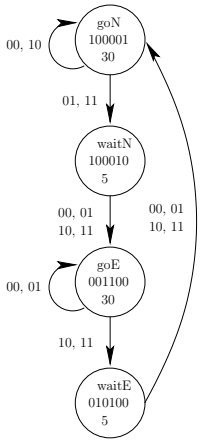
Lecture 4 Supplemental Material

Scott R. Little

January 25, 2007
ECE 5780/6780

Moore FSM & State Table



	No cars	Car E	Car N	Car N,E
goN	goN	waitN	goN	waitN
waitN	goE	goE	goE	goE
goE	goE	goE	waitE	waitE
waitE	goN	goN	goN	goN

Assembly Implementation of Traffic Light Controller

```
      org  $800
OUT   equ  0 ;offset for output
WAIT  equ  1 ;offset for time (8 bits+OUT)
NEXT  equ  3 ;offset for next state (16 bits+WAIT)
goN   fcb  $21 ;East red, north green
      fdb  3000 ;30 second delay
      fdb  goN,waitN,goN,waitN
waitN fcb  $22 ;East red, north yellow
      fdb  500 ;5 second delay
      fdb  goE,goE,goE,goE
goE   fcb  $0C ;East green, north red
      fdb  3000 ;30 second delay
      fdb  goE,goE,waitE,waitE
waitE fcb  $14 ;East yellow, north red
      fdb  500 ;5 second delay
      fdb  goN,goN,goN,goN
```


Assembly Implementation of Traffic Light Controller

```
Main  lds  #$4000 ;stack init
      bsr  Timer_Init ;enable TCNT
      movb #$FF,DDRB ;PORTB5-0 set to output to lights
      movb #$00,DDRA ;PORTA1-0 set to input from sensors
      ldx  #goN ;Initialize state pointer (register X)
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03 ;Keep the bottom two bits
      lslb ;Multiply by two b/c addresses are 2 bytes
      abx  ;add 0,2,4,6
      ldx  NEXT,x
      bra  FSM
```


Memory Map

```
      org  $0800
OUT   equ  0
WAIT  equ  1
NEXT  equ  3
goN   fcb  $21
      fdb  3000
      fdb  goN,waitN,
           goN,waitN
waitN fcb  $22
      fdb  500
      fdb  goE,goE,
           goE,goE
goE   fcb  $0C
```

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	XX XX
RegY	XX XX
AccB	XX

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	XX XX
AccB	XX

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x ;0800+0
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	XX XX
AccB	21

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x ;0800+1
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	21

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	81

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	01

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 00
RegY	0B B8
AccB	02

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x
      bra  FSM
```

RegX	08 02
RegY	0B B8
AccB	02

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Code Execution

```
      ldx  #goN
FSM   ldab OUT,x
      stab PORTB
      ldy  WAIT,x
      bsr  Timer_Wait10ms
      ldab PORTA
      andb #$03
      lslb
      abx
      ldx  NEXT,x ;0802+3
      bra  FSM
```

RegX	08 0B
RegY	0B B8
AccB	02

State	Address	Value	Comment
goN	0800	21	out
	0801	0B B8	wait
	0803	08 00	ns0
	0805	08 0B	ns1
	0807	08 00	ns2
	0809	08 0B	ns3
waitN	080B	22	out
	080C	01 F4	wait
	080E	08 16	ns0
	0810	08 16	ns1
	0812	08 16	ns2
	0814	08 16	ns3
goE	0816	0C	out

Mealy FSM Example

- Similar to Moore FSM except that the output depends on both input and current state.
- This results in the two "tables" in the assembly code.
- Both the output value and next state value must be looked up for a given input.

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	XXXX
0802	XXXX
0804	XXXX
0806	XXXX

SP	0806
RegX	FFFF
AccD	XXXX

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	XXXX
0802	XXXX
0804	FFFF
0806	XXXX

SP	0804
RegX	FFFF
AccD	XXXX

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	XXXX
0802	XXXX
0804	FFFF
0806	XXXX

SP	0804
RegX	0804
AccD	XXXX

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	XXXX
0802	XXXX
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	XXXX

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x ;0804-4
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0000
0802	XXXX
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	XXXX

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x ;0804-2
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0000
0802	0064
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	XXXX

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x ;0804-2
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0000
0802	0064
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0064

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x ;0804-4
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0000
0802	0064
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0064

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x ;0804-4
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0064
0802	0064
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0064

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x ;0804-2
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0064
0802	0064
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0064

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	0064
0802	0064
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0063

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x ;0804-2
      bne  loop
      txs
      pulx
```

0800	0064
0802	0063
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0063

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	13BA
0802	0000
0804	FFFF
0806	XXXX

SP	0800
RegX	0804
AccD	0000

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	13BA
0802	0000
0804	FFFF
0806	XXXX

SP	0804
RegX	0804
AccD	0000

Local variable allocation/deallocation

```
sum   set  -4
n     set  -2
calc  pshx
      tsx
      leas -4,sp
      movw #0,sum,x
      movw #100,n,x
loop  ldd  n,x
      addd sum,x
      std  sum,x
      ldd  n,x
      subd #1
      std  n,x
      bne  loop
      txs
      pulx
```

0800	13BA
0802	0000
0804	FFFF
0806	XXXX

SP	0806
RegX	FFFF
AccD	0000

Correct code: Who do you believe?

- pg. 128 of your textbook - "Recursive algorithms are often easy to prove correct."
- Gerard J. Holzman "The Power of Ten" - Eliminating recursion can help prove boundedness of code.
- A summary follows of Gerard J. Holzman - "The Power of Ten – Rules for Developing Safety Critical Code" - http://spinroot.com/gerard/pdf/P10.pdf

Introduction

- Coding guidelines that cannot be checked by a tool are less effective.
- Too many coding guidelines aren't effective because they are not remembered or enforceable.
- The cost of restrictive guidelines may pay off with code that is more correct.

Rule 1

Rule: Restrict all code to very simple control flow constructs – do not use *goto* statements, *setjmp* or *longjmp* constructs, and direct or indirect *recursion*.

- Simple control translates into easier code verification and often improved clarity.
- Without recursion the function call graph is acyclic which directly aids in proving boundedness of the code.
- This rule doesn't require a single return point for a function although this often simplifies control flow.

Rule 2

Rule: All loops must have a fixed upper-bound. It must be trivially possible for a checking tool to *prove* statically that a preset upper-bound on the number of iterations of a loop cannot be exceeded. If the loop-bound cannot be proven statically, the rule is considered violated.

- The absence of recursion and presence of loop bounds prevents runawway code.
- Functions intended to be nonterminating must be proved to *not* terminate.
- Some functions don't have an obvious upper bound (i.e. traversing a linked list), so an artificial bound should be set and checked via an assert.

Rule 3

Rule: Do not use dynamic memory allocation after initialization.

- Memory allocation code is unpredictable from a time standpoint and therefore impractical for time critical code.
- Many errors are introduced by improper dynamic memory allocation.
- Without dynamic memory allocation the stack is used for dynamic structures and without recursion bounds can be proved on stack size.

Rule 4

Rule: No function should be longer than what can be printed on a single sheet of paper in a standard reference format with one line per statement and one line per declaration. Typically, this means no more than 60 lines of code per function.

- Long functions often indicate poor code structure.

Rule 5

Rule: The *assertion density* should average to a minimum of two assertions per function. Assertions are used to check for anomalous conditions that should never happen in real-life executions. Assertions must always be side-effect free and should be defined as Boolean tests. When an assertion fails, an explicit recovery action must be taken.

- Use of assertions is recommended as part of a strong defensive coding strategy.
- Assertions can be used to check pre- and post-conditions of functions, parameter values, return values, and loop invariants.
- Assertions can be disabled in performance critical code because they are side-effect free.

Rule 6

Rule: Data objects must be declared at the smallest possible level of scope.

- Variable will not be modified in unexpected places if they are not in scope.
- It can be easier to debug a problem if the scope of the variable is smaller.

Rule 7

Rule: The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked in each function.

- If the response to the error would be no different to the response to the success then there is no point in checking the value.
- Useless checks can be indicated by casting the return value to (void).

Rule 8

Rule: The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions. Token pasting, variable argument lists, and recursive macro calls are not allowed. All macros must expand into complete syntactic units. The use of conditional compilation directives is often also dubious but cannot always be avoided. Each use of a conditional compilation directive should be flagged by a tool-based checker and justified in the code.

- Conditional compilation directives can result in an exponentially growing number of code versions.

Rule 9

Rule: The use of pointers should be restricted. Specifically, no more than one level of dereferencing is allowed. Pointer dereference operations may not be hidden in macro definitions or inside *typedef* declarations. Function pointers are not permitted.

- Pointers are easily misused even by experienced programmers.
- Function pointers can severely limit the utility of static code checkers.

Rule 10

Rule: All code must be compiled, from the first day of development, with *all* compiler warnings enabled at the compiler's most pedantic setting. All code must compile with these settings without any warnings. All code must be checked daily with at least one, but preferably more than one, state-of-the-art static code analyzer and should pass the analyses with zero warnings.

- This rule should be followed even in the case when the warning is invalid.
- Code that confuses the compiler or checker enough to result in an invalid warning should be rewritten for clarity.
- Static checkers should be required for any serious coding project.