

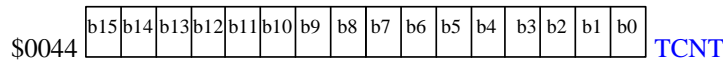
**Recap**

- Pointers
- Indexed mode
- Arrays and strings

**Overview**

- Timer
- Fixed-time delay
- Finite state machine

**4.5. 16-bit timer**



Addr	Bit 7	6	5	4	3	2	1	Bit 0	Name
\$0044	Bit 15	14	13	12	11	10	9	Bit 8	TCNT
\$0045	Bit 7	6	5	4	3	2	1	Bit 0	TCNT
\$0046	TEN	TSWAI	TSFRZ	TFFCA	0	0	0	0	TSCR1
\$004D	TOI	0	0	0	TCRE	PR2	PR1	PR0	TSCR2
\$004F	TOF	0	0	0	0	0	0	0	TFLG2

Table 4.11. 9S12 timer ports.

PR2	PR1	PR0	Divide by	E = 8 MHz		E = 24 MHz	
				TCNT period	TCNT frequency	TCNT period	TCNT frequency
0	0	0	1	125 ns	8 MHz	41.7 ns	24 MHz
0	0	1	2	250 ns	4 MHz	83.3 ns	12 MHz
0	1	0	4	500 ns	2 MHz	167 ns	6 MHz
0	1	1	8	1 μs	1 MHz	333 ns	3 MHz
1	0	0	16	2 μs	500 kHz	667 ns	1.5 MHz
1	0	1	32	4 μs	250 kHz	1.33 μs	667 kHz
1	1	0	64	8 μs	125 kHz	2.67 μs	333 kHz
1	1	1	128	16 μs	62.5 kHz	5.33 μs	167 kHz

Table 4.12. Given an E clock frequency, the PR2 PR1 and PR0 bits define the TCNT rate.

**Fixed time delay software using the built-in timer**

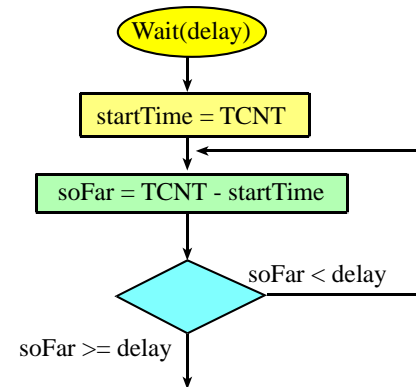
```

void Timer_Init(void){
    TSCR1 = 0x80; // enable TCNT
    TSCR2 = 0x00; // 125ns
}
void Timer_Wait(unsigned short cycles){
    unsigned short startTime = TCNT;
    while((TCNT-startTime) <= cycles){}
}
    
```

Program 4.5. Timer function that implement a fixed time delay.

```

****Timer_Init*****
* Initialize Timer
* Input: none
* Outputs: none
* error: none
Timer_Init
    movb #$80,TSCR1    enable TCNT
    movb #$00,TSCR2    ;125ns in RUN mode
    rts
    
```



```

;***Timer_Wait*****
; Time delay function
; Input: RegD time to wait (125ns)
; Outputs: none
; error: input must be less than 60000
Timer_Wait
    std  delay        ;time to wait
    movw TCNT,start  ;TCNT at start
Wloop ldd  TCNT        ;now
    subd start        ;soFar=TCNT-Start
    cpd  delay
Wchk  blo  Wloop      ;loop if soFar<delay
    rts

```

### Show Timer\_Wait starter file

Run until `start` is 50000 `RegD` is about 14000

Put a scan point at `Wchk`

Watch `TCNT` roll over

It works because all data are 16-bit unsigned

### Real time systems

Bounded latency

Input interface:

input interface latency RDRF -> Read SCIDRL

Output interface:

output interface latency TDRE -> Write SCIDRL

Periodic process (square wave, Labs 6, 7 and 8):

software activity occurs at a periodic rate,  $\Delta t$

let  $t_n$  be the  $n$ th time the process executes

goal to make  $t_n - t_{n-1} = \Delta t$

$\delta t = \text{jitter}$   $\Delta t - \delta t < t_i - t_{i-1} < \Delta t + \delta t$  for all  $i$

\*\*\*\*\*TimerWait\*\*\*\*\*

Does the LED flash at exactly 5 Hz?

Put a ScanPoint at PTP toggle, measure jitter

show the sliding time

measure the jitter

$100\text{ms} - \delta t < t_n - t_{n-1} < 100\text{ms} + \delta t$

fix to have almost perfect timing

measure the jitter again

## 8.7. Finite state machines with statically-allocated linked structures

### 8.7.1. Abstraction

Software abstraction

define a problem with a set of basic abstract principles

separate policies mechanisms

Finite State Machine (FSM.)

inputs, outputs, states, and state transitions

state graph defines relationships of inputs and outputs

The three advantages of this abstraction are

- 1) it can be faster to develop
- 2) it is easier to debug (prove correct) and
- 3) it is easier to change

What is a state?

Description of current conditions

What is a state graph?

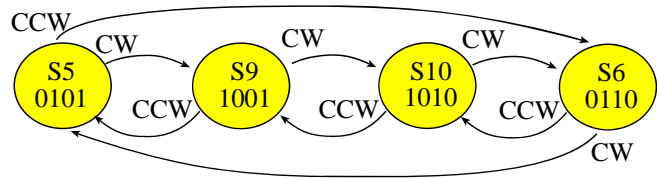
Graphical interconnection between states

What is a controller?

- Software that inputs, outputs, changes state
- Accesses the state graph

What is a finite state machine?

- Input sensors
- Output actuators
- Controller
- State graph



Moore FSM

- output value depends only on the current state, and
- inputs affect the state transitions
- significance is being in a state

input: when to change state

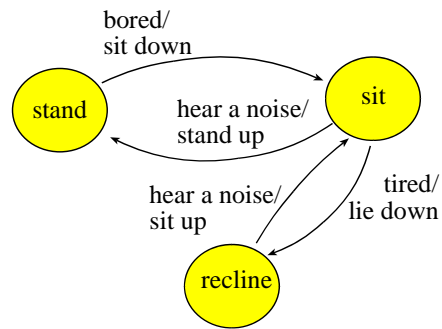
output: how to be in that state

Mealy FSM depend both on the current state and the inputs.

- output value depends on input and current state
- inputs affect the state transitions.
- significance is the state transition

input: when to change state

output: how to change state



**data structure** embodies the FSM

- multiple identically-structured nodes
- statically-allocated fixed-size linked structures
- one-to-one mapping FSM state graph and linked structure
- one structure for each state

**linked structure**

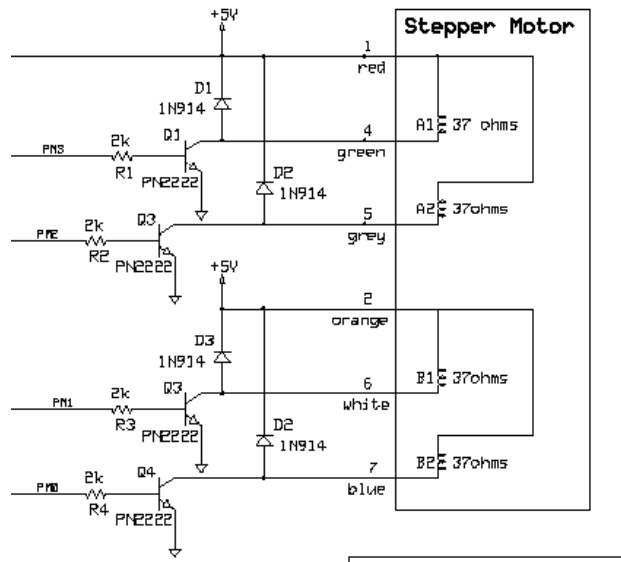
- pointer (or link) to other nodes (define next states)

**table structure**

- indices to other nodes (define next states)

**Stepper motor controller**

*This stepper motor FSM has two input signals four outputs.*



Many hardware circuits in this class will be drawn with a free drawing tool  
PCBArtist, <http://www.4pcb.com/>

- 1) Design- data flow graph, flowchart, pseudocode
- 2) Implement in TExaS, debug it
- 3) Switch to Real mode
- 4) place 9S12 in LOAD mode
  - Cable from PC to docking module
  - Power applied to embedded system, reset button
  - Execute Assemble to download
  - Run from debugger, 24 MHz
- 5) place in RUN mode
  - Power applied to embedded system, reset button
  - It's running at 8 MHz

Write in 9S12C32 assembly

```
;*****
PTAD    equ  $0270    ; Port AD I/O Register
DDRAD   equ  $0272    ; Port AD Data Direction Register
ATDDIEN equ  $008D    ; ATD Input Enable Mask Register
DDRM    equ  $0252    ; Port M Data Direction Register
PTM     equ  $0250    ; Port M I/O Register
DDRT    equ  $0242    ; Port T Data Direction Register
PTT     equ  $0240    ; Port T I/O Register
TCNT    equ  $0044    ; Timer Count Register
TSCR1   equ  $0046    ; Timer System Control Register1
TSCR2   equ  $004D    ; Timer System Control Register 2
        org  $3800    ; Globals go in 2K Ram
delay   ds.w 1        ; number of cycles to wait
start   ds.w 1        ; TCNT value at the start of wait
Pt      rmb 2        ;pointer to current state

        org  $4000
out     equ  0        ;8-bit output
wait    equ  1        ;time to wait, 32us units
next    equ  3        ;4 pointers to next state
S5      fcb  $05      ;4-bit output
        fdb  4000
        fdb  S5,S9,S6,S5 ;next for each in
S6      fcb  $06
        fdb  4000
        fdb  S6,S5,S10,S6
S10     fcb  $0A
        fdb  4000
        fdb  S10,S6,S9,S10
S9      fcb  $09
        fdb  4000
        fdb  S9,S10,S5,S9
* ROM program
Main    lds  #$4000
        bsr  Timer_Init    ; activate TCNT
        bset DDRT,$03      ; PT1 PT0 output to LEDs
        bset ATDDIEN,$C0   ; PAD6,7 digital
        bclr DDRAD,$C0     ; PAD6,7 input
        bset DDRM,$0F      ; PM3-0 output
        movb #$05,PTM      ; initial output
        movw #S5,Pt        ; initial state
        cli                ; allow debugger
loop    ldx  Pt
        movb out,X,PTM     ; step motor
        ldd  wait,X
        bsr  Timer_Wait    ; wait specified time
```

```
ldaa PTAD      ; read inputs (negative logic)
eora #$C0     ; positive logic
anda #$C0     ; just CCW,CW
              ; 0,40,80,C0
lsra          ; 0,20,40,60
lsra          ; 0,10,20,30
lsra          ; 0,08,10,18
lsra          ; 0,04,08,0C
lsra          ; 0,02,04,06
leax next,X   ; list of pointers
ldx A,X       ; next depends on in
stx Pt
ldaa PTT
eora #$01
staa PTT      ; heart beat
bra loop
```

**Run in simulator, scan point on PTM output**  
**Run on 9S12C32**

#### **The bottom line**

**Timer is an accurate wait to create delays**  
**Finite state machines provide for abstraction**  
**Simple controller, complicated state graph**