Recap
Interrupts
The Task Model

Course outline

- **The plan is indicative, subject to change!**

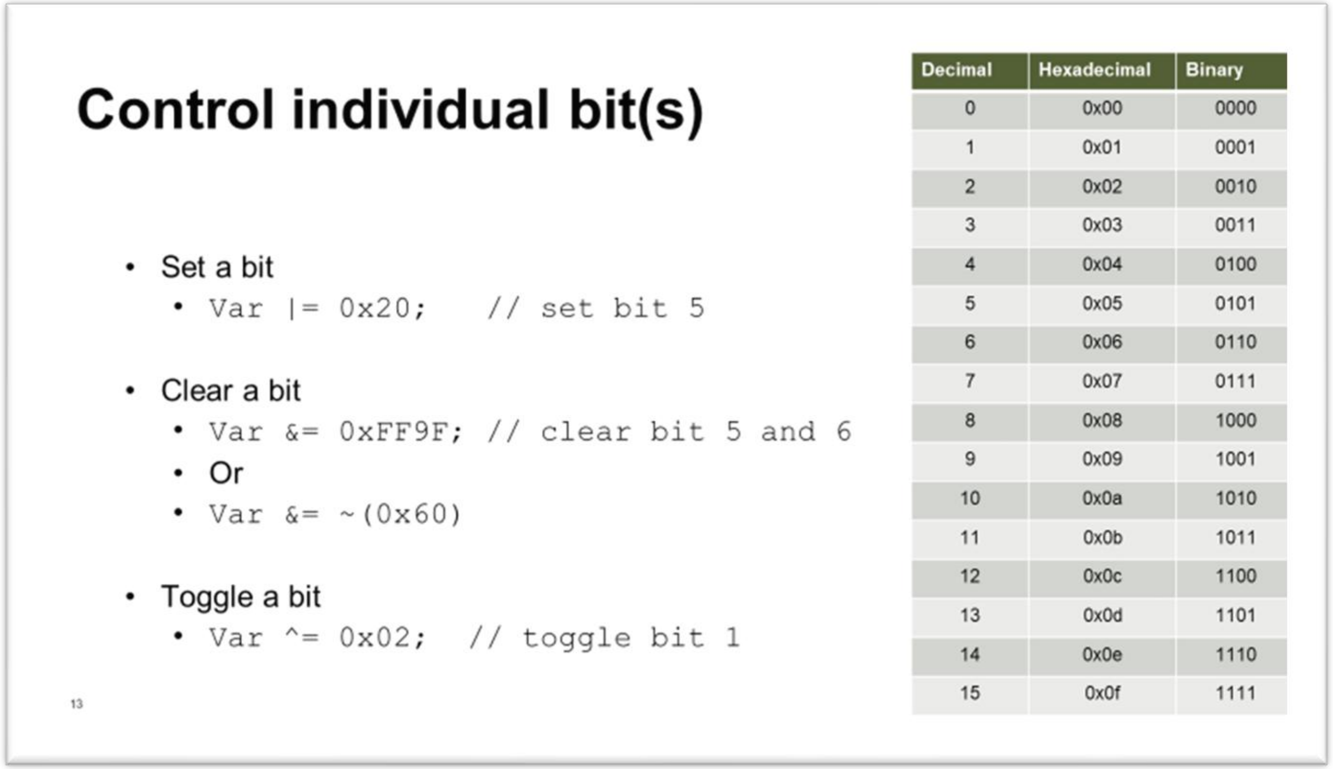
Date	Lecture	Subject(s)	Lab	Assignment due
Feb 2	1	Introduction; ARM Cortex-M4	Lab1: Setting up the development env.	
Feb 9	2	EMP coding standard; Bit manipulation	Lab2: Bit manipulation	
Feb 16	3	State machines; The compiler chain; EMP-Board	Lab3: State machines	PF1 – Assignment 1
Feb 23	4	The task model; The pre-processor	Lab4: A clock radio task model	PF1 – Assignment 2
Mar 2	5	Queues and semaphores; Debugging	Lab5: Debug with a serial connection	PF1 – Assignment 3
Mar 9	6	Run to complete scheduler	Lab6: RTCS, Run to compile scheduler	PF1 – Assignment 4
Mar 16	7	More debugging; C: printf()	Lab7: RTCS, Debugger	PF1 – Assignment 5
March 23	8	FreeRTOS	Lab8: FreeRTOS	PF1 – Assignment 6
March 30 April 6		Easter holiday		
Apr 13	9	More queues; Assembler in C	Lab9: FreeRTOS (continued)	PF1 – Assignment 7
Apr 20	10	Re-entrance		PF1 – Assignment 8
Apr 27	11	Work on the final assignment, consultations		
May 4	12	Poster session		PF2- Final Assignment

Recap Lecture01

- Introduction to the course
  - Learning objectives
  - Assignments, Exam
  - Class outline: assignment presentation, lecture, lab, assignment introduction
  - Semester overview
  - Suggested books to read
- ARM Cortex M4
  - Why embedded, why ARM
  - Basic characteristics

Lecture02

- EMP coding standard
  - How to format your code to make it readable, reusable
- Bit manipulation
  - Why it is important
  - How to use C operators



Control individual bit(s)

- Set a bit
  - `Var |= 0x20;   // set bit 5`
- Clear a bit
  - `Var &= 0xFF9F; // clear bit 5 and 6`
  - Or
  - `Var &= ~(0x60)`
- Toggle a bit
  - `Var ^= 0x02;  // toggle bit 1`

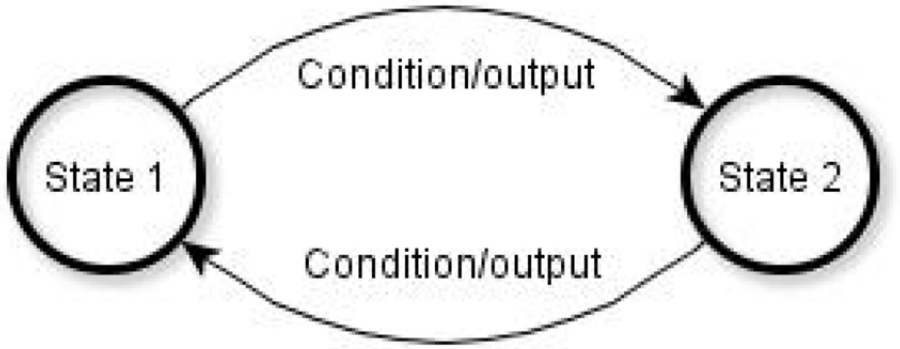
Decimal	Hexadecimal	Binary
0	0x00	0000
1	0x01	0001
2	0x02	0010
3	0x03	0011
4	0x04	0100
5	0x05	0101
6	0x06	0110
7	0x07	0111
8	0x08	1000
9	0x09	1001
10	0x0a	1010
11	0x0b	1011
12	0x0c	1100
13	0x0d	1101
14	0x0e	1110
15	0x0f	1111

13

Lecture03

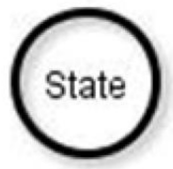
- The Build Process
  - The embedded C build process
  - Preprocessor, compiler, linker, locator
- Debugging
- State machines
  - Concept
  - Door example
  - Solution to Assignment01 with state machines

State Machines - recap

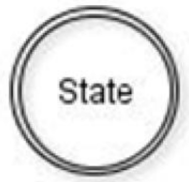


State Charts - Elements

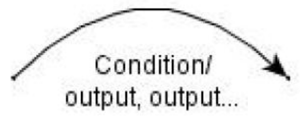
A State



Start State

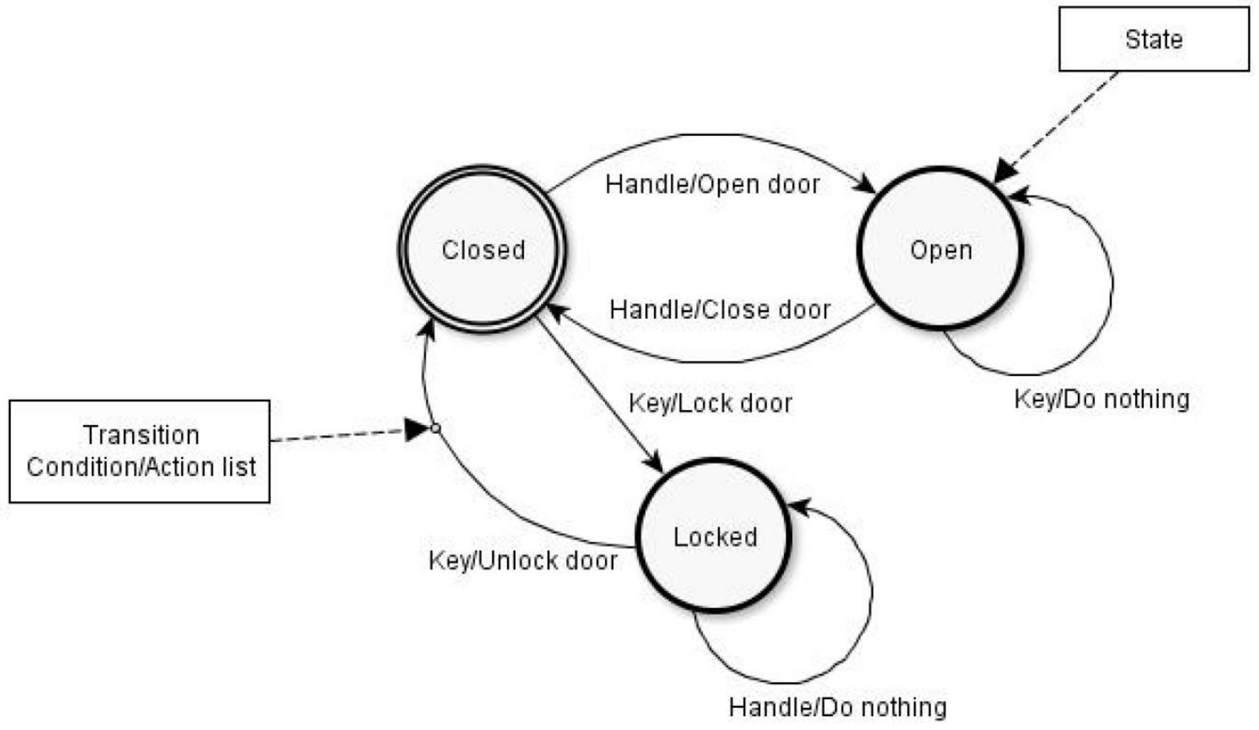


Transition Arrow



(Condition may be an event or an external state)

Example task: door



Example task: door

```
int8u the_door( event )
int8u event;
{
  static int8u  state  = CLOSED;
  int8u         action = NO_ACTION;

  switch( state )
  {
    case CLOSED:
      switch( event )
      {
        case HANDLE:
          action = OPEN_DOOR;
          state  = OPEN;
          break;
        case KEY:
          action = LOCK_DOOR;
          state  = LOCKED;
          break;
        default:
      }
    case OPEN:
      switch( event )
      {
        case HANDLE:
          action = CLOSE_DOOR;
          state  = CLOSED;
          break;
        case KEY:
          break;
        default:
      }
    case LOCKED:
      switch( event )
      {
        case HANDLE:
          break;
        case KEY:
          action = UNLOCK_DOOR;
          state  = CLOSED;
          break;
          default:
      }
    default:
  }
  return( action );
}
```

break;

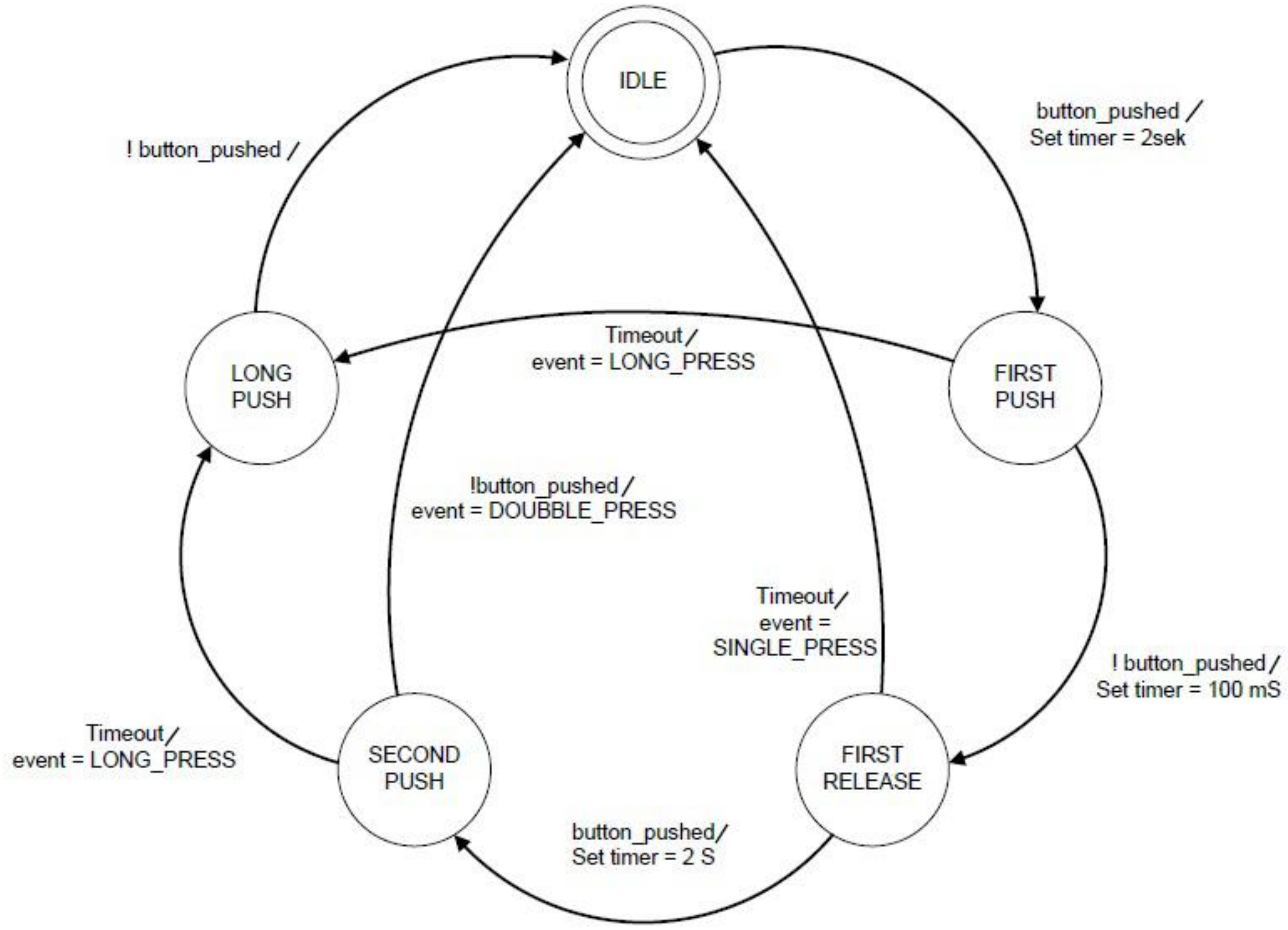
break;

or

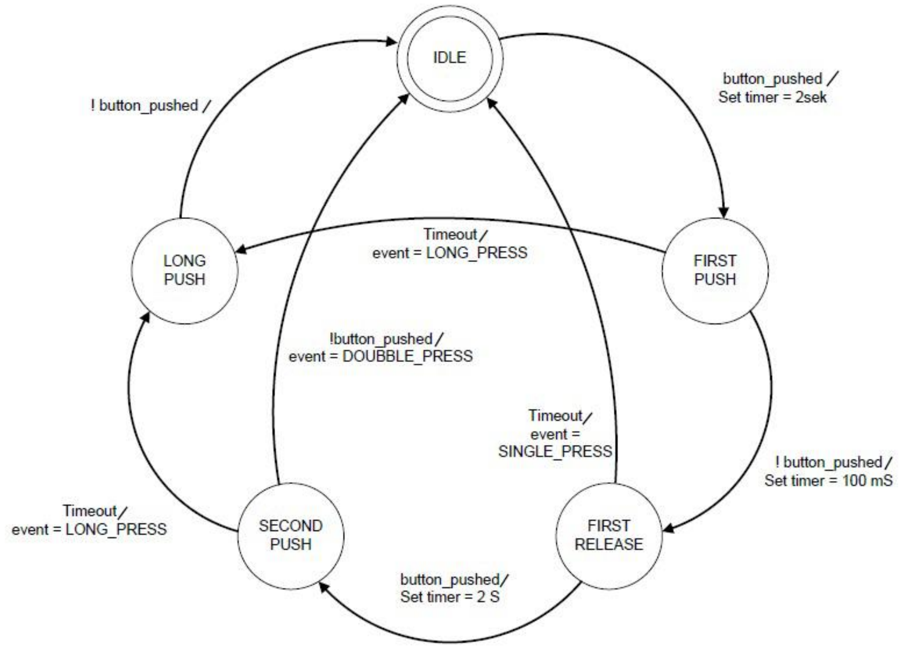
```
int8u the_door( event )
int8u event;
{
  static int8u  state  = CLOSED;
  int8u         action = NO_ACTION;

  switch( event )
  {
    case HANDLE:
      switch( state )
      {
        case CLOSED:
          action = OPEN_DOOR;
          state  = OPEN;
          break;
        case OPEN:
          action = CLOSE_DOOR;
          state  = CLOSED;
          break;
        case LOCKED:
          break;
        default:
      }
    case KEY:
      switch( state )
      {
        case CLOSED:
          action = LOCK_DOOR;
          state  = LOCKED;
          break;
        case OPEN:
          break;
        case LOCKED:
          action = UNLOCK_DOOR;
          state  = CLOSED;
          break;
        default:
      }
    default:
  }
  return( action );
}
```

break;

Select button – state machine


IDLE
button_pushed / Set timer = 2sek
FIRST PUSH
Timeout / event = LONG_PRESS
LONG PUSH
! button_pushed /
! button_pushed / Set timer = 100 mS
FIRST RELEASE
Timeout / event = SINGLE_PRESS
button_pushed / Set timer = 2 S
SECOND PUSH
!button_pushed / event = DOUBBLE_PRESS
Timeout / event = LONG_PRESS

Select button state machine



IDLE
button_pushed / Set timer = 2sek
! button_pushed /
LONG PUSH
FIRST PUSH
Timeout / event = LONG_PRESS
!button_pushed / event = DOUBBLE_PRESS
Timeout / event = SINGLE_PRESS
! button_pushed / Set timer = 100 mS
Timeout / event = LONG_PRESS
SECOND PUSH
FIRST RELEASE
button_pushed / Set timer = 2 S

```
INT8U select_button(void)
{
  switch( button_state )
  {
    case BS_IDLE:
      if( button_pushed( ))       // if button pushed
      {
        button_state = BS_FIRST_PUSH;
        button_timer = TIM_2_SEC;// start timer = 2 sek;
      }
      break;
    case BS_FIRST_PUSH:
      if( ! --button_timer )      // if timeout
      {
        button_state = BS_LONG_PUSH;
        button_event = BE_LONG_PUSH;
      }
      else
      {
        if( !button_pushed( ) )// if button released
        {
          button_state = BS_FIRST_RELEASE;
          button_timer = TIM_100_MSEC;// start timer = 100 milli sek;
        }
      }
      break;
    case BS_FIRST_RELEASE:
:::::
```

Your turn

- State machine for combination lock?
- State machine for traffic light?

Recap Interrupts

Interrupts

- An **interrupt** is the automatic transfer of software execution in response to a hardware event that is asynchronous with the current software execution
- This hardware event is called a **trigger**
- When this happens, an ISR (interrupt service routine, i.e. interrupt handler) is called
- This pauses the execution of the regular code, until the ISR returns or a higher priority interrupt occurs

Exceptions and Interrupts on TIVA

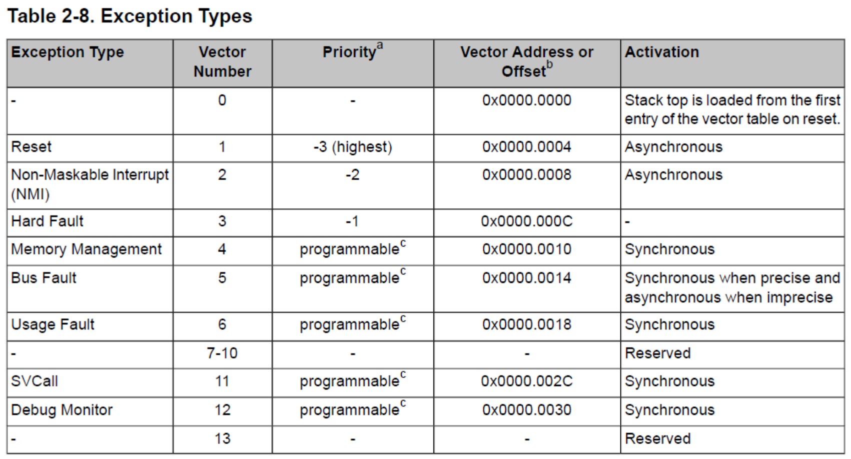


Table 2-8. Exception Types

Exception Type	Vector Number	Priority[a]	Vector Address or Offset[b]	Activation
-	0	-	0x0000.0000	Stack top is loaded from the first entry of the vector table on reset.
Reset	1	-3 (highest)	0x0000.0004	Asynchronous
Non-Maskable Interrupt (NMI)	2	-2	0x0000.0008	Asynchronous
Hard Fault	3	-1	0x0000.000C	-
Memory Management	4	programmable[c]	0x0000.0010	Synchronous
Bus Fault	5	programmable[c]	0x0000.0014	Synchronous when precise and asynchronous when imprecise
Usage Fault	6	programmable[c]	0x0000.0018	Synchronous
-	7-10	-	-	Reserved
SVCall	11	programmable[c]	0x0000.002C	Synchronous
Debug Monitor	12	programmable[c]	0x0000.0030	Synchronous
-	13	-	-	Reserved

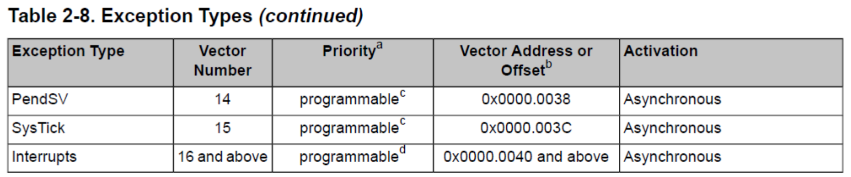


Table 2-8. Exception Types *(continued)*

Exception Type	Vector Number	Priority[a]	Vector Address or Offset[b]	Activation
PendSV	14	programmable[c]	0x0000.0038	Asynchronous
SysTick	15	programmable[c]	0x0000.003C	Asynchronous
Interrupts	16 and above	programmable[d]	0x0000.0040 and above	Asynchronous

Exceptions – software generated faults

Interrupts – hardware generated events

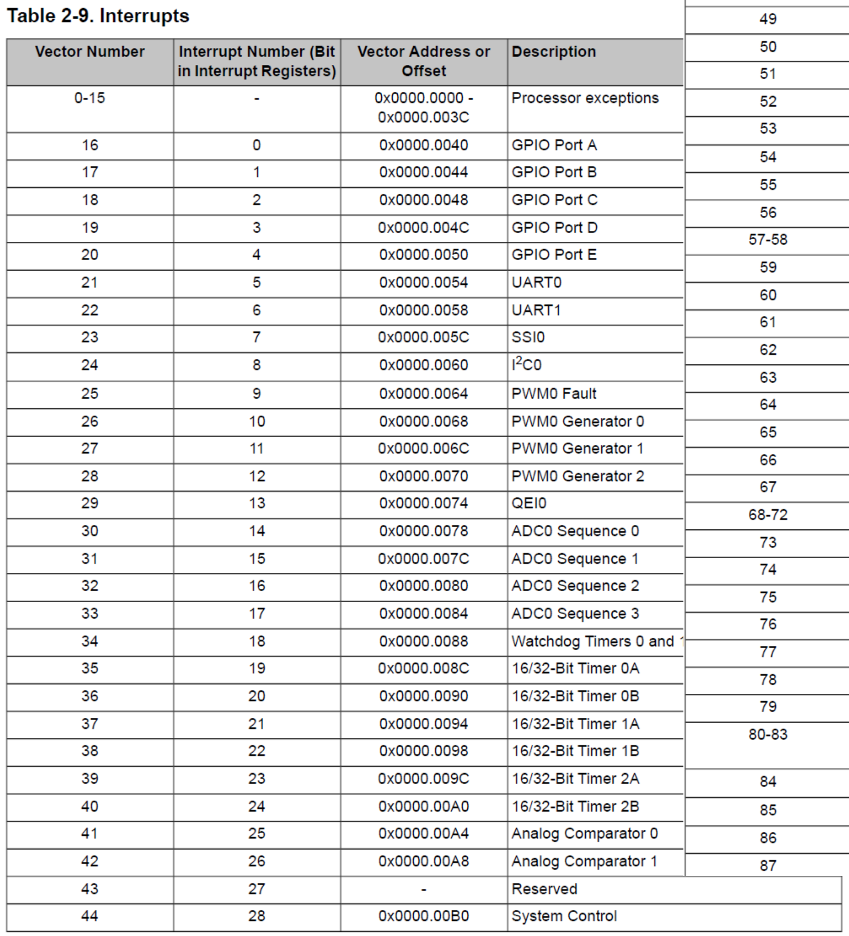


Table 2-9. Interrupts

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
0-15	-	0x0000.0000 - 0x0000.003C	Processor exceptions
16	0	0x0000.0040	GPIO Port A
17	1	0x0000.0044	GPIO Port B
18	2	0x0000.0048	GPIO Port C
19	3	0x0000.004C	GPIO Port D
20	4	0x0000.0050	GPIO Port E
21	5	0x0000.0054	UART0
22	6	0x0000.0058	UART1
23	7	0x0000.005C	SSI0
24	8	0x0000.0060	I^{2}C0
25	9	0x0000.0064	PWM0 Fault
26	10	0x0000.0068	PWM0 Generator 0
27	11	0x0000.006C	PWM0 Generator 1
28	12	0x0000.0070	PWM0 Generator 2
29	13	0x0000.0074	QEI0
30	14	0x0000.0078	ADC0 Sequence 0
31	15	0x0000.007C	ADC0 Sequence 1
32	16	0x0000.0080	ADC0 Sequence 2
33	17	0x0000.0084	ADC0 Sequence 3
34	18	0x0000.0088	Watchdog Timers 0 and
35	19	0x0000.008C	16/32-Bit Timer 0A
36	20	0x0000.0090	16/32-Bit Timer 0B
37	21	0x0000.0094	16/32-Bit Timer 1A
38	22	0x0000.0098	16/32-Bit Timer 1B
39	23	0x0000.009C	16/32-Bit Timer 2A
40	24	0x0000.00A0	16/32-Bit Timer 2B
41	25	0x0000.00A4	Analog Comparator 0
42	26	0x0000.00A8	Analog Comparator 1
43	27	-	Reserved
44	28	0x0000.00B0	System Control

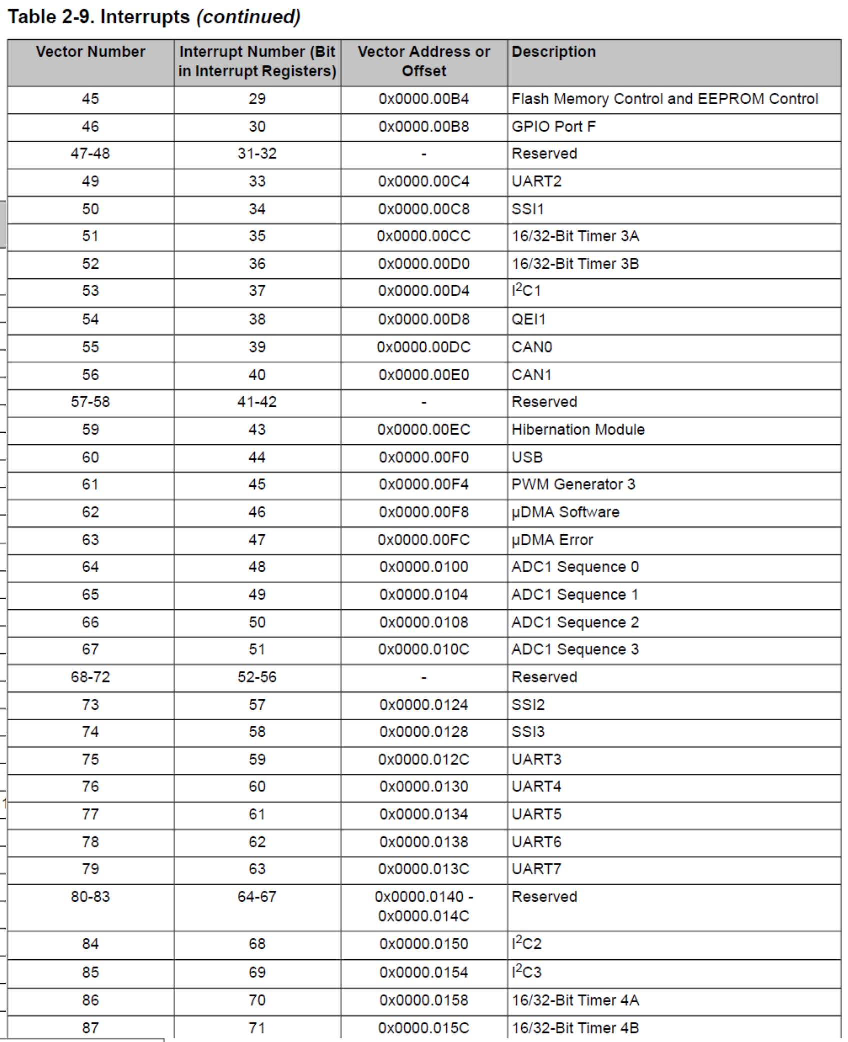
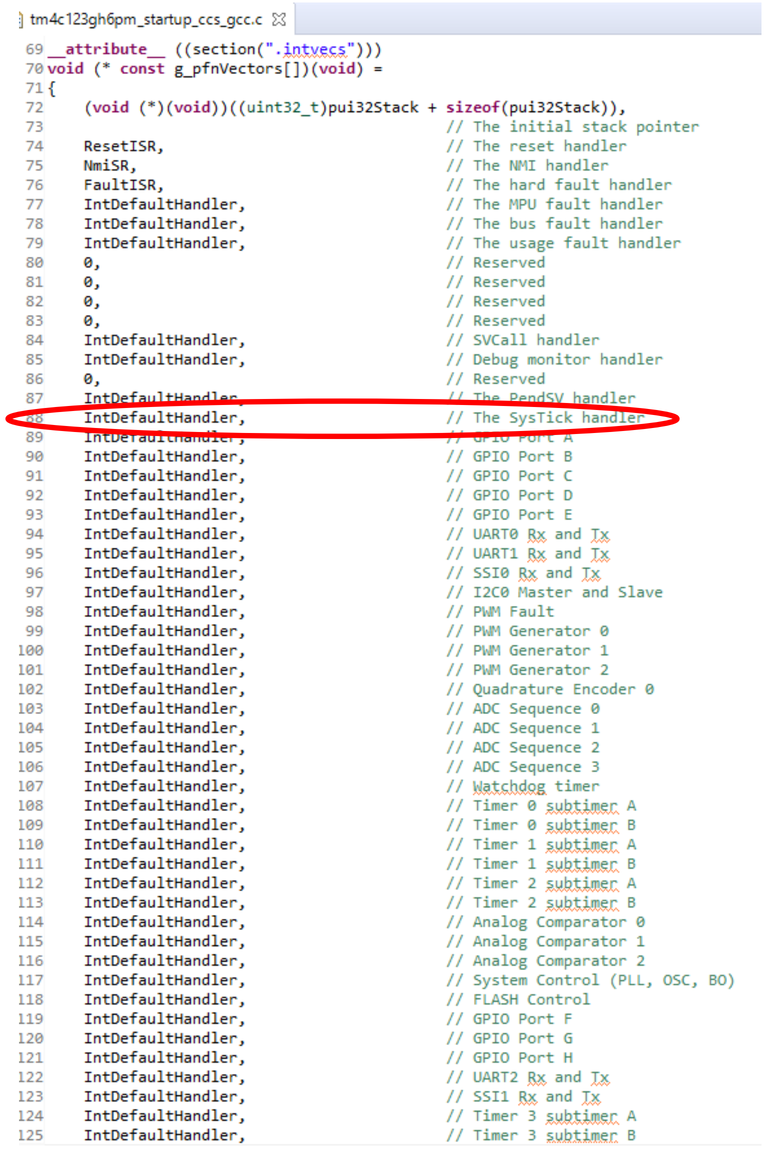


Table 2-9. Interrupts *(continued)*

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
45	29	0x0000.00B4	Flash Memory Control and EEPROM Control
46	30	0x0000.00B8	GPIO Port F
47-48	31-32	-	Reserved
49	33	0x0000.00C4	UART2
50	34	0x0000.00C8	SSI1
51	35	0x0000.00CC	16/32-Bit Timer 3A
52	36	0x0000.00D0	16/32-Bit Timer 3B
53	37	0x0000.00D4	I^{2}C1
54	38	0x0000.00D8	QEI1
55	39	0x0000.00DC	CAN0
56	40	0x0000.00E0	CAN1
57-58	41-42	-	Reserved
59	43	0x0000.00EC	Hibernation Module
60	44	0x0000.00F0	USB
61	45	0x0000.00F4	PWM Generator 3
62	46	0x0000.00F8	µDMA Software
63	47	0x0000.00FC	µDMA Error
64	48	0x0000.0100	ADC1 Sequence 0
65	49	0x0000.0104	ADC1 Sequence 1
66	50	0x0000.0108	ADC1 Sequence 2
67	51	0x0000.010C	ADC1 Sequence 3
68-72	52-56	-	Reserved
73	57	0x0000.0124	SSI2
74	58	0x0000.0128	SSI3
75	59	0x0000.012C	UART3
76	60	0x0000.0130	UART4
77	61	0x0000.0134	UART5
78	62	0x0000.0138	UART6
79	63	0x0000.013C	UART7
80-83	64-67	0x0000.0140 - 0x0000.014C	Reserved
84	68	0x0000.0150	I^{2}C2
85	69	0x0000.0154	I^{2}C3
86	70	0x0000.0158	16/32-Bit Timer 4A
87	71	0x0000.015C	16/32-Bit Timer 4B

Interrupts on TIVA

- Interrupt handlers need to be registered in tm4c123gh6pm_startup_ccs_gcc.c
- This file is automatically generated and added to your project
- Additional reading: http://shukra.cedt.iisc.ernet.in/edwiki/EmSys:Interrupts_in_TM4C123GH6PM_Launchpad



tm4c123gh6pm_startup_ccs_gcc.c

```
__attribute__ ((section(".intvecs")))
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)pui32Stack + sizeof(pui32Stack)),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
    0,                                      // Reserved
    IntDefaultHandler,                      // The PendSV handler
    IntDefaultHandler,                      // The SysTick handler
    IntDefaultHandler,                      // GPIO Port A
    IntDefaultHandler,                      // GPIO Port B
    IntDefaultHandler,                      // GPIO Port C
    IntDefaultHandler,                      // GPIO Port D
    IntDefaultHandler,                      // GPIO Port E
    IntDefaultHandler,                      // UART0 Rx and Tx
    IntDefaultHandler,                      // UART1 Rx and Tx
    IntDefaultHandler,                      // SSI0 Rx and Tx
    IntDefaultHandler,                      // I2C0 Master and Slave
    IntDefaultHandler,                      // PWM Fault
    IntDefaultHandler,                      // PWM Generator 0
    IntDefaultHandler,                      // PWM Generator 1
    IntDefaultHandler,                      // PWM Generator 2
    IntDefaultHandler,                      // Quadrature Encoder 0
    IntDefaultHandler,                      // ADC Sequence 0
    IntDefaultHandler,                      // ADC Sequence 1
    IntDefaultHandler,                      // ADC Sequence 2
    IntDefaultHandler,                      // ADC Sequence 3
    IntDefaultHandler,                      // Watchdog timer
    IntDefaultHandler,                      // Timer 0 subtimer A
    IntDefaultHandler,                      // Timer 0 subtimer B
    IntDefaultHandler,                      // Timer 1 subtimer A
    IntDefaultHandler,                      // Timer 1 subtimer B
    IntDefaultHandler,                      // Timer 2 subtimer A
    IntDefaultHandler,                      // Timer 2 subtimer B
    IntDefaultHandler,                      // Analog Comparator 0
    IntDefaultHandler,                      // Analog Comparator 1
    IntDefaultHandler,                      // Analog Comparator 2
    IntDefaultHandler,                      // System Control (PLL, OSC, BO)
    IntDefaultHandler,                      // FLASH Control
    IntDefaultHandler,                      // GPIO Port F
    IntDefaultHandler,                      // GPIO Port G
    IntDefaultHandler,                      // GPIO Port H
    IntDefaultHandler,                      // UART2 Rx and Tx
    IntDefaultHandler,                      // SSI1 Rx and Tx
    IntDefaultHandler,                      // Timer 3 subtimer A
    IntDefaultHandler,                      // Timer 3 subtimer B
```

Task Model

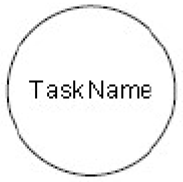
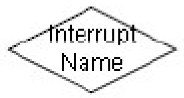
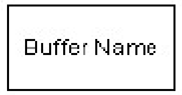
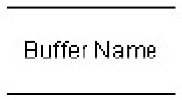
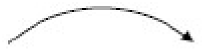
Task Model

- Very important concept
- Task
  - A task is an independent thread of execution that can compete with other concurrent tasks for processor execution time.
  - A task is defined by a set of parameters and supporting data structures:
    - a name
    - a unique ID
    - a priority
    - a task control block (TCB)
    - a stack
    - a task routine

How to divide application into tasks?

- Division into tasks is not trivial. It is often done by experience and intuition
- Criteria for task creation
  - Parallelism
    - Simultaneous and independent functionality
  - Timing
    - Different timing constraints
  - Priority
    - Divide tasks with different priority
  - Structure
    - Each task handles one well defined problem
  - Coupling
    - Divide problem into loosely coupled tasks
  - Periodicity
    - A task that must execute with a fixed period is a task by itself

Task Diagram Elements

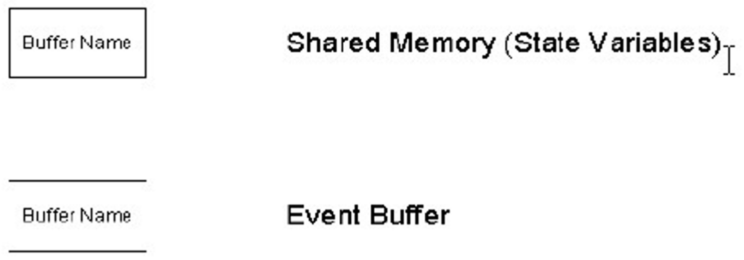
	Task
	Interrupt Service Routine
	Shared Memory (State Variables)
	Event Buffer
	Data or Information Flow.

Don’t mix up state machines and task models

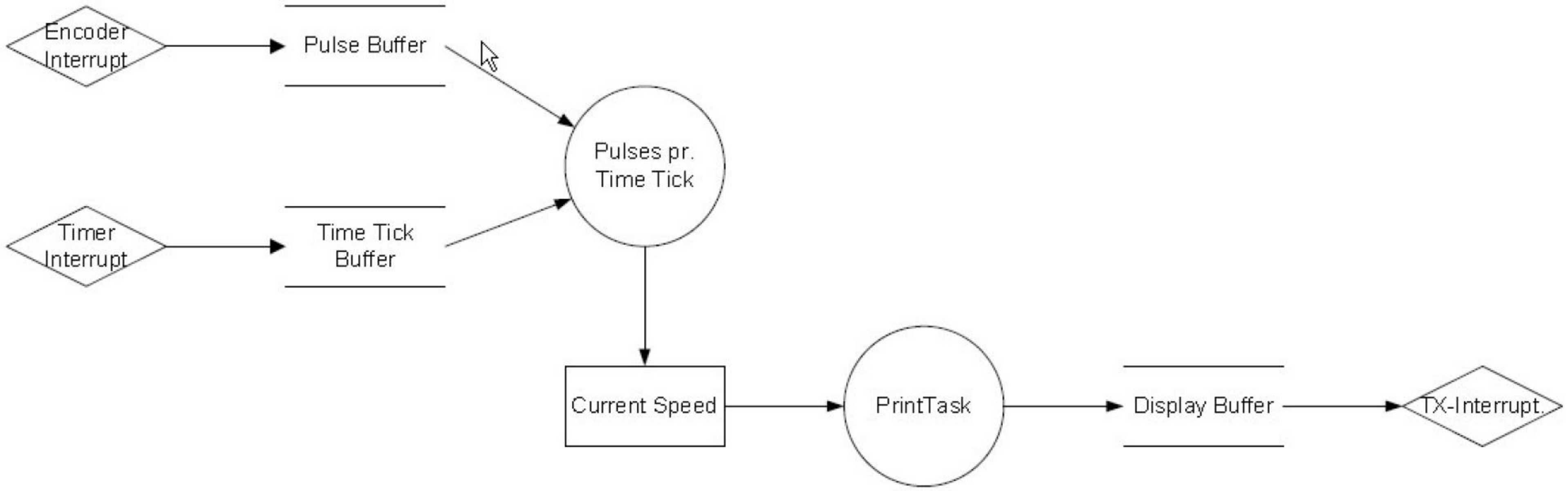
- State machines describe the single thread of execution based on states and transitions between them
    - In other words, state machines describe how states are connected to each other and how **control is transferred** from one state to the next
- Task models describe multiple tasks and the **flow of information** between them which can run in parallel
- Task models and state machines are compatible with each other – one solution can contain both

Shared memory vs event buffer

- A shared memory element keeps its value after it has been read (e.g. a state variable)
- An event from the event buffer gets destroyed after reading and processing

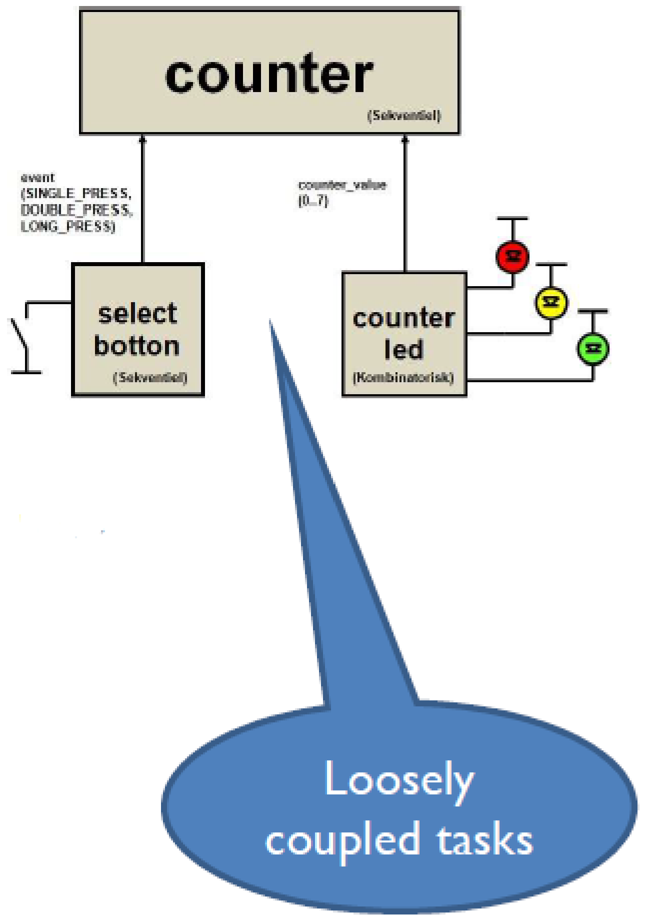


Task Diagram Example - speedometer

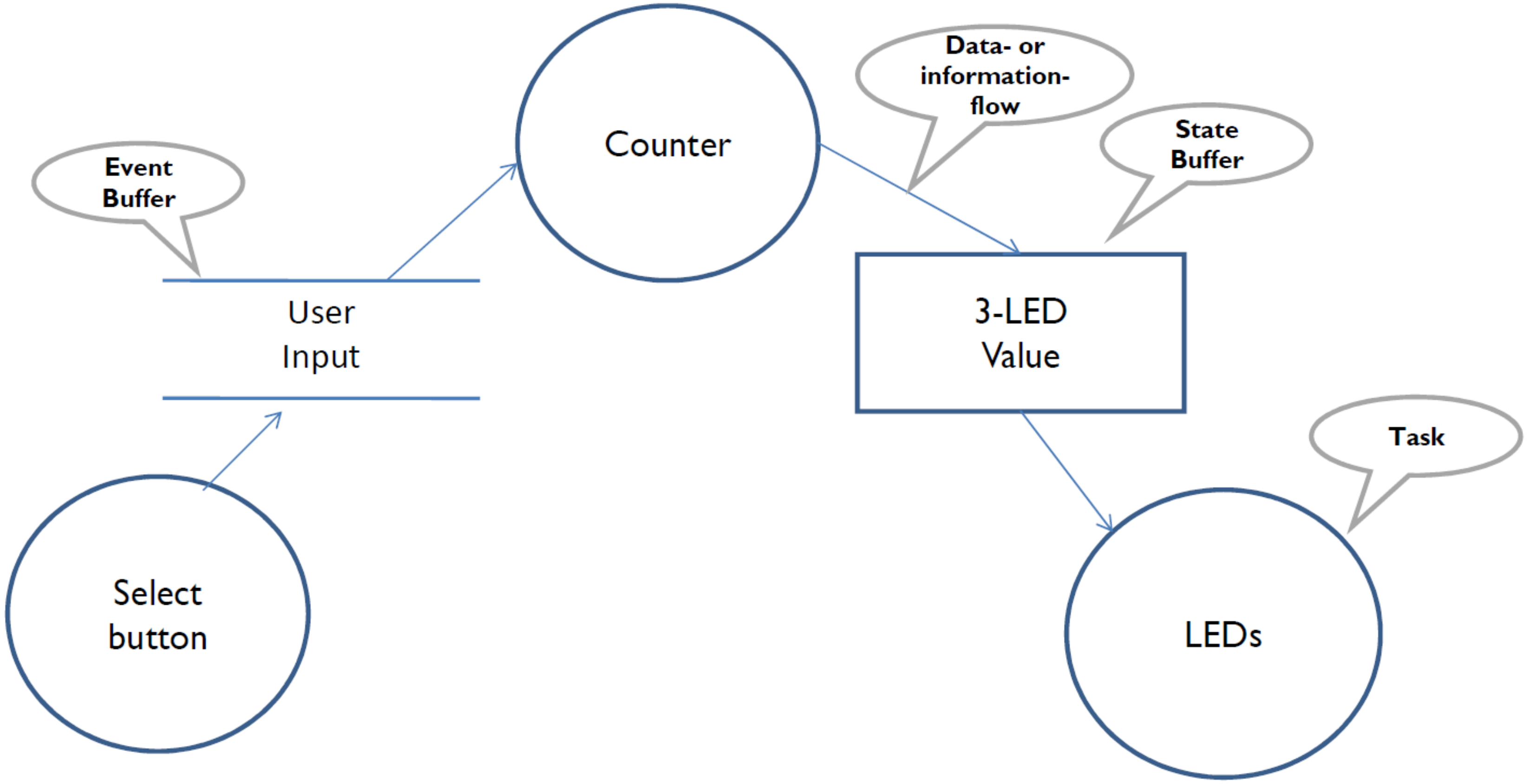


Solution to Assgn1 revisited with tasks

```
// The Super Loop.
// ---------------
while(1)
{
  // System part of the super loop.
  // ------------------------------
  while( !ticks );
    // The following will be executed every 5 mS
    ticks--;
    if( ! --alive_timer )
    {
      alive_timer =TIM_500_MSEC;
      GPIO_PORTF_DATA_R ^= 0x01;
    }
   // Application part of the super loop.
   // -----------------------------------
  event = select_button();
  counter_value = counter( event );
  counter_leds( counter_value );
}
```



Task model for Assgnment1



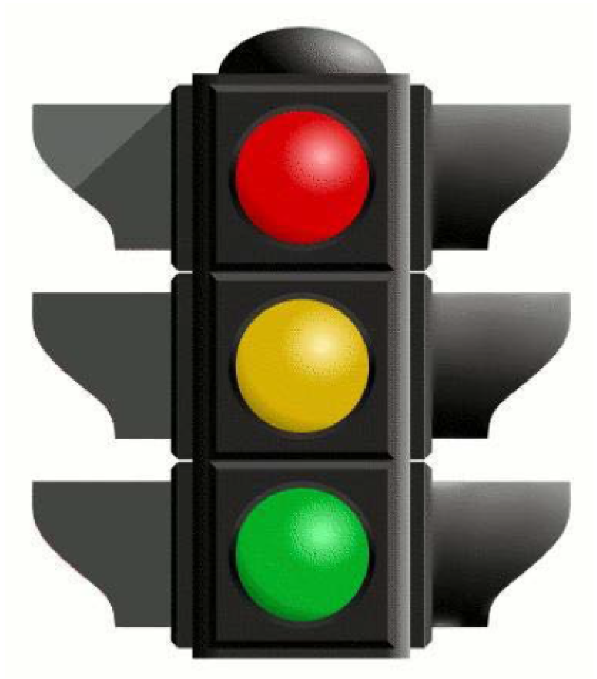
Assignment2

Assignment 2 – Traffic Light

Design a program for the kit, which implements a traffic light, using the multicolored LED of the Tiva board.

The traffic light must cycle through the states: red, yellow and green using a suitable timing.
Remember, that sometimes the red and the yellow lights are lit at the same time. As we are using a single multicolor LED, let the pink color represent the red and yellow lights coming on at the same time.

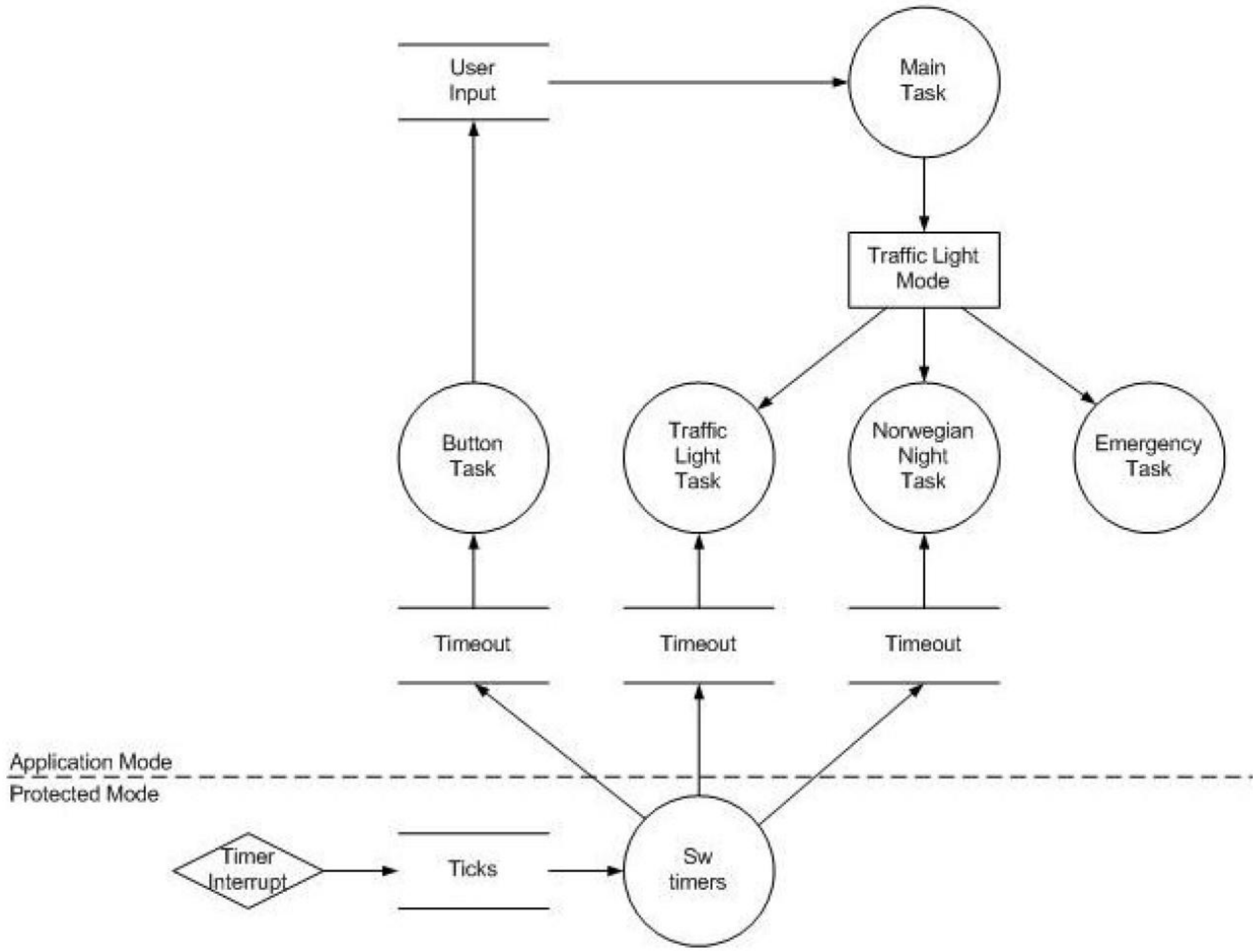
A single press at <SW1> puts the traffic light in "Norwegian night" state, where the yellow light flashes once every second. A double press at <SW1> puts the traffic light in an emergency state, where the red light is lit constantly. A 2-second-long press at <SW1> returns the traffic light back to normal operation.



All the timing in the system is controlled by a single timer interrupt and the "ticks" variable.

The Program must be designed using at least 2 separated tasks:
...the Traffic Light must be implemented as a state machine with <SW1> presses and timer events as inputs.
...a **driver**, which read the <SW1> button and hands over a <SW1> press event to the traffic light.

Assignment2 Task Diagram



Tmodel.h

```
// Tasks.
// ------
#define TASK_SW_TIMERS 11
#define TASK_MAIN 12
#define TASK_BUTTON 13
#define TASK_TRAFFIC_LIGHT 14
#define TASK_NORWEGIAN_NIGHT 15
#define TASK_EMERGENCY 16
// Interrupt Service Routines.
// ---------------------------
#define ISR_TIMER 21
// Shared State Buffers.
// --------------------
#define SSM_TRAFFIC_LIGHT_MODE 31
// Shared Event Buffers.
// ---------------------
#define SEB_USER_INPUT 41
#define SEB_TO_BUTTON 42 // Time Out
#define SEB_TO_TRAFFIC_LIGHT 43 // Time Out
#define SEB_TO_NORWEGIAN_NIGHT 44 // Time Out
```

The main loop (main.c)

```c
void main(void)
{
  initialize_system();
  while(1)
  {
    while( !ticks );
    // The following will be executed every 5 mS
    ticks--;
    // Operating System Mode
    swt_ctrl();
    // Application Mode
    button_task( TASK_BUTTON );
    main_task( TASK_MAIN );
    tl_task( TASK_TRAFFIC_LIGHT );
    flash_task( TASK_NORWEGIAN_NIGHT );
    red_task( TASK_EMERGENCY );
  }
}
```

Software timer control

```c
void swt_ctrl(void)
{
  INT8U i;
  for( i= 0; i<MAX_SWTIMERS; i++ )
  {
    if( POT[i].timer_value )
    {
      POT[i].timer_value--;
      if( !POT[i].timer_value )
      {
        put_msg_event( POT[i].event_buffer, TE_TIMEOUT );
      }
    }
  }
}
```

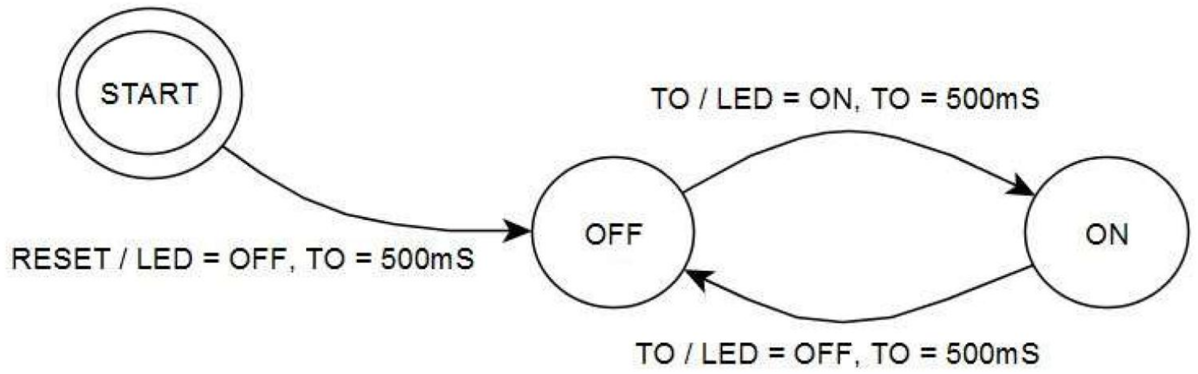
Message Events

```
INT8U get_msg_event( INT8U );
/*****************************************************************
***
* Input : Event Buffer ID
* Output : Event
* Function : Get an event
******************************************************************
**/
void put_msg_event( INT8U, INT8U );
/*****************************************************************
***
* Input : Event Buffer ID, Event
* Output : -
* Function : Put an Event
******************************************************************
**/
```

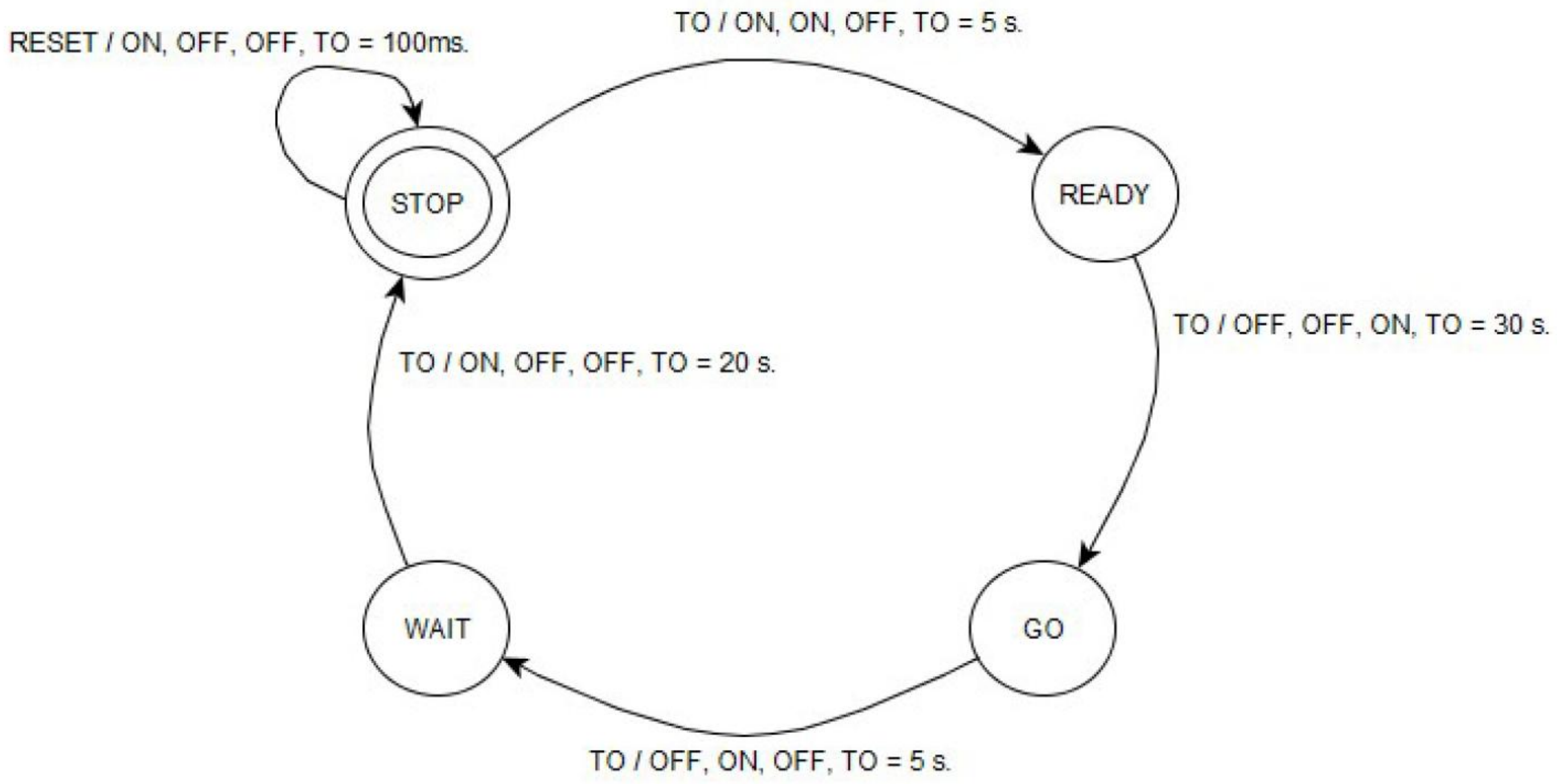
Message States

```
INT8U get_msg_state( INT8U );
/****************************************************************
***
* Input : State Memory ID
* Output : State
* Function : Get a Message State
*****************************************************************
**/
void put_msg_state( INT8U, INT8U );
/****************************************************************
**
* Input : State Memory ID, State
* Output : -
* Function : Put a Message State
*****************************************************************
*/
```

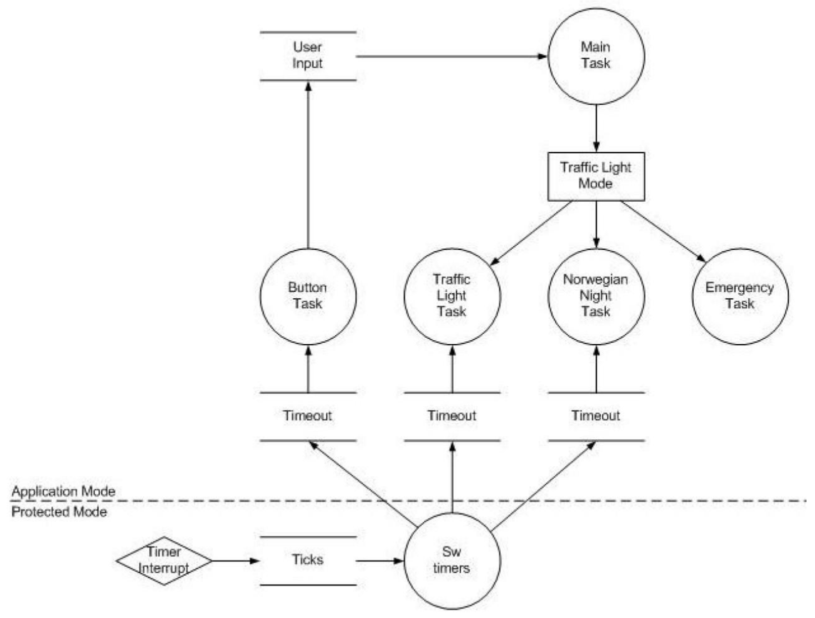
Flash task – state machine


START
TO / LED = ON, TO = 500mS
OFF
ON
RESET / LED = OFF, TO = 500mS
TO / LED = OFF, TO = 500mS

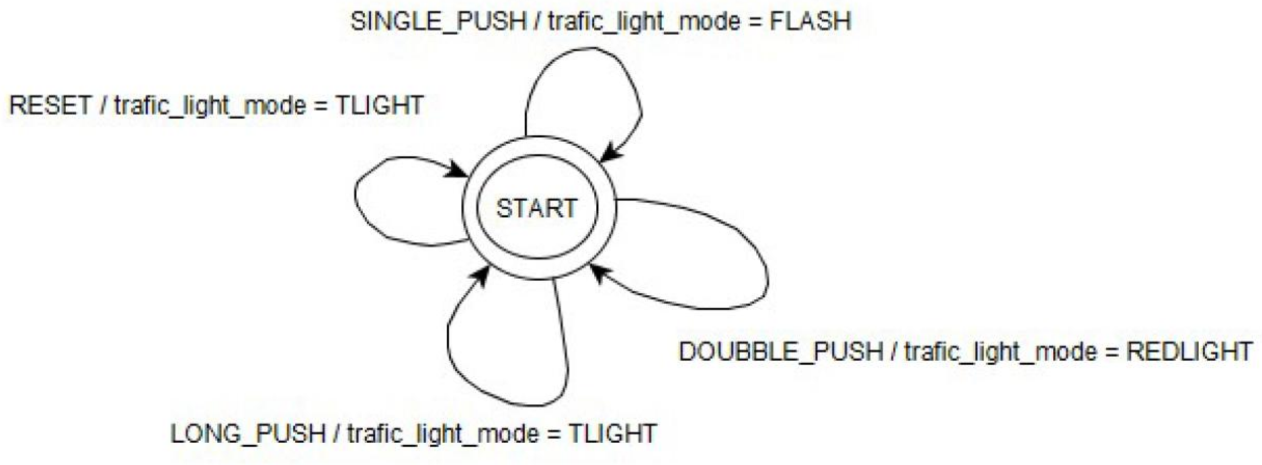
Traffic light task – state machine


RESET / ON, OFF, OFF, TO = 100ms.
TO / ON, ON, OFF, TO = 5 s.
STOP
READY
TO / OFF, OFF, ON, TO = 30 s.
TO / ON, OFF, OFF, TO = 20 s.
WAIT
GO
TO / OFF, ON, OFF, TO = 5 s.

Main task – state machine



Task model


SINGLE_PUSH / trafic_light_mode = FLASH
RESET / trafic_light_mode = TLIGHT
START
DOUBBLE_PUSH / trafic_light_mode = REDLIGHT
LONG_PUSH / trafic_light_mode = TLIGHT

State machine