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

Driver.c
Driver.h

Driver.c contains: main
Driver.h contains: PWM

Few points to note!

PWM is configured with value that is required set of all configurations, which mandates queuing of data

Is there a way that we can put all this in external?
What does configuring Device Driver in C Code means?

E.g. PWM Driver Code

PWM.c

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 pass 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

Few points to note!

PWM.c is populated with code that is subset 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**
Example PWM Driver Code

**PWM.c**

```c
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
}
```

**This code needs editing if driver settings changes**

**PWM.h**

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

**PWM_Config.h**

```c
#define c1
#define c2
#define c3
#define c4
#define c5
#define c6
#define c7
#define c8
#define c9
#define c10
```
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
}
PWM.h

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

PWM_Config.h

#define ch1
#define DOut 0
#define CR1Val 0xA1
#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 adoption
2. Understanding S-functions and S-Function Block
3. Mocking
4. Embedded C Knowledge
5. Matlab Scripting

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

Steps

Steps:
- Step 1: Configuration Block
- Step 2: PWM 1
- Step 3: PWM 2
- Step 4: Driver Interface Block
- Output

Points to Note:
- Ensure that the configuration block is properly set up.
- Verify that PWM 1 and PWM 2 are correctly configured.
- Check the driver interface block to ensure proper communication.
- Review the output to confirm successful implementation.
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
Steps

Step 1: Use S-function Builder to create Config Block

This step gives you the
architecture, block
for PWM(DutyCycle in %)
1) PWM driver example
2) PWM/Config example
3) PWM/Config example

Step 2: Use S-function block

In this step, the S-function block is created and the PWMConfig block is used.

Step 3: Mask this S-function

The S-function block is masked to prevent the user from editing the block.

Step 4: Edit PWMConfig.m

In this step, the PWMConfig.m is modified and the PWMConfig parameter is set.

Step 5: Creating Driver Interface Block

To create the driver interface block, follow the steps below:
1) Replace the S-function block with a custom function block.
2) Ensure the function block is connected to the PWMInterface block.

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*
<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:**

[Unmask] [OK] [Cancel] [Help] [Apply]
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

[Buttons: OK, Cancel, Help, Apply]
TLC Execution Process

Final Code Generated by TLC

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

  Control_register1 = CRNVal;
}

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

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

// This code is generated by PWMConfigBlk.tlc */
#define DOout 0
#define CR1Val 0xAA
#define CR2Val 0xBB
#define CRNVal 0xNN
```
TLR Execution Process
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 0xFF

Prezi
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 Blocks by enabling Application Programmer to efficiently write inputs based on Sensor names rather than File/Channel No.

This feature provides readability and prevents errors due to mismatched pin allocation in model. It hides board hardware details to Matlab Users.

BSP in a Nutshell

- We have created BSP using Matlab's in-house tool.
- Each object of BSP comprises a driver block with an instance of different board.
- Superblocks contains isolates for each I/O component along with their details.
- Information of Files/Channels is saved in separate folder.
- Method of each subcomponents within the driver blocks with a corresponding identifier in Matlab so that sensor name or identifier is seen other than File/Channel Not.
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
Development of Device Driver (TDL) using Target Language Compiler and Matlab Scripting

Presented By
Anirudh Kumar Singh
Sachin Pundalik
Anavartha Selvan

Thank you

Questions?