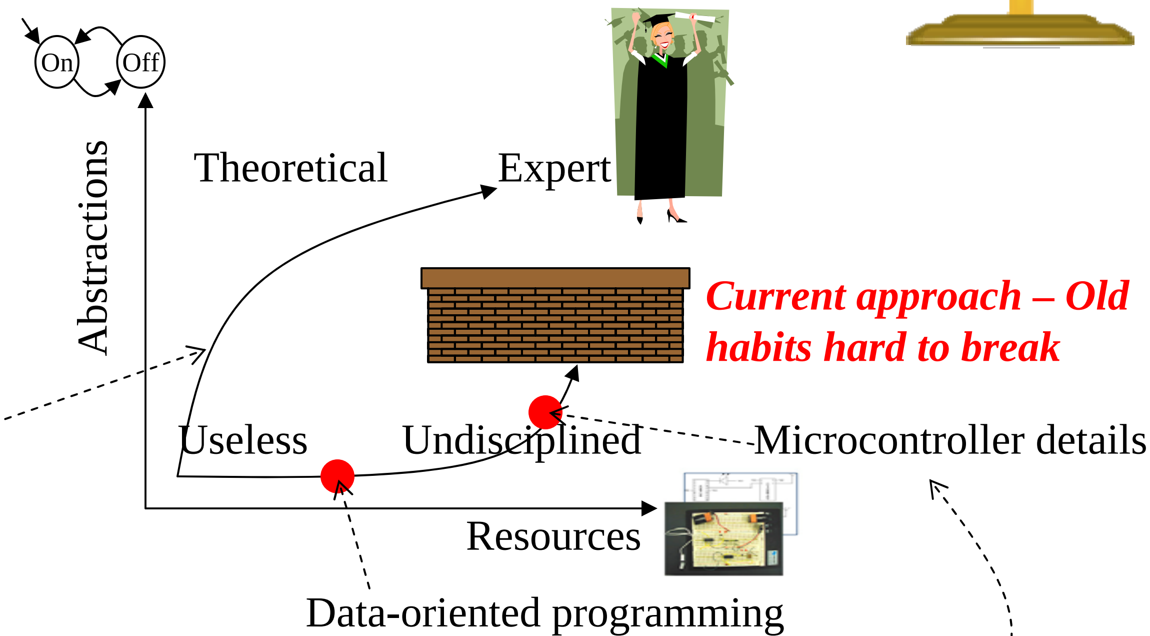
```
for(;;){
if (debounce == 1)
{
  while (debounce_done == 0 );
  if (data_val == 1) {
    eb_status = YES;

  }
  else {
    eb_status = NO;
  }
  update = 1;
  debounce = 0;
```

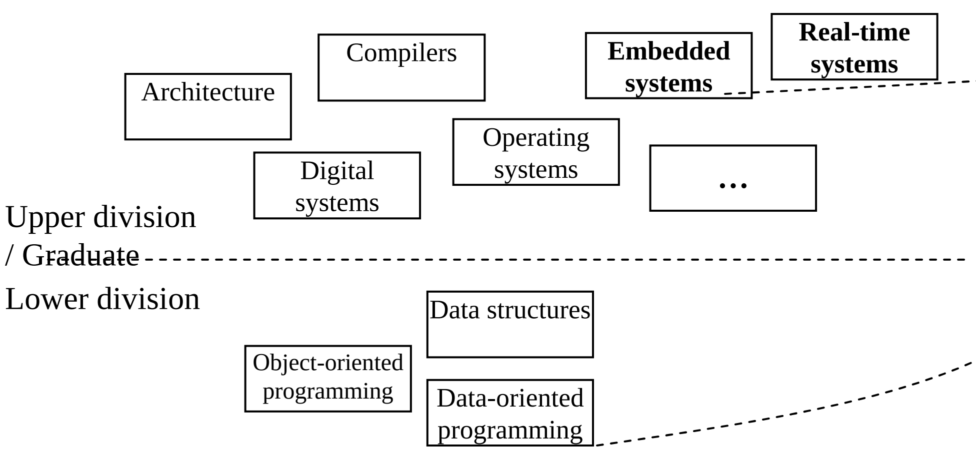


Need to find right order and balance

ES Education Goals

"Technique"





Scientific American, April 28, 2008
In Abstract: Avoid Concrete Examples When Teaching Math
New study indicates that extraneous information in word problems may cover up mathematical concepts
...
The results: students who learned using symbols on average scored 80 percent; the others scored between 40 and 50 percent

Leads to Less-Than-Ideal Coders

- Higher-level discipline lacking, leading to
  - Unstructured code
  - Error prone
  - Hard to maintain
  - Not formally verifiable
  - Job security for original program author


```
for(;;){
if (debounce == 1)
{
   while (debounce_done == 0 );
   if (data_val == 1) {
      eb_status = YES;
   }
   else {
      eb_status = NO;
   }
   update = 1;
   debounce = 0;
}
if( update == 1) {
  //clear update flag
  update = 0;
  led_status();
  sci_SendAlive();
  sci_PutByte(BOOL | eb_status);
  goto_sleep = 0;
  sleep_cnt = 0;
  tx_cnt = 0;
}
else if (send_alive == 1) {
  sci_SendAlive();
  send_alive = 0;
}

if (goto_sleep == 1) {
      led_off();
      while(!TXIF); //wait to finish send
      sleep();
}
```

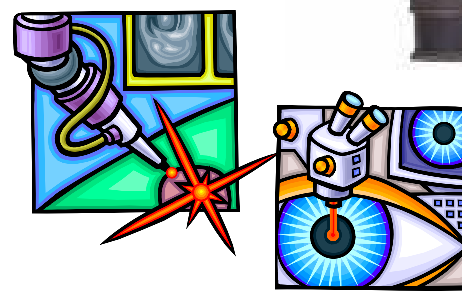
Ouch

Commercial code I helped debug, summer 2008

```
if(Flags.Bit.Uart_B == 1)  //Set if interrupt occurs
{
             INTCONbits.GIEH = 0;//Disable Global Ints
             Flags.Bit.Uart_B = 0;
             INTCONbits.GIEH = 1;
             uart_chk_intB();
             check_Rx_B();
}
/*** Check Uart B Tx (GMS Transmitter) Interrupt ***/
if(Flags.Bit.Uart_B_Command)             {
             if(uart_INTB==0)
             {
                          Flags.Bit.Uart_B_Command = 0;
                          for(loop = 0; loop < tx_byte_count; loop++)
                          {
                                       uart_write_Trans(msg_Tx_Out[0]);
                          }
             }
}
/*           One msec Timer              */
if (Flags.Bit.Timeout == 1)
//1 msec interrupt
{
             Flags.Bit.Timeout = 0;
             one_msec_counter++;

/*           Ten msec Timer              */
             if(one_msec_counter == 10)
             {
                          one_msec_counter = 0;
                          ten_msec_counter++;
                          two_hund_msec_counter++;
                          if(two_hund_msec_counter == 20)
                          {
                                       two_hund_msec_counter  = 0;
                          }
/*           One Second Timer            */
                          if (ten_msec_counter == 100)
                          {
                                       ten_msec_counter = 0;
                                       one_sec_counter++;

...
```

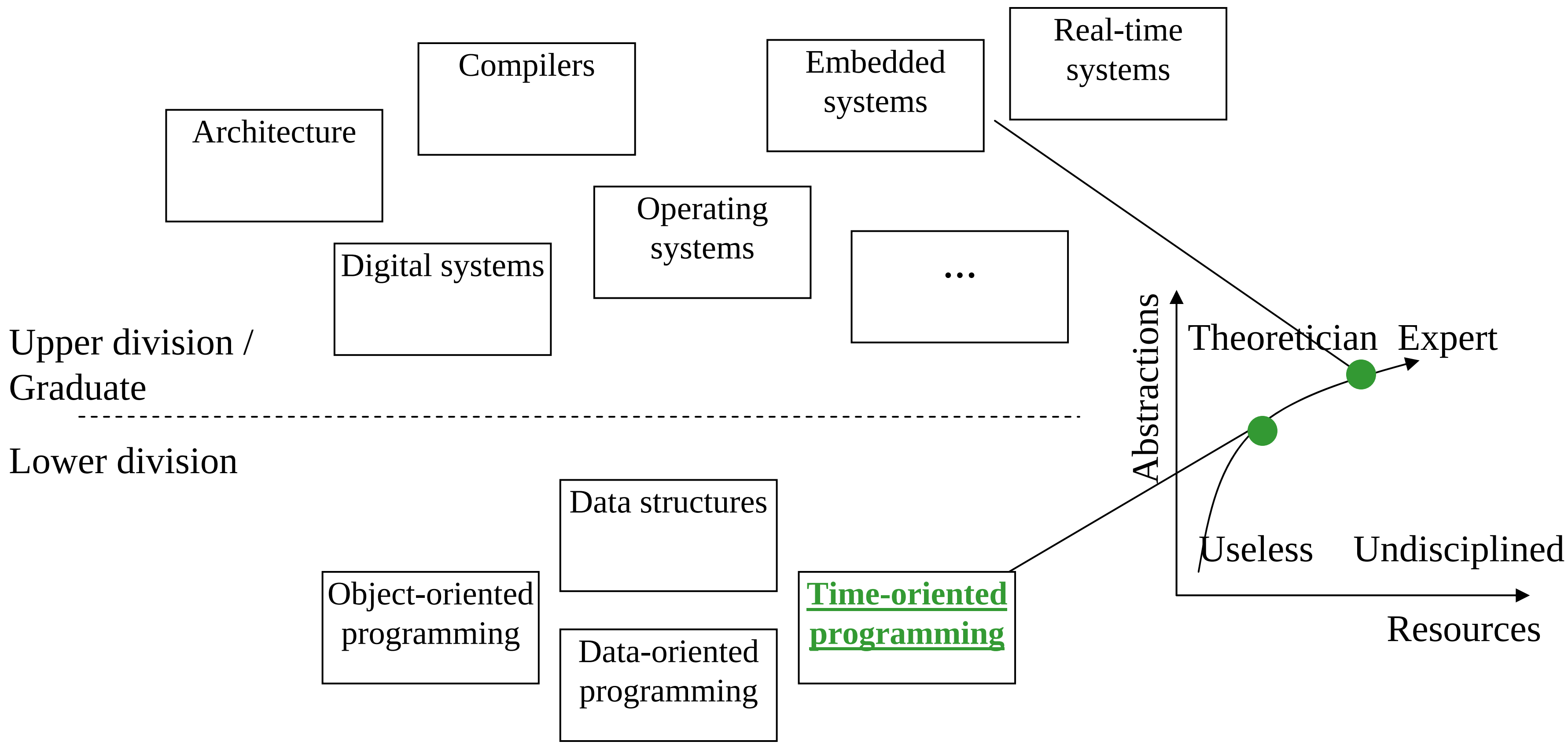
Towards a Higher-Level ES Programming Discipline – What Abstractions?

- Basic ES examples:
  - Blinking tennis shoe
    - When vibration detected, blink lights for 500 ms in particular pattern
  - Motion sensing light
    - Detect motion of at least ½ sec duration, turn on lamp for 10 sec
  - Laser eye surgery device
    - When button pressed, turn on laser for 300 ms
- Feature*: Timed behaviors*
  - Detect input events or set output events for specific time durations
  - Desktop computing focuses instead on transforming data (read file, transform, write)



“`If precise timeliness in a networked embedded system is absolutely essential, what has to change?’ The answer, unfortunately, is `nearly everything.’” Ed Lee, TR 2005-05, UCB.

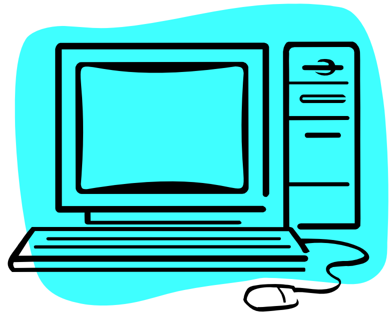
Time-Oriented Programming a Fundamental Topic

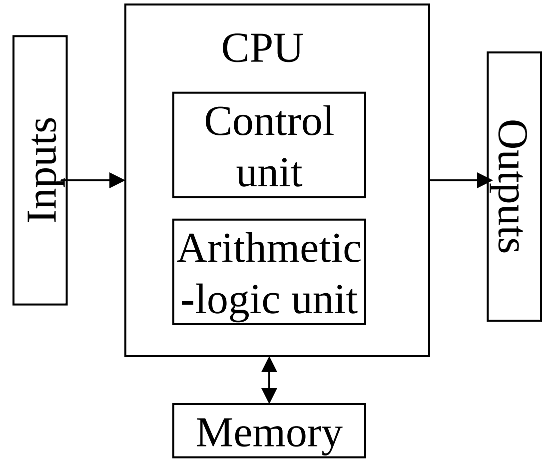


Time-Oriented Programming

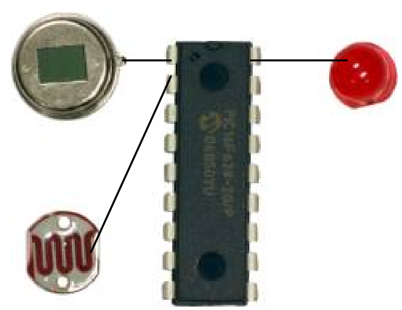
- Requires
  - Clean hardware abstraction (avoid swimming in details)
  - Good computation models (it’s not about C)

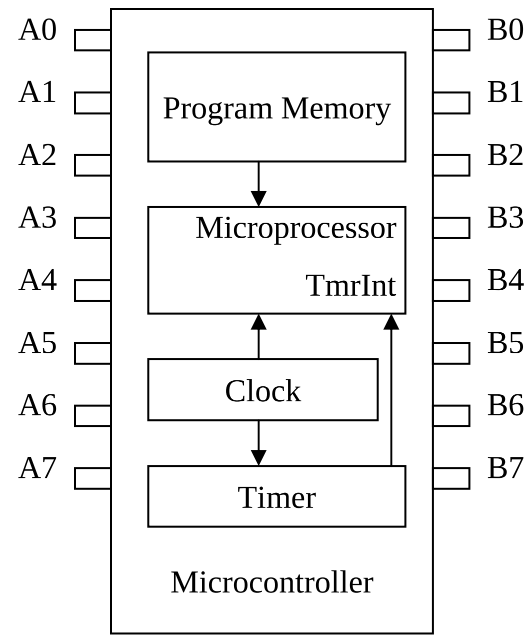
Time-Oriented Programming – Abstraction 1: "Clean" Microcontroller





- Desktop prog. – Starts clean
  - No OS install, hw setup, etc. -- Intentionally simplified
- Embedded prog. – Start clean
  - No complicated configuration of pins, timers, clock, etc.
  - No electrical issues like high impedance, pull up resistors, etc.
  - Hard for instructors to resist sharing expert detail knowledge
- e.g., "Virtual Microcontroller" (Sirowy et al WESE'08)

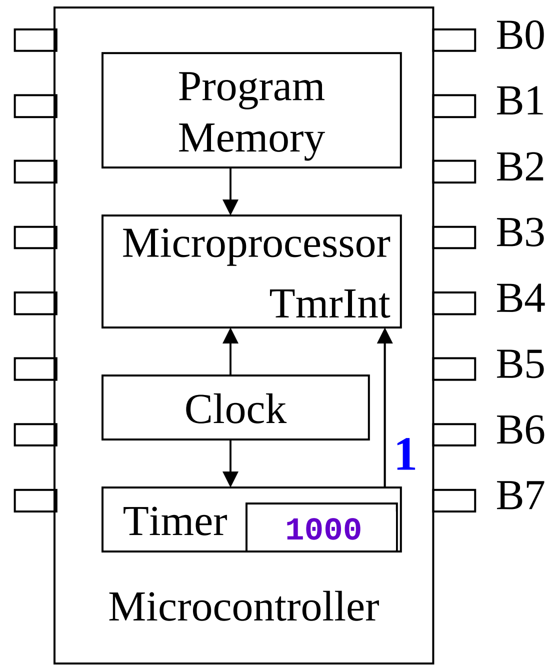




Clean Microcontroller – Idealized Timer and ISR

- Expose key resources, simplified

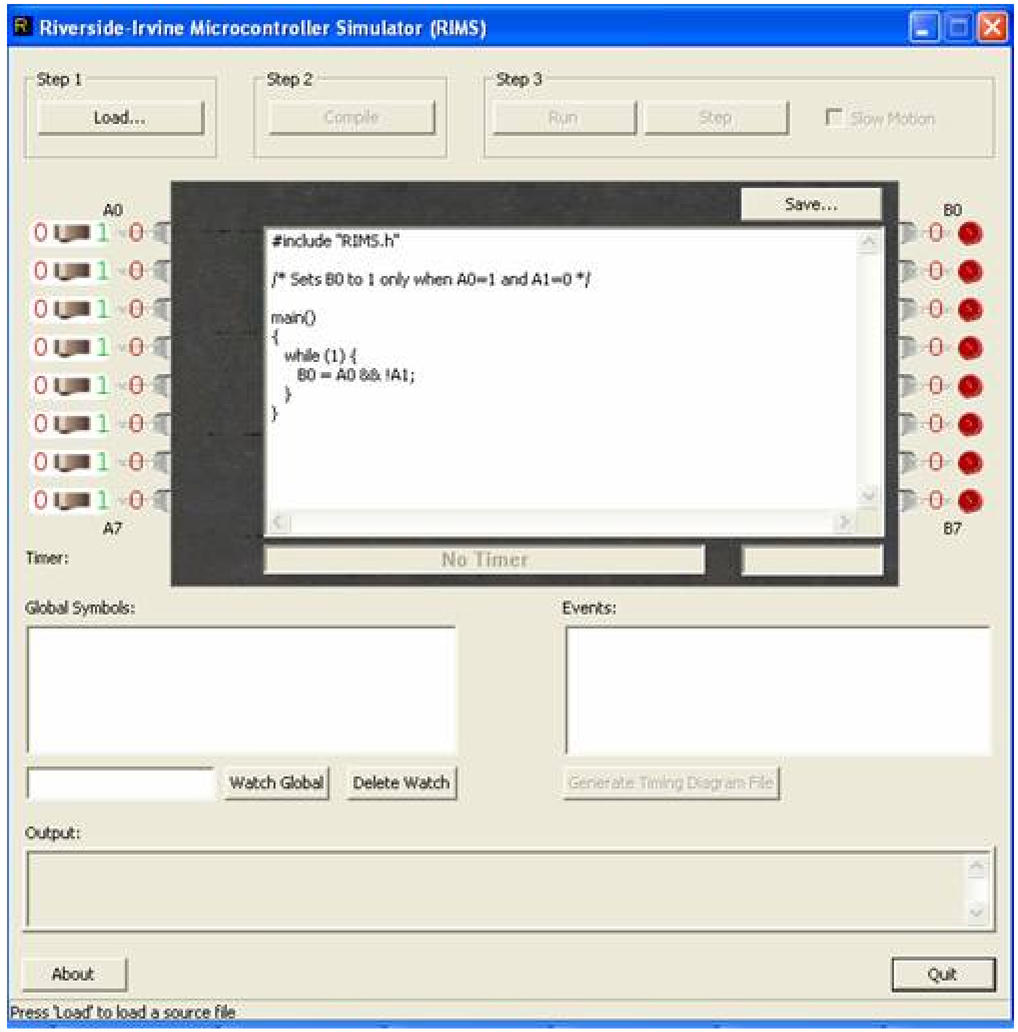
```
#define Door1_i  A0
#define Door2_i  A1
#define Avail_o  B0
#define Occup_o  B1

void TimerISR()
{
}

void main(void)
{
   B0=B1=B2=B3=B4=B5=B6=B7=0;
   TimerSet(1000); TimerOn();
   while (1) {
      Avail_o = !Door1_i || !Door2_i;
      Occup_o = Door1_i && Door2_i;
   }
}
```



Note: Flashing speed not to scale

Virtual Microcontroller Environment/Simulator

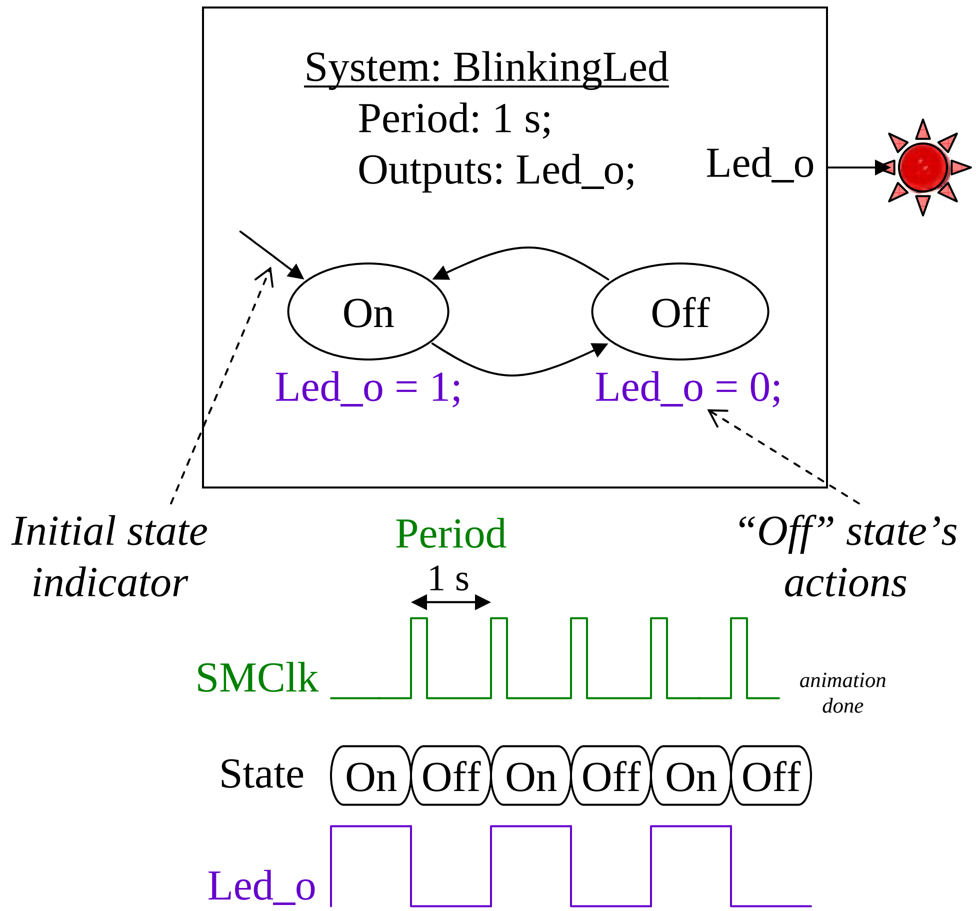
- Single environment for
  - Code
  - Compile
  - Simulate
  - Debug
  - Waveforms



Time-Oriented Programming – Abstraction 2: Computation Model

- Sequential program model (C) unwieldy for time/event behavior
- Synchronous State Machines (synchSM) –
  - *Timed and event behavior*
  - Actions
  - Transitions each clock tick
    - Period – basis of timed behavior
- Blinking LED example
  - The “Hello World” of ES

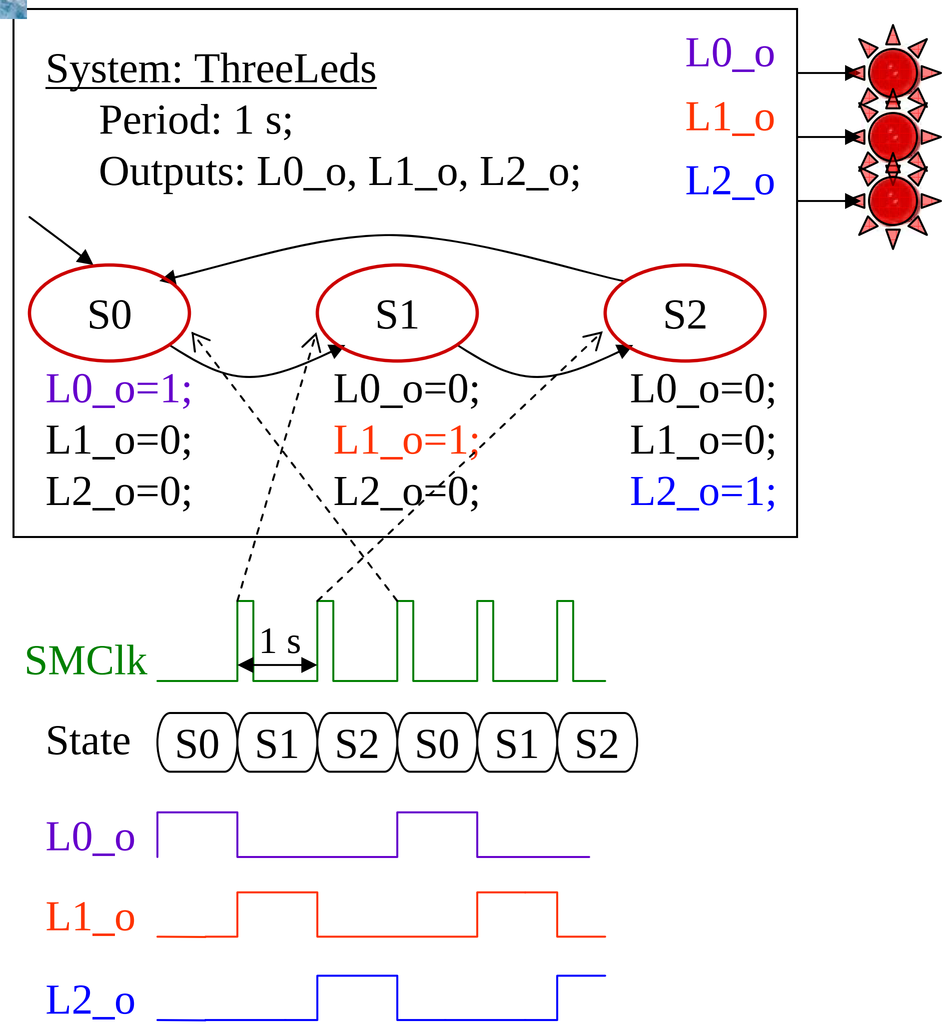
```
... // Blinking LED
   while (1) {
      Led_o = 1;
      while (!TimerFlag); // wait 1 s
      TimerFlag = 0;
      Led_o = 0;
      while (!TimerFlag); // wait 1 s
      TimerFlag = 0;
   }
```



Related abstraction: Timing diagrams

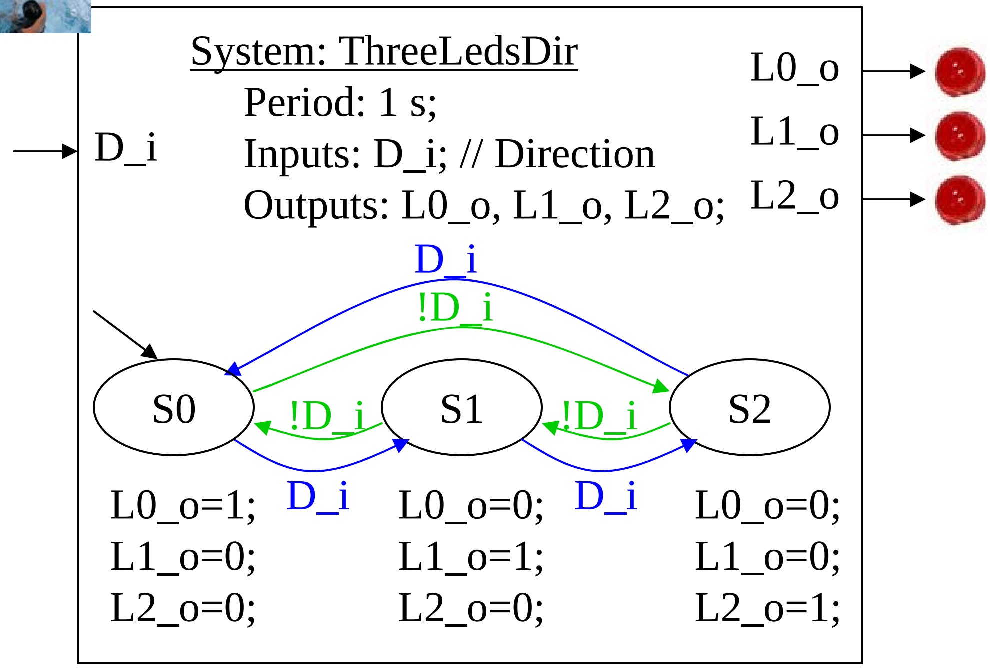
Swim in SynchSM Abstraction for a While

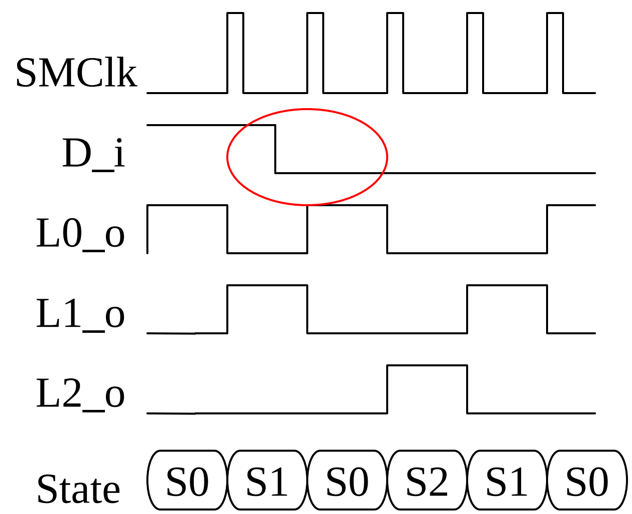
```
... // Three LEDs
   while (1) {
      L0_o=1; L1_o=0; L2_o=0;
      while (!TimerFlag);
      TimerFlag = 0;
      L0_o=0; L1_o=1; L2_o=0;
      while (!TimerFlag);
      TimerFlag = 0;
      L0_o=0; L1_o=0; L2_o=1;
      while (!TimerFlag);
      TimerFlag = 0;
   }
```



Keep Swimming...

D_i = 1 – top to bottom sequence
D_i = 0 – bottom to top sequence

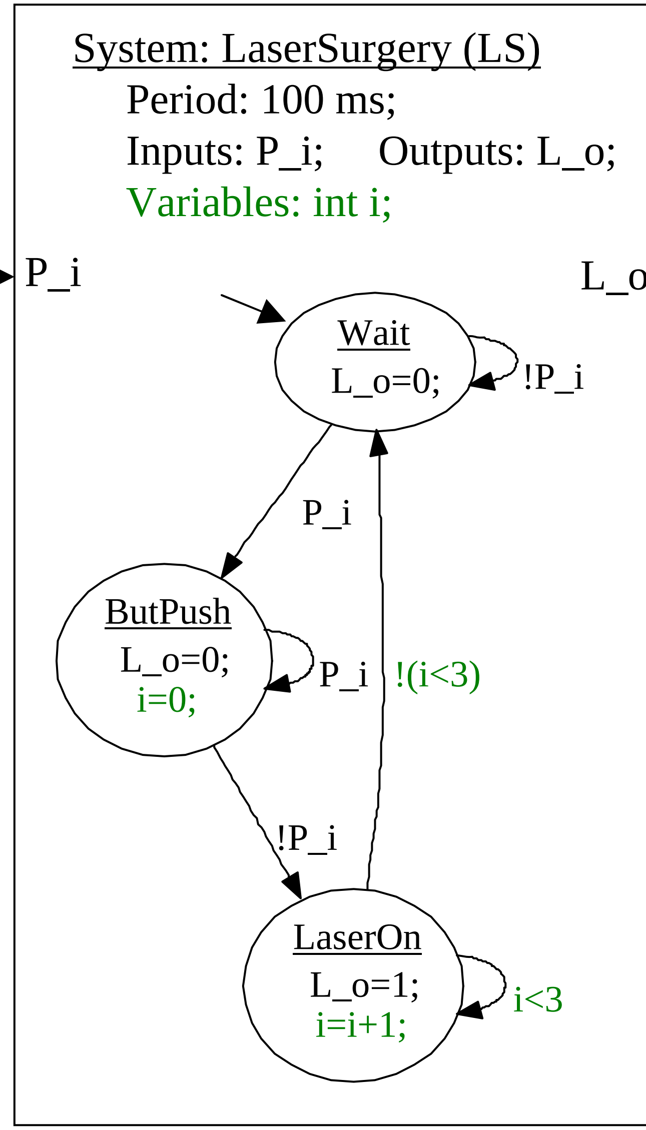


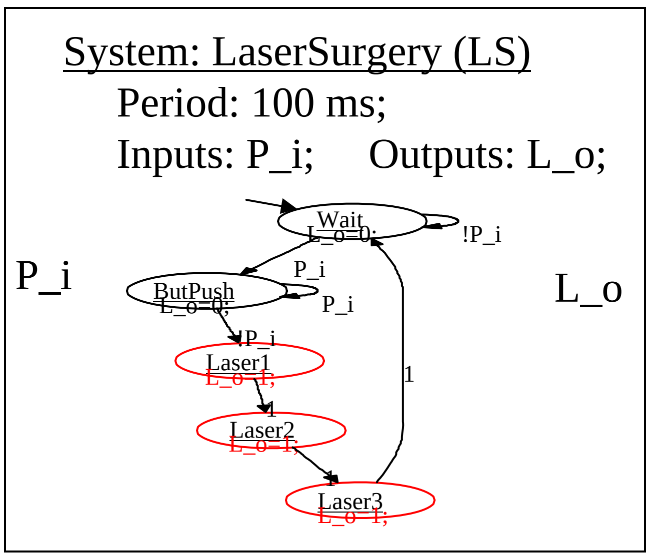


Frank Vahid, Univ. of California, Riverside

And Swimming...

- Add variables

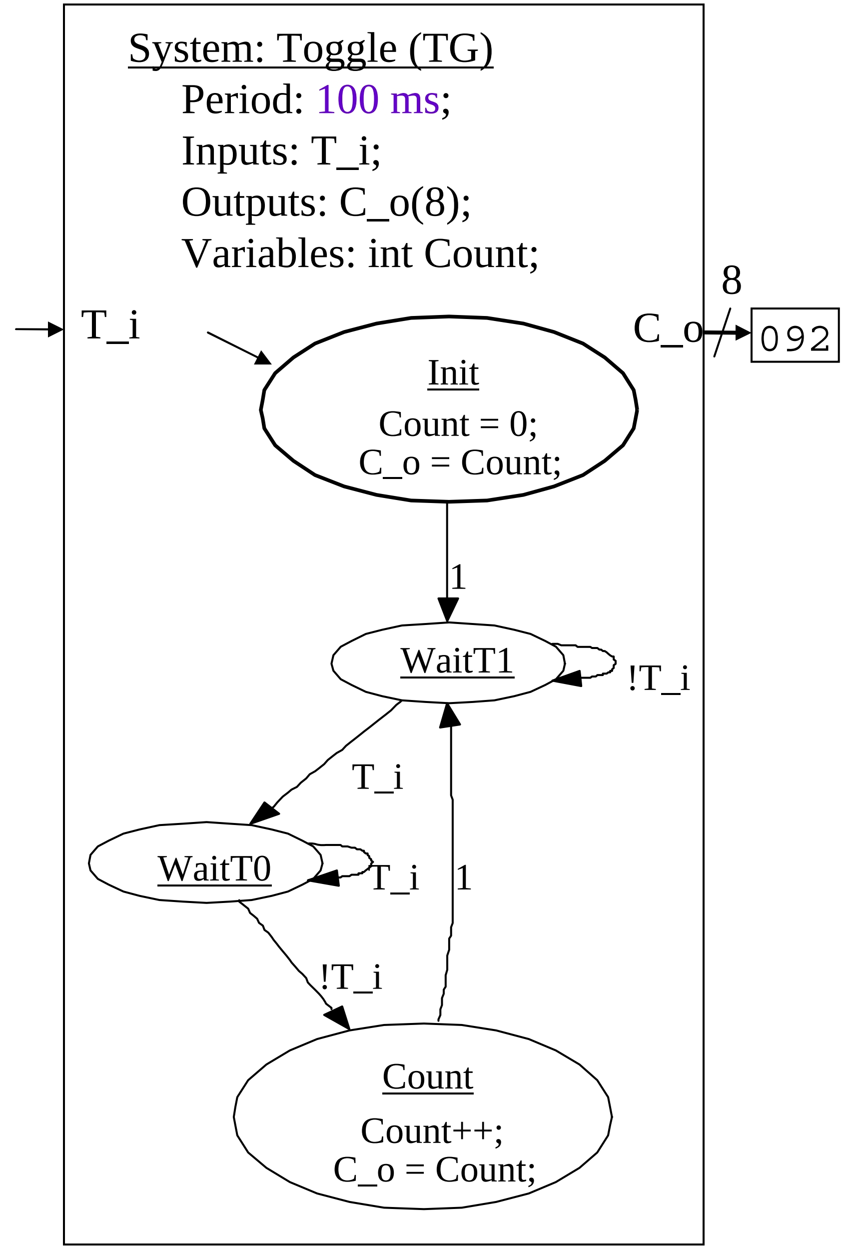




Previous synchSM

And Swimming.

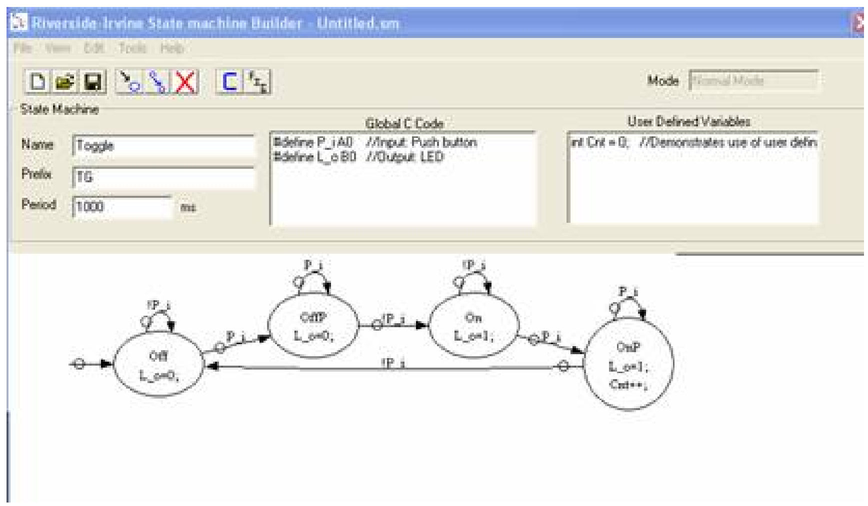
- Add C statements
  - If, for, switch, ...
  - Expressivity on par with C



from pes_ch3_TurnstileCtr.sm

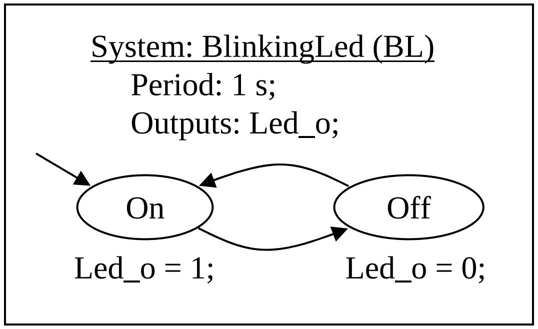
Capture/Simulator Tool Needed

- SM simulators available in various tools, but not particularly accessible
- Solution – Easy-to-use synchSM simulator



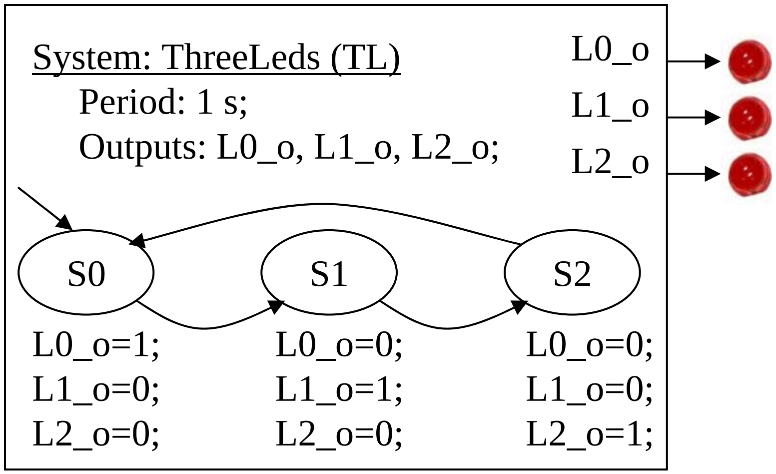
But We Have to be Practical

- Implement model in C
- Standard template
  - ***No creativity involved***
  - Obvious via automatic generation from synchSM too




```
#define Led_o B0

int BL_Clk=0;
void TimerISR()
{
   BL_Clk = 1;
}

void main(void)
{
   enum BL_StateType {BL_On, BL_Off} BL_State;
   B0=B1=B2=B3=B4=B5=B6=B7=0;// Init outputs
   TimerSet(1000); // 1 second
   TimerOn();

   BL_State = BL_On;
   while (1) {
      switch (BL_State) { // State actions
         case BL_On:
            Led_o = 1;
            break;
         case BL_Off:
            Led_o = 0;
            break;
      } // State actions

      while (!BL_Clk); BL_Clk = 0;

      switch (BL_State) { // Transitions
         case BL_On:
            BL_State = BL_Off;
            break;
         case BL_Off:
            BL_State = BL_On;
            break;
      } // Transitions
   } // while(1)
} // main
```

Train Students in this “Mindless” Translation



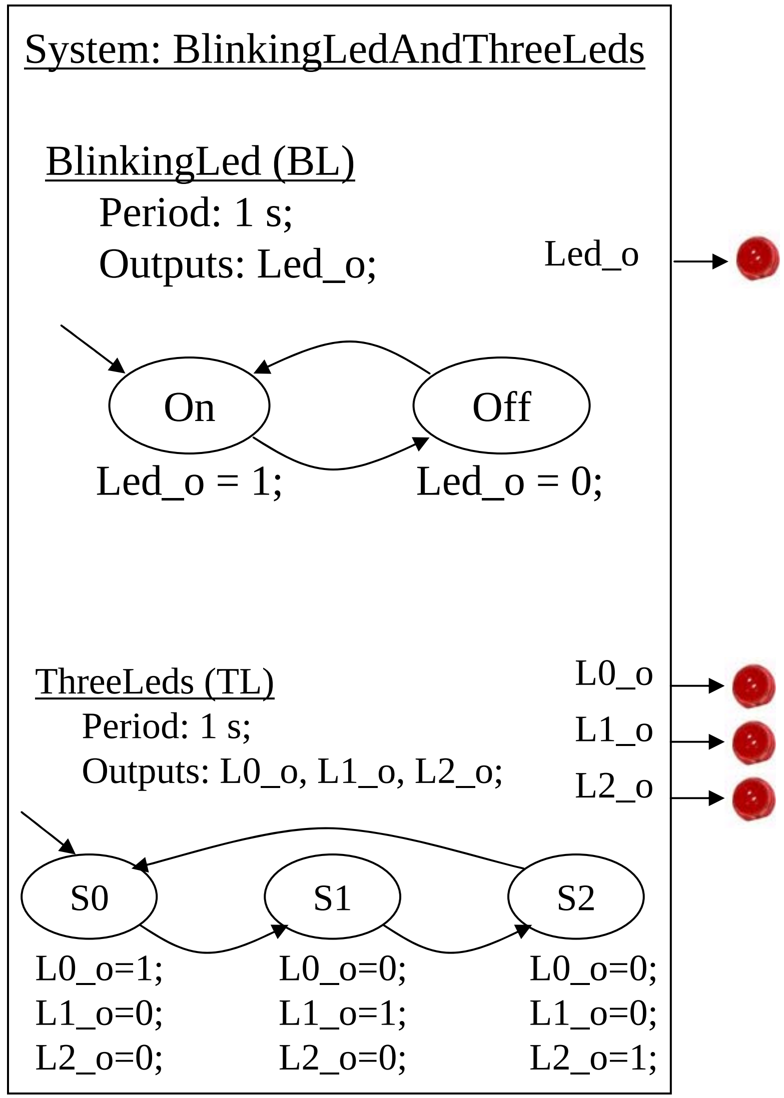

```
#define L0_o B0
#define L1_o B1
#define L2_o B2

int TL_Clk = 0;

void TimerISR(){
   TL_Clk = 1;
}
...
```

State actions / Transitions

```
void main() {
   enum states {TL_S0, TL_S1, TL_S2} TL_State;
   B0=B1=B2=B3=B4=B5=B6=B7=0; // Init outputs

   TimerSet(1000); // 1 second
   TimerOn();

   TL_State = TL_S0; // Initial state
   while(1) {
      switch(TL_State) { // State actions
         case TL_S0:
            L0_o=1; L1_o=0; L2_o=0;
            break;
         case TL_S1:
            L0_o=0; L1_o=1; L2_o=0;
            break;
         case TL_S2:
            L0_o=0; L1_o=0; L2_o=1;
            break;
      } // State actions

      while( !TL_Clk ); TL_Clk = 0;

      switch(TL_State) { // Transitions
         case TL_S0:
            TL_State = TL_S1;
            break;
         case TL_S1:
            TL_State = TL_S2;
            break;
         case TL_S2:
            TL_State = TL_S0;
            break;
         default:
            TL_State = TL_S0;
      } // Transitions
   } // while(1)
} // main
```

Multiple Behaviors

- Capture TWO synchSMs
  - Execute concurrently
- Communication methods
- ...



Again, Use C Template

Thus, SynchSM's processed in a round-robin manner

- Do one SM's actions, then the other SM's actions
- Wait on clock tick (assuming same period, so use one clock)
- Do one SM's transitions, then the other SM's transitions

```
#define Led_o B0

#define L0_o B1
#define L1_o B2
#define L2_o B3

int BL_TL_Clk = 0;
void TimerISR(){
   BL_TL_Clk = 1;
}
```

```
void main() {
   enum BL_states {BL_On, BL_Off} BL_State;
   enum TL_states {TL_S0, TL_S1, TL_S2} TL_State;
   B0=B1=B2=B3=B4=B5=B6=B7=0; // Init outputs
   TimerSet(1000); // 1 second
   TimerOn();
   BL_State = BL_On; // Initial state
   TL_State = TL_S0; // Initial state
   while(1) {
      // State actions
      switch (BL_State) {
         case BL_On:
            Led_o = 1;
            break;
         case BL_Off:
            Led_o = 0;
            break;
      }
      switch(TL_State) {
         case TL_S0:
            L0_o=1; L1_o=0; L2_o=0;
            break;
         case TL_S1:
            L0_o=0; L1_o=1; L2_o=0;
            break;
         case TL_S2:
            L0_o=0; L1_o=0; L2_o=1;
            break;
      } // State actions
```

```
      while( !BL_TL_Clk );
      BL_TL_Clk = 0;

      // Transitions
      switch (BL_State) {
         case BL_On:
            BL_State = BL_Off;
            break;
         case BL_Off:
            BL_State = BL_On;
            break;
      }
      switch(TL_State) {
         case TL_S0:
            TL_State = TL_S1;
            break;
         case TL_S1:
            TL_State = TL_S2;
            break;
         case TL_S2:
            TL_State = TL_S0;
            break;
         default:
            TL_State = TL_S0;
      } // Transitions
   } // while(1)
} // main
```

from pes_ch4_BlinkingLedAndThreeLeds.c

Leads Naturally to Parallel Programming and RTOS Concepts

- "Natural" behavior decomposition
  - Synch concurrency easier to initially learn than asynch
- Communication methods among SMs
- Multitasking
  - Diff periods, priorities
- C coding with schedulable blocks
- Task periods, deadlines, etc
- synchSM serves as perspective for parallel and real-time concepts
  - Better RTOS understanding and usage
  - Structured code in absence of RTOS
    - Like surgeon who learns with scalpel

- By the way, also leads naturally to RTL design
  - FPGAs

Other Models and Approaches

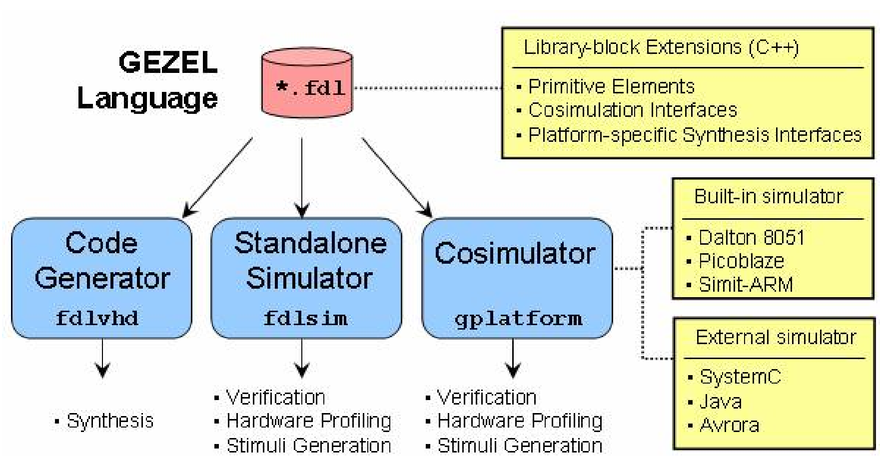
- Synchronous dataflow
- Kahn process networks
- Esterel, Statecharts

Polis CFSMs

UML Statecharts

Statecharts XML (SCXML)

...



Peter Marwedel's Intro ES course (prereqs: programming, SMs):

- Introduction
- Models of computation

3 FSM+shared memory: StateCharts
4 FSM+message passing: SDL; Petrinets
5 Data flow: Kahn process networks, SDF, Labview
6 Imperative programming: Java, ADA, CSP, MPI
7 Discrete event models: VHDL, SystemC
8 ...

Summary

- Early time-oriented programming becoming essential
  - Sequential habits are hard to break
  - Clean microcontroller good starting point
  - SynchSM one good computation model
    - Explicit time management
    - Basis for RTOS concepts
    - Models in C
- Introducing time-oriented programming courses EARLY in computing curricula will be a challenge
  - (OO precedent – “Objects first”)
  - A “time first” approach