SDU

Lesson03

Agenda for today

- Presentation of Assignment01
- Organizational
- State machines – solution to A01
- Build process – compiler chain
- Debugging
- Intro to the EMP board
- Intro to Assignment02
- Lab03

SDU

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

Groups

- Group sign-up sheet not editable anymore:
  - https://tinyurl.com/sduemp2026
- If you need any changes, email me
- Each group member should contribute
- Inter-group dynamics
  - Solve problems by yourself
  - If you can’t, let me know

	Group member 1	Group member 2	Group member 3
1	Magnus Meldgaard	Rune Kildahl Frederiksen	Frederik Nørregaard Wilkens
2	Magne Jacobsen	Alexander Bom Kjærbo	Lukas Hjeronimus Sørensen
3	Jonathan Balder Dietrich	Jacob Grud Agerbæk Madsen	Yunseo Cho
4	Christian Peter Kirk	Abbas Al-Ansari	Mads Bendorff Thomasen
5	Anders Christiansen	Alexander Hansen	Emil Klitgaard
6	Jeppe Rønnow	Kristoffer Nielsen	Radim Dvorák
7	Karl Soneff	Nikoline Dahl Jensen	Rasmus W. Kildenberg
8	Malte Borg-Andersen	Henriette Mindstruplund	Dawid Klasa
9	Jacob Flindt	Emil Yao Luther	August Bachmann Bjerregaard
10	Mikołaj Pułaski	Samuel Lupták	Ryan Choy
11	Niklas Nygaard	Sam Luca Hasselbalch-Gang	Jannik Vedel Olsen
12	Jakob Tietgen	Peter Ladefoged	Povl Christiansen
13			
14	Philip Kaslund	Martin Rasmussen	Gylfi Karlsson
15	Özgür Türkseven	Elif Çetinkaya	Jas Keerat Singh
16	Andrej Cvecka	Babak Rahpeima	Oleksandra Mysiuk
17	Carl Schmidt	Mads Jensen	Silas Tvingsholm
18	Ahad Asaad	Mads Thomsen	
19	Elias Alstrup	Asmus Rise	August Tranberg
20	Felix Mogensen	Dominik Łuniewski	Ahad Asaad
21	Mads Bovbjerg	Rasmus Madsen	Sebastian SNitkjær

Exam tips

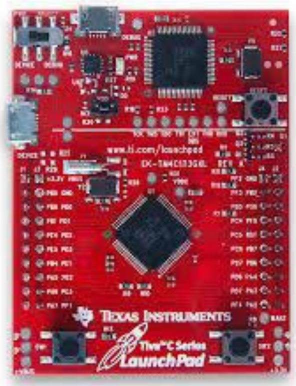
- Oral exam
- Any medical condition needs to be registered with study administration well in advance of the exam (now)
- If you have a documented medical condition, let me know before the exam
- At the exam discussion about exam conditions or why you could not prepare for the exam is not welcome other than documented medical conditions
- At the end of the exam we will give you reasoning for your grade

Rules on using AI

- AI tools should not be used to generate any of the code submitted as solutions to the assignments in this course
- AI tools can be used for education purposes
- You have to disclose the use of any AI tools for any purpose in your assignments
- In the teacher's opinion: you will learn more if you do not over-rely on AI tools in general in your education

State Machines

with Solution to Assignment 1



Assignment01

Assignment:
Write a program for the kit. The program must implement a binary counter (0-7) with the value shown at the RGB –LED of the kit, giving 8 different colors in a given sequence.
The counter must be able to count up and down. The counter must advance one step whenever the button is pushed. The direction (up/down) must be toggled when the button is double clicked. A continuous press at for more than 2 seconds must set the counter in AUTO MODE. In AUTO MODE the counter will automatically advance one step (up or down) every 200 millisecond. Any push to the button while the counter is in AUTO MODE must return the counter the normal state.

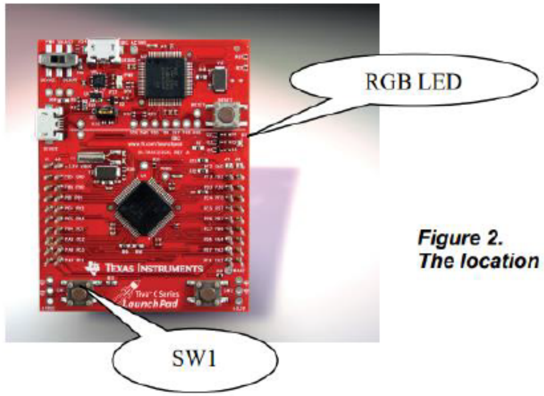


Figure 2.
The location of the RGB-LED and SW1.

RGB-LED:
Showing the binary counter on the RGB-LED will generate the following colors.

Counter value	*bit 2 red PF1*	*bit 1 blue PF2*	*bit 0 green PF3*	*color*		
0	0	0	0	LEDs turned off		
1	0	0	1	Green		
2	0	1	0	Blue		
3	0	1	1	Cyan		
4	1	0	0	Red		
5	1	0	1	Yellow		
6	1	1	0	Magenta		
7	1	1	1	White		

Main module

```
/******************************************************************************
* University of Southern Denmark
* ...
* ...
*****************************************************************************/
#include "inc\lm3s6965.h"
...
...
/***************************** Defines ********************************/
/***************************** Constants ********************************/
/***************************** Variables ********************************/
/***************************** Functions ********************************/
int main(void)
/******************************************************************************
* Input : -
* Output : Result code
* Function : The application main function.
*******************************************************************************/
{
  // Initialization
  // --------------
  Init();
  ...
  ...
  // The Super Loop.
  // ---------------
  while(1)
  {
    ...
    ...
  }
  return 0 ;
}
```

Super loop timing

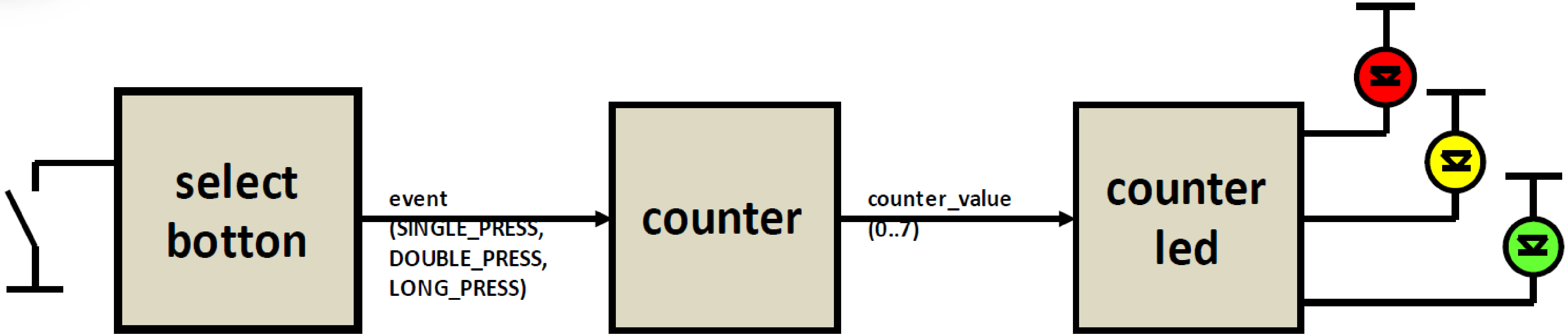
```
/****************************** Variables *******************************/
volatile INT16S ticks = 0;
/****************************** Functions *******************************/
int main(void)
/*****************************************************************************
* Input : -
* Output : Result code
* Function : The application main function.
******************************************************************************/
{
  BOOLEAN event;
  INT8U counter_value;
  // System Initialization.
  // ----------------------
  init();
  // The Super Loop.
  // ---------------
  while(1)
  {
    while( !ticks );
    // The following will be executed every 5mS
    ticks--;
  }
  return 0 ;
}
```

```
Void systick_isr(void)
/*******************************************************
* Function: See modulespecification(.h-file).
*******************************************************/
{
  // Mark timer tick.
  ticks++;
}
```

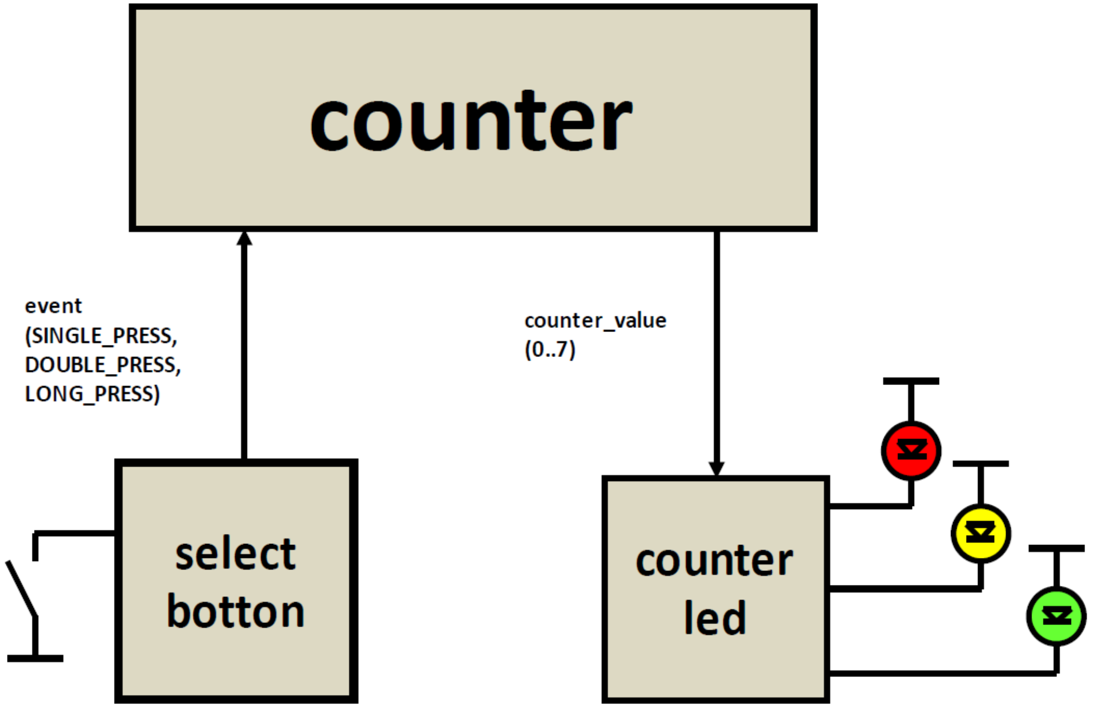
The main() function

```
int main(void)
/*****************************************************************************
* Input : -
* Output : Result code
* Function : The application main function.
*****************************************************************************/
{
  BOOLEAN event;
  INT8U counter_value;
  // System Initialization.
  // ----------------------
  disable_global_int();
  init_clk_system();
  init_gpio();
  init_systick();
  enable_global_int();
  // The Super Loop.
  // ---------------
  while(1)
  {
    // System part of the super loop.
    // ------------------------------
    while( !ticks );
    // The following will be executed every 5mS
    ticks--;
    // Application part of the super loop.
    // -----------------------------------
    event = select_button();
    counter_value = counter( event );
    counter_leds( counter_value );
  }
  return 0 ;
}
```

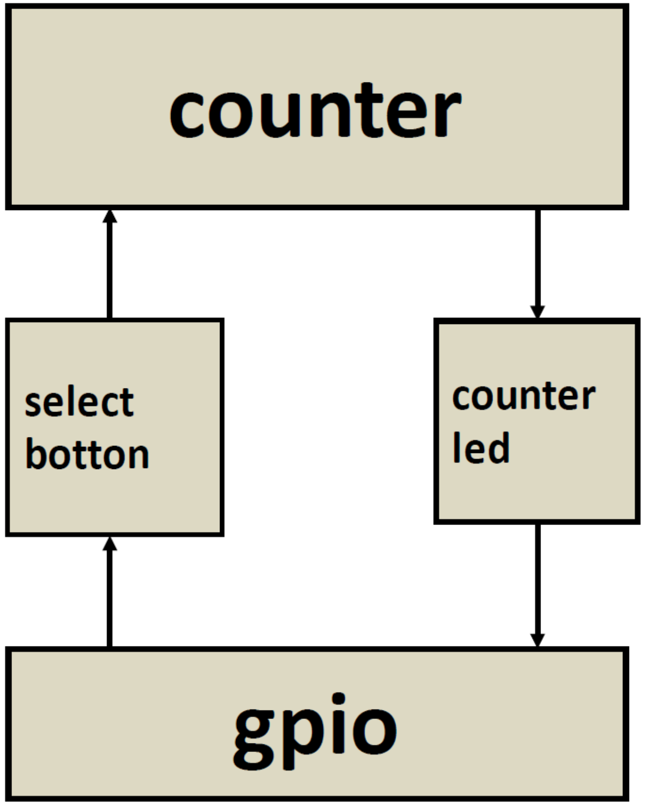
Information flow view



A layered view



A layered view



SDU

Modules

- button
  - button.c
  - button.h
- counter
  - counter.c
  - counter.h
- countled
  - countled.c
  - countled.h
- gpio
  - gpio.c
  - gpio.h

- Application
  - main.c
  - events.h

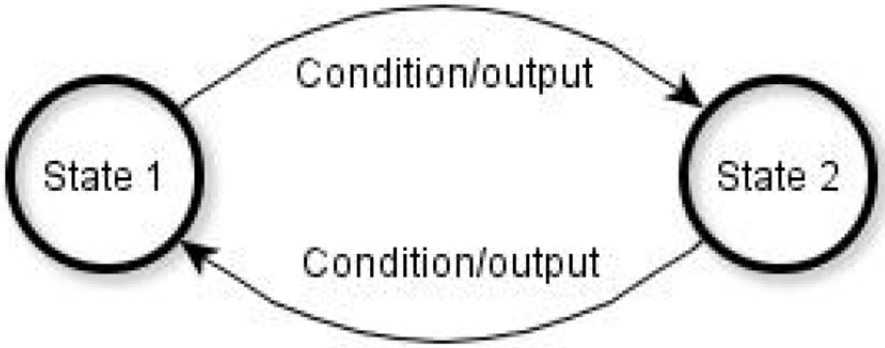
- System
  - cr_startup.c
  - systick.c
  - swtimers.h
  - emp_type.h

i_am_alive

```
// The Super Loop.
// ---------------
while(1)
{
  // System part of the super loop.
  // ------------------------------
  while( !ticks );
  // The following will be executed every 5mS
  ticks--;
  if( ! --alive_timer )
  {
    alive_timer = TIM_500_MSEC;
    GPIO_PORTD_DATA_R ^= 0x40;
  }

  // Application part of the super loop.
  // -----------------------------------
  event = select_button();
  counter_value = counter( event );
  counter_leds( counter_value );
}
```

State Machines

- Also known as Finite State Machines
- A computational model
- An abstract machine which can be in only one state at each time
- Can be represented by a state machine diagram
- Two versions:
  - Moore machine – output values are only dependent on the state
  - **Mealy machine** – output values are determined by both states and inputs
- Easily implementable in computer code using a state variable and condition statements (if, switch)

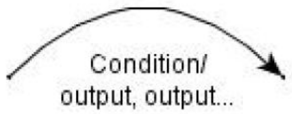


State Diagrams - Elements

A State

Start State

Transition Arrow

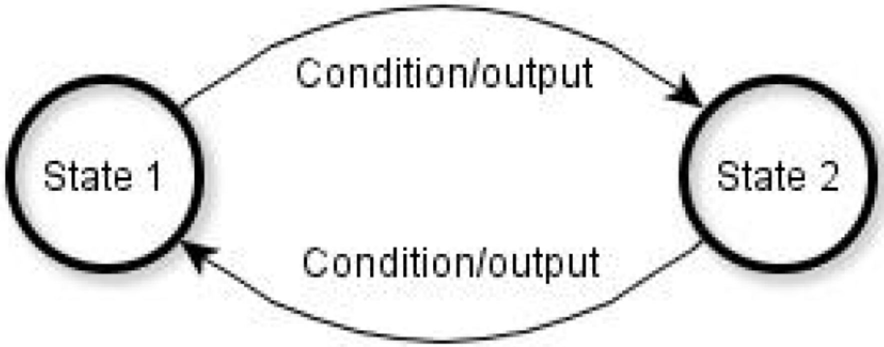


(Condition may be an event or an external state)

Transition arrows

- Conditions
  - Events
  - Special conditions
    - Timeout
    - Reset

- Outputs
  - Hardware output
  - Message to other parts of the system
  - Start timer



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

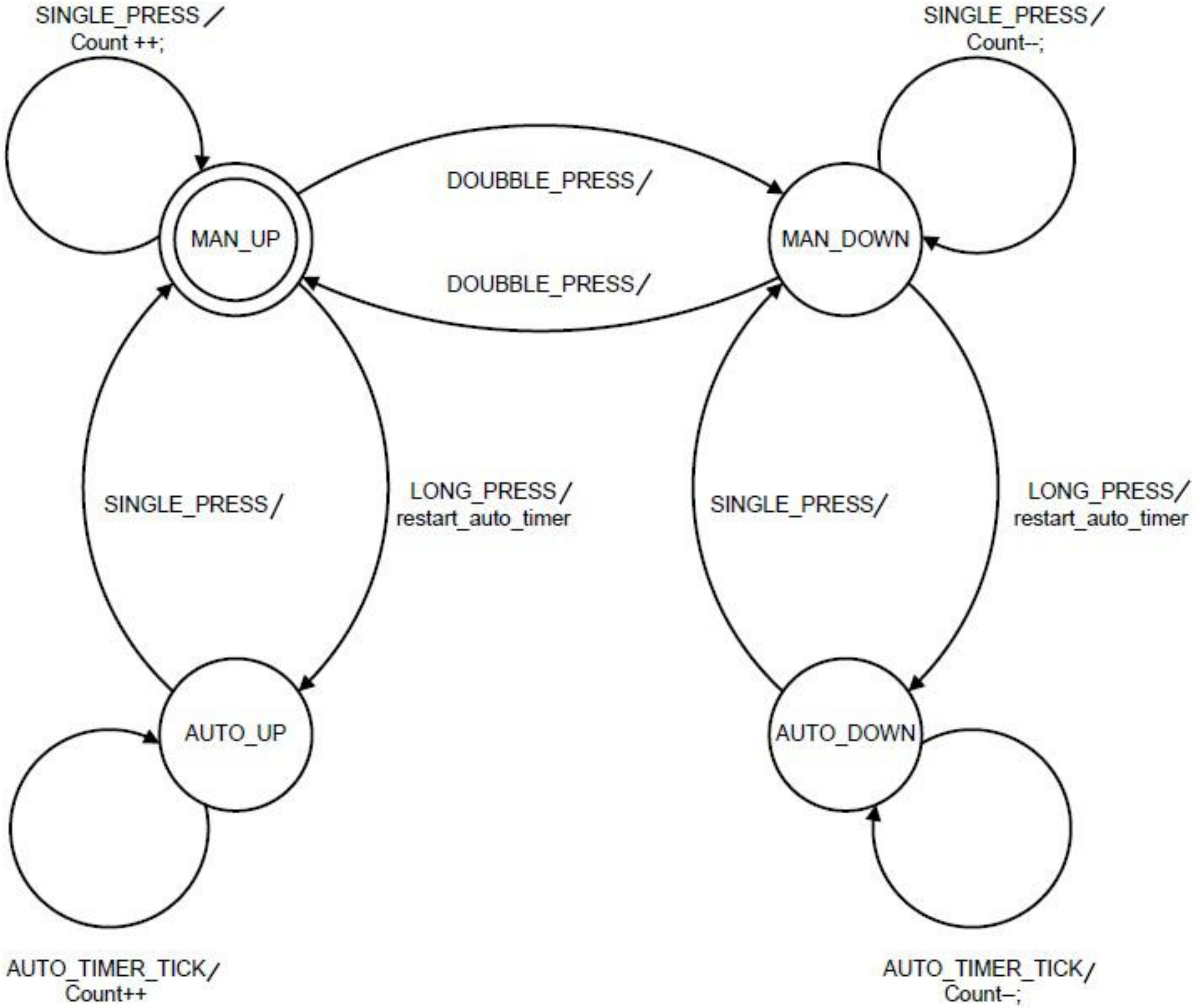
```
int8u the_door( event )
int8u event;
{
  static int8u  state  = CLOSED;
  int8u         action = NO_ACTION;

  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;

Back to Assignment1

- We implement two state machines
  - One for button operation
  - One for counting

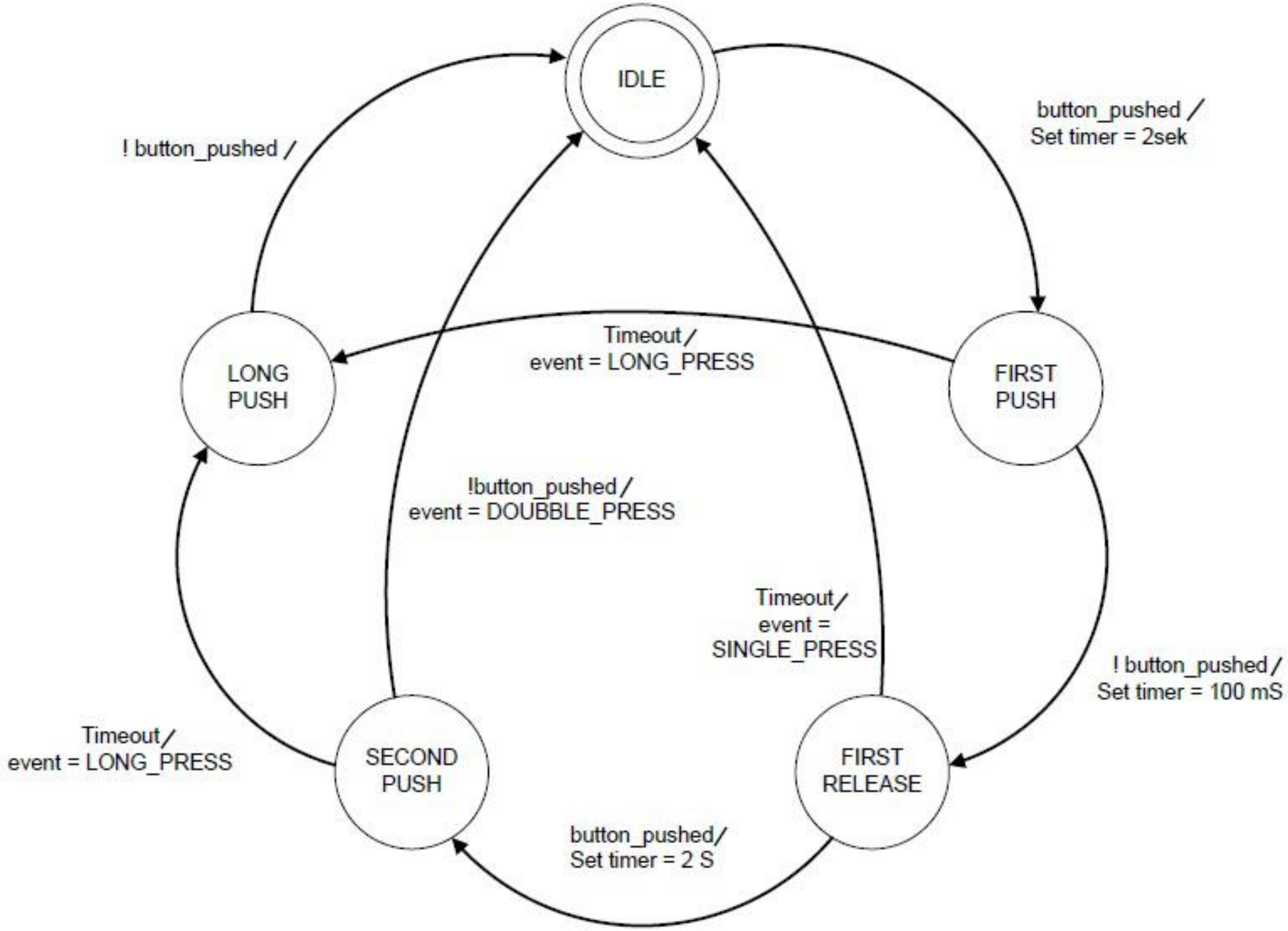
Counter state machine



Counter state machine

```
INT8U counter( INT8U event )
{
  if( event )
  {
  switch( counter_state)
  {
    case CS_MAN_UP:
    switch( event )
    {
    case BE_SINGLE_PUSH:
      counter_value++;
      break;
    case BE_DOUBBLE_PUSH:
      counter_state= CS_MAN_DOWN;
      break;
    case BE_LONG_PUSH:
      counter_state= CS_AUTO_UP;
      counter_timer= TIM_200_MSEC;
      break;
    default:
      break;
    }
    break;
  case CS_MAN_DOWN:
  switch( event )
  {
     case BE_SINGLE_PUSH:
      :::::::
```

Select button – state machine



Select button state machine

```
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:
```

Counter_leds

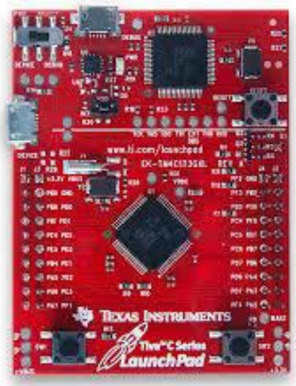
```
void counter_leds( INT8U counter_value )
/*****************************************************
**********************
* Input :
* Output :
* Function :
*****************************************************
**********************/
{
  if( counter_value & 0x01 )
    GPIO_PORTG_DATA_R &= 0xFD;
  else
    GPIO_PORTG_DATA_R |= 0x02;
  if( counter_value & 0x02 )
    GPIO_PORTG_DATA_R &= 0xFE;
  else
    GPIO_PORTG_DATA_R |= 0x01;

  if( counter_value & 0x04 )
    GPIO_PORTD_DATA_R &= 0xBF;
  else
    GPIO_PORTD_DATA_R |= 0x40;
}
```

Static variables

- Look up how static variables work in C
- They keep their values between unrelated calls to the function

```
INT8U counter( INT8U event )
/******************************************************
***********************
*   Input    : event from button press
*   Output   : counter value to set LED color
*   Function : controls counter and switches between
manual and auto mode
*******************************************************
***********************/
{
  static INT8U  counter_state =  CS_MAN_UP;
  static INT8U  counter_value = 0;
  static INT16U counter_timer = 0;
```

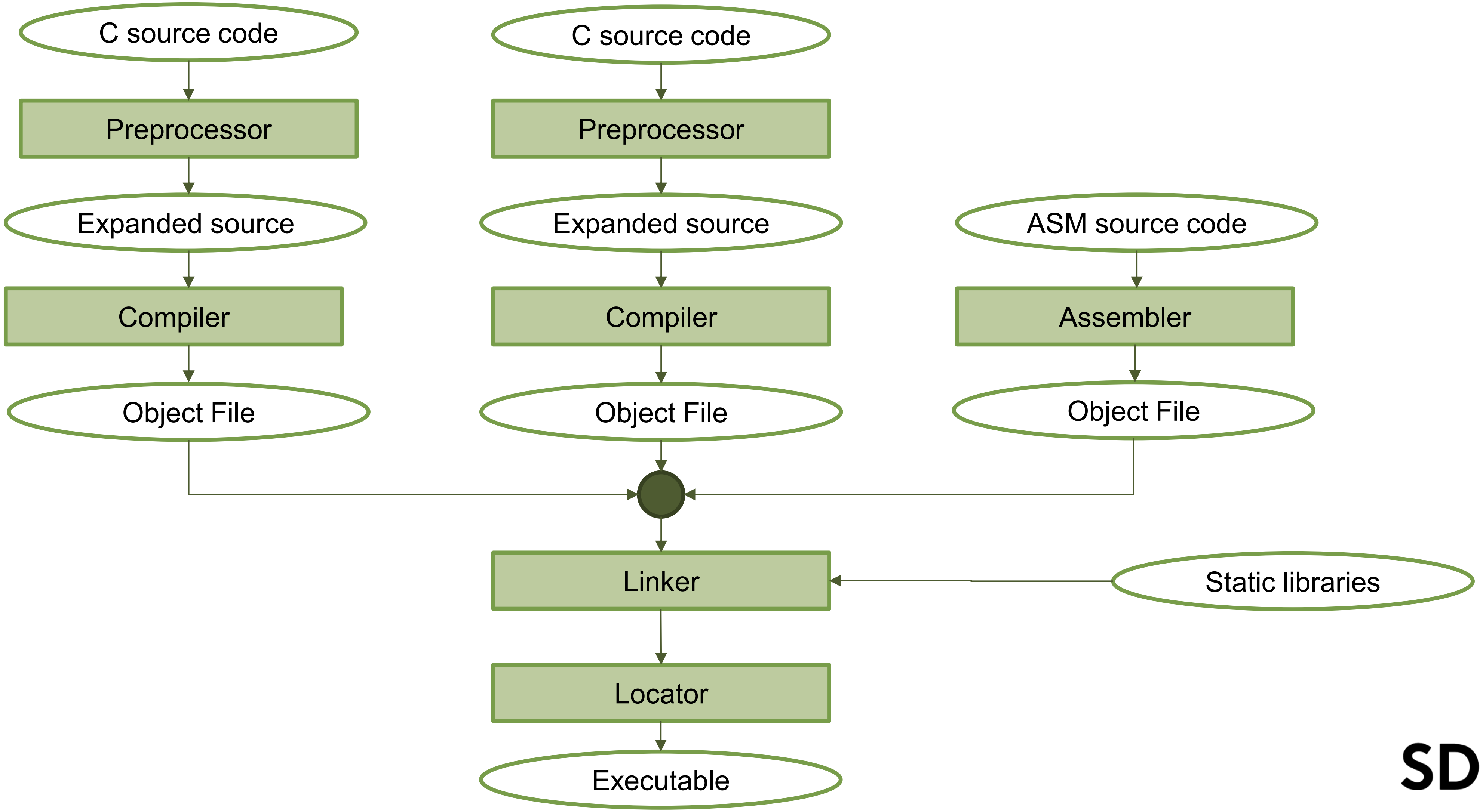
EMP Build Process
Debugging



The build process

- Hardware specific
- Compiling on PC but for TIVA controller – cross-compiler
- Handled by the integrated development environment (IDE – Code Composer Studio)
  - You don’t have to worry about it unless something goes wrong…
- We are using the IDE and not MAKE, but you can if you like it more

The build process



Preprocessor

- Handles preprocessor directives in your code (starting with #)
  - #include – copies header files into the source code
  - #define – replaces the definition with the defined values
  - #ifndef – if not defined yet, then include otherwise don’t
- Removes comments
- Combines split lines
- Passes expanded code to compiler

The #include directive

os.c

```
#include ”os.h”

INT8U variable
INT8U tasks[MAX_TASKS]
…
```

os.h

```
#ifndef OS_H
#define OS_H

#include ”compiler.h”
#include ”configuration.h”

#endif
```

compiler.h

```
#ifndef COMPILER_H
#define COMPILER_H

typedef unsigned char INT8U;
typedef signed char INT8S;
typedef unsigned int INT16U;

#endif
```

configuration.h

```
#ifndef CONFIGURATION_H
#define CONFIGURATION_H

#define MAX_TASKS 10
#define SEM_EN TRUE

#endif
```

The #include directive

os.c

os.h

compiler.h

typedef unsigned char INT8U;
typedef signed char INT8S;
typedef unsigned int INT16U;

configuration.h

#define MAX_TASKS 10
#define SEM_EN 1

INT8U variable
INT8U tasks[MAX_TASKS]

Compiler

- Translates human-readable code into machine-readable code
- It is a cross-compiler – runs on PC but for TIVA
- Groups code and data into separate structures
- Output is an object file
  - Binary file containing instructions and data
  - Not meant for execution

Linker

- Links object files together
- Some internal variables and function references have not yet been resolved
- Resolve all unresolved symbols
- Looks for references in static libraries
- Relocatable output – no memory addresses assigned to code

Locator

- Transforms a relocatable program into executable binary image
- Specific addresses for each code line
- On a PC location is taken care of by the operating system, but not on embedded processors
- Sometimes part of the linker