Development of Device Driver (TSP) using Target Language Compiler and Matlab Scripting

Presented By
Aravind Kumar Singh
Sachin Pund
Anavartha Selvan
Introduction

Every Automotive Embedded Engineer has to use Device Drivers in order to communicate with different sensors and actuators that are fed and fed out from ECU (Electronic Control Unit).

In Model Based development tool like Matlab and Simulink, it becomes handy if apart from application development we can also configure hardware drivers in Matlab environment.
What does configuring Device Driver in C Code means?

E.g. PWM Driver Code

Few points to note!
- PWM is a protocol with many pins, each configured with different settings and values.
- Configuring a PWM Driver code by itself is just the first step; it involves interacting with the actual device.
- Is there a way to test all this in isolation?
What does configuring Device Driver in C Code means?

- **Driver.c**
- **Driver.h**
- **Driver_Config.h**

1) Generally Driver Code contains these files:
   A. Driver.c  e.g. PWM.c
   B. Driver.h  e.g. PWM.h
   C. Driver_Config.h  e.g. PWM_Config.h

2. Driver.c contains hardware specific driver code for e.g.
   Driver_Init(), Driver_Read() or Driver_Write() and post
   pin direction set commands.

3. Driver.h contains code with all possible combination of
   configuration

4. User selects configuration using Driver_Config.h file
   which generally contains preprocessor commands.

5. While compiling this C code Preprocessor removes all
   code in Driver.c that are not configured in
   Driver_Config.h

---

**E.g. PWM Driver Code**

```
PWM.c
This code needs adding if driver
settings changes
```

---

**Few points to note!**

- PWM.c is populated with code that is a super
set of all configuration, which reduces
readability of C code.

- In order to interface this Driver code with
Matlab one has to create too many global
variables.

- Configuring same Driver code by MATLAB
  user is not convenient and cells for
  interaction with PWM.c code.

- Is there a way that we can set all this in Matlab?
1) Generally Driver Code contains three files
   A: **Driver.c** e.g., PWM.c
   B: **Driver.h** e.g., PWM.h
   C: **Driver_Config.h** e.g., PWM_Config.h

2. **Driver.c** contains hardware specific driver code for e.g. **Driver_Init()**, **Driver_Read()** or **Driver_Write()** and port pin direction set commands

3. **Driver.c** contains code with all possible combination of configuration

4. User selects configuration using **DriverConfig.h** file which generally contains preprocessor commands

5. While compiling this C code Preprocessor removes all code in **Driver.c** that are not configured in **Driver_Config.h**
E.g. PWM Driver Code

**PWM.c**

```c
void PWM_Init()
{
    /* Set Hardware pin for PWM */
    PWMPortPinDir = DOout;

    /* Set Control Registers */
    Control_registers1 = CR1Val;
    Control_registers2 = CR2Val;
    ........
    Control_registersN = CRNVal;
}

void PWM_Write(char DutyCycle)
{
    #ifdef ch1
        ........
        Channel1 = DutyCycle;
        ........
    #endif

    #ifdef ch2
        ........
        Channel2 = DutyCycle;
        ........
    #endif
}
```

**PWM.h**

```c
void PWM_Write(char DutyCycle);
void PWM_Init(void);
```

**PWM_Config.h**

```
#define c1
#define C0 DO
#define B0 A0
#define C1 DO
#define C2 DO
```

This code needs editing if driver settings changes.
void PWM_Init()
{
    /* Set Hardware pin for PWM*/
    PWMPortPinDir = DOut;

    /* Set Control Registers*/
    Control_registers1 = CR1Val;
    Control_registers2 = CR2Val;
    ..........
    Control_registersN = CRNVal;
}

void PWM_Write(char DutyCycle)
{
    #ifdef ch1
        ........
        Channel1 = DutyCycle;
        ........
    #endif

    #ifdef ch2
        ........
        Channel2 = DutyCycle;
        ........
    #endif
void PWM_Write (char DutyCycle);
void PWM_Init(void);

PWM_Config.h

#define ch1
#define DOut 0
#define CR1Val 0xAA
#define CR2Val 0xBB
#define CRNVal 0xNN

For changing configuration this file is edited by user
Few points to note!

PWM.c is populated with code that is super set of all configuration, which reduces readability of C code.

In order to interface this Driver code with Matlab one has to create too many global variables

Configuring same Driver code by MATLAB user is not convenient and calls for interaction with PWM.c code.

Is there a way that we can set all this in Matlab?
Configuring Hardware Driver in Matlab

Prerequisites for this type of development:
1. Target Language compiler adopting
2. Understanding S-functions and S-function builder
3. Simulink
4. Embedded C Knowledge
5. Matlab Scripting

Steps:
Step 1: Overview and Hardware Setup
Step 2: Configuration
Step 3: Interface Builder
Step 4: Driver Interface Blocks
Step 5: Output

Details of Implementation:
Implementation requires following components:
1. Configuration Blocks: To generate PWM_Config, PWM 2, and PWM 3
2. Driver Interface Blocks: To establish interface between the driver and application code in Matlab

Points to Note:
- Ensure that all hardware devices are properly connected and configured.
- Validate the system's response and adjust parameters as necessary.
Prerequisites for this type of development

1. Target Language compiler scripting
2. Understanding S-functions and S-Function Builder
3. Masking
4. Embedded C Knowledge
5. Matlab Scripting
Details of Implementation

Implementation requires following components

1) **Configuration Block**: To generate PWM_Config.h, PWM.c and PWM.h
2) **Driver Interface Blocks**: To establish interface between the driver and application code in Matlab

![PWM Driver Example](image-url)
Steps

Step 1: Use S-function Block to create Config Block
- This step shows the S-function block creating the ConfigBlock.
- The block is configured with specific parameters and settings.

Step 2: Use S-function block
- This step demonstrates the use of an S-function block within the model.
- The block's parameters and functionalities are highlighted.

Step 3: Mask this S-function
- Closeout of Masking in S-function block in previous step.
- The masking process is shown with relevant screenshots and diagrams.

Step 4: Edit PWMConfig.tlc
- In this step, open the PWMConfig.tlc file to perform necessary changes.
- The file contains model parameters and settings for PWM configuration.

Step 5: Creating Driver interface Block
- To create driver interface block (Step 2), 2 steps to create PWM driver interface block are explained:
- The block's design and implementation are illustrated with diagrams.

Output

PWM Driver Example
Step 1: Use S function Builder to create Config Block

This step gives you the following files
1) PWMConfig.c
2) PWMConfig_wrapper.c
3) PWMConfig.tlc

S function Builder is build with no Input/Output ports and parameters
Step 2: Use S-Function block

In this step take S-function block in new model and assign PWMConfig.c to this block as shown
Step 3: Mask this S-function

Create GUI by Masking S-function block in Previous Step

This configuration Block takes hardware dependent values from user e.g., Prescalar Value, Clock Source and code configuration
## Mask Editor: S-Function Configuration Parameters

### Dialog parameters

<table>
<thead>
<tr>
<th>#</th>
<th>Prompt</th>
<th>Variable</th>
<th>Type</th>
<th>Evaluate</th>
<th>Tunable</th>
<th>Tab name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prescalar Select for Clock A</td>
<td>PWMPRCLKA_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>General Settings</td>
</tr>
<tr>
<td>2</td>
<td>Prescalar Select for Clock B</td>
<td>PWMPRCLKB_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>General Settings</td>
</tr>
<tr>
<td>3</td>
<td>PWM Polarity Set Channel 0</td>
<td>PWMPOLCh0_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>4</td>
<td>PWM Polarity Set Channel 1</td>
<td>PWMPOLCh1_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>5</td>
<td>PWM Polarity Set Channel 2</td>
<td>PWMPOLCh2_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>6</td>
<td>PWM Polarity Set Channel 3</td>
<td>PWMPOLCh3_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>7</td>
<td>PWM Polarity Set Channel 4</td>
<td>PWMPOLCh4_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>8</td>
<td>PWM Polarity Set Channel 5</td>
<td>PWMPOLCh5_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>9</td>
<td>PWM Polarity Set Channel 6</td>
<td>PWMPOLCh6_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>10</td>
<td>PWM Polarity Set Channel 7</td>
<td>PWMPOLCh7_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SetPWMPolarity</td>
</tr>
<tr>
<td>11</td>
<td>PWM Clock Select Channel 0</td>
<td>PWMCCLKCh0_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SelectClock</td>
</tr>
<tr>
<td>12</td>
<td>PWM Clock Select Channel 1</td>
<td>PWMCCLKCh1_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SelectClock</td>
</tr>
<tr>
<td>13</td>
<td>PWM Clock Select Channel 2</td>
<td>PWMCCLKCh2_Config</td>
<td>popup</td>
<td></td>
<td></td>
<td>SelectClock</td>
</tr>
</tbody>
</table>

### Type-specific options

- No type-specific options

### Generic options

- In dialog:
  - Enable parameter
  - Show parameter

### Dialog callback:

[Enter callback details here]
Step 4: Edit PWMConfig.tlc

In this step we edit this PWMConfig.tlc to perform following things:
A) To read Mask Parameters
B) With TLC basic I/O File handling functions generate
PWM_config.h, PWM.c and PWM.h using parameter from step A.

PWMConfig.tlc will contain many If and Else Statement that will generate code with configuration that matlab user sets using GUI. Code is similar to Driver_config.h

Functions used by TLC are: LibCreateSourceFile(), LibCreateHeaderFile() etc.

This PWMConfig.tlc file is responsible for generating driver files without any #ifdef, #ifndef commands as seen in earlier slider
Step 5: Creating Driver Interface Block

To create driver interface block perform Steps 1, 2, 3 again to create PWM_Write

Here PWM_Write.tlc is modified in order to place function call only in code with any input or output if specified in GUI (i.e., created by mask) for e.g., PWM channel in case of PWM Driver
% generated code.

%function BlockTypeSetup(block, system) Output
%openfile buffer
   /* Defines Channel No*/
#define Channel0 0
#define Channel1 1
#define Channel2 2
#define Channel3 3
#define Channel4 4
#define Channel5 5
#define Channel6 6
#define Channel7 7
%closefile buffer
%<LibCacheDefine(buffer)>
%endfunction

% Function: Outputs  =========================================================
% Purpose:
%    Code generation rules for mdlOutputs function.
%function Outputs(block, system) Output
%   %assign pu0 = LibBlockInputSignalAddr(0, "", ",", 0)
%   %assign pu_width = LibBlockInputSignalWidth(0)
%   %assign ::PWMFreqSet = 0
%   %assign ChannelNo = FEVAL("get_param", block.Name, "ChannelNo")
%   %assign ChannelFrequency = FEVAL("get_param", block.Name, "ChFreq")
%   PWM_Out(<ChannelNo>, <pu0>);
%endfunction
Sink Block Parameters: S-Function1

S-Function (mask)

Parameters

Select Channel: Channel1
TLC Execution Process

Output

Final Code Generated by TLC

PWM.c
/* code generated by PWMConfigBlk.tlc */
void PWM_Init()
{
    /* Set Hardware pin for PWM */
    PWMPortPinDir  = DoOut;
    Control_register1 = CR1Val;
    Control_register2 = CR2Val;
    Control_register3 = CRNVal;
}

void PWM_Write(char DutyCycle)
{
    /* code without stall */
    Channel1 = DutyCycle;
}

PWM.h
/* Prototype declaration */
void PWM_Init();
void PWM_Write(char DutyCycle);

PWM_Config.h
/* This code is generated by PWMConfigBlk.tlc */
#define DoOut 0
#define CR1Val 0xAA
#define CR2Val 0xBB
#define CRNVal 0xNN

Prezi
TLC Execution Process

1. Masking
   Set parameter for driver configuration

2. Simulink
   1. Application
   2. Driver Config
   3. Driver Interface Block

3. Build

4. Adjust System Configuration

5. Simulink Coder
   1. Target file

6. Block Target
   1. Driver Config
   2. Driver Interface

7. System Target File
   File Process file

8. TLC function Libraries

9. Target Language Compiler

10. C-File
    1. Model.C
    2. Driver.C
    3. Driver_Config.h
    4. Driver.h
    5. Main.C

11. Cross Compiler

12. Execute file
Final Code Generated by TLC

PWM.c
/* code generated by PWMConfigBlk.tlc */
void PWM_Init()
{
  /* Set Hardware pin for PWM */
  PWMPortPinDir = DOut;
  Control_register1 = CR1Val;
  Control_register2 = CR2Val;
  ..........  
  Control_registerN = CRNVal;
}

void PWM_Write(char DutyCycle)
{
  /* code without #if */
  Channel1 = DutyCycle;
}

PWM.h
/* Prototype declaration */
void PWM_Init();
void PWM_Write(char DutyCycle);

PWM_Config.h
/* This code is generated by PWMConfigBlk.tlc */
#define DOut 0
#define CR1Val 0xAA
#define CR2Val 0xBB
#define CRNVal 0xNN


Points to Note!

Whenever model build is initiated PWMConfig.tlc and PWM_Write.tlc is automatically executed and generates PWM.c, PWM.h and PWM_Config.h

S-function blocks that are created in these steps serves as **Target Support Packages** (Device Driver blocks) which are **hardware specific**. Which can be **reused** in different for same hardware platform.
BSP - Board Support Package

**Introduction to BSP**

- BSP adds value to Driver Stacks by enabling Application Engineer to efficiently select inputs based on Sensor names rather than File/Channel No.
- This is done to provide readability and prevent errors due to incorrect pin allocation in model.
- BSP hides board hardware details to Matlab Users.

**BSP in a Nutshell**

- We have created BSP using ClassLib in Matlab.
- Each object of BSP is a class in an instance of different board.
- Superclass contains variables for each bus components along with methods.
- Information of Bus/Channel busses and identifier (i.e., Sensor name) is stored in lookup tables.
- Methods of each subcomponents within the driver hides with a corresponding identifier in model so that sensor name or identifier is seen other than Bus/Channel busses.
Introduction to BSP

BSP adds value to Driver Blocks by enabling Application Engineer to efficiently select Inputs based on Sensor names rather than Pins/Channel No.

This is turns provides readability and prevent errors due to mismatch pin allocation in model. It hides board hardware details to Matlab Users.
BSP in a Nutshell

We have created BSP using Classes in Matlab

Each object of BSP super class is an instance of different board

Superclass contains subclass for each sub components along with methods

Information of Pin/channel No w.r.t Identifier (i.e., Sensor Name) is stored in Lookup tables

Methods of each subcomponents updates the driver blocks with corresponding identifier in mask so that sensor name or identifier is seen other than Pin/Channel No
Conclusion

A graphical interface provides convenience to Matlab user to configure driver

Only required code gets generated. No dead code appear in C file

Function call method is used instead of global variable

BSP adds value to Device Driver Configuration by adding identifier instead of pin/channel no
Thank you

Questions?