A Basic Approach to Embedded Software Architecture

Salvador Almanza-Garcia
Special Projects - Embedded Software
Vector CANtech, Inc., Novi MI, USA
IEEE SEM GOLD Vice-Chair

Embedded Systems Workshop 2012
IEEE Computer Society
Southeastern Michigan Section
October 13th, 2012
Objectives

- To introduce basic concepts and examples of software architecture applied to embedded system design and development

- To provide a basic point of view about embedded software architecture to students and other developers who are not yet involved in this product development culture

- To establish the important concepts and design methodology during undergraduate/graduate embedded systems courses

Note:

- The present material is intended for the audience attending the embedded systems workshop at Oakland University (mainly students). The content respect to methodology and source code is based on Author previous experience and current projects related to academics, it is not related to Vector CANTech Inc. products and/or tools
Definitions

Software Architecture

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. [Bass, Clements & Kazman, 2003]

- Collection of software components that follows an organized structure, and describes the overall system and its components behavior from a high-level design perspective

Embedded Software Architecture

- Structure and organization of multiple software components through different abstraction layers that intends to provide hardware independence, maximizes code reusability and propagates component behaviors, between multiple platforms of purpose-specific embedded computers
“All architecture is design, but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change” [Grady Booch]
Definitions

Abstraction

- Simplified view of a system containing only the details important for a particular purpose [Berzins & Luqi, 1991]

Embedded Software Abstraction

- Design methodology used to hide hardware architecture details from the application software domain by the isolation and encapsulation of relevant parameters that describe the behavior of an specific hardware entity, in order to facilitate software component reusability and portability

Software Component

- In software system, a software component is an entity with well defined behavior and interacts with other components and modules within the system
Definitions

Software Interface
- A mechanism used by a software component or module to interact with the external world (i.e., analog/digital signals, RF) and other software components

Coupling
- Degree of dependency between different software components within a system

Cohesion
- Measures the degree of relationship between elements within a software component.
Reasons for Embedded Software Architecture

- The increasing complexity of system requirements as consequence of technology advancements in semiconductor industry

- Complex requirements critically impact the product life cycle. It is difficult to satisfy time-to-market demands (reduce development time and cost)

- Optimize and speed-up software development, without compromising safety, robustness and quality of the software components

- Improve software component reusability through multiple hardware platforms
Layered Architecture

Software Architecture Structure

- Application
- HAL System Interface
- Middleware and System Management
- Hardware Abstraction
- MCU Peripheral Drivers
- MCU Hardware
- External Drivers
- OS
Software Layers Description

MCU Peripheral Drivers

- Internal device drivers
- Hardware access to MCU peripherals
- Provide MCU low-level abstraction
- Hardware dependence is high, therefore, reuse is limited at this level
- Provide standard interfaces used by abstraction, OS and external driver layers
Software Layers Description

Hardware Abstraction Layer (HAL)

- Provides access to MCU hardware features through peripheral interfaces
- Hides hardware details not relevant to upper software layers
- Interfaces with MCU and external drivers in the low level side, and with HAL signal interface at the upper side
Software Layers Description

Middleware and System Management

- Facilitate the interaction between application components and other modules and/or components within the system:
  - Graphics Library
  - Networking
  - File Systems
  - Databases
  - Other Middleware components, i.e., off-the-shelf components

- Provides system management
  - Power Management
  - Memory management
  - Diagnostics

- Due to overhead, it is an optional layer
Software Layers Description

External Drivers

- Implements direct hardware access to external devices through MCU peripherals
- Meet all functional and timing requirements of the external devices

Examples:
- EEPROM (I2C™, SPI™, Microwire™, etc)
- External ADCs (i.e. Delta-Sigma high-resolution converters)
- Sensors and actuators
Software Layers Description

HAL System Interface

- Provides to the application one more level of abstraction and hardware independence
- Translates logical signals into a meaningful format for the application
- Facilitates the communication between application software components and/or lower layer modules
- It is application specific
- Due to overhead, it is an optional layer
Software Layers Description

Application Layer

- Product specific functions

- Contains the software components that implements the desired functionality (unique) for a specific embedded computer system

- A high-level design methodology ignores the details of the hardware

- Reusability of application components strongly depends in the availability and efficiency of lower layers
Software Layers Description

OS Layer

- Provides support for multi-tasking

- Task scheduling and synchronization

- If real-time OS (RTOS)
  - Context switching
  - Task preemption
  - Interrupt management
Example - Software components and module interaction

<table>
<thead>
<tr>
<th>Window Switch Component</th>
<th>Window Manager Component</th>
<th>Window Ctrl Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ap_getswitchState ()</td>
<td>Ap_WndwRequest(wndw, dir)</td>
<td>Activate_Window (wndw, dir)</td>
</tr>
<tr>
<td>Ap_SwitchEventCmd(event)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

HAL System Interface

OS

Interface

Hw Abstraction

Motor Driver

GPIO

MCU Peripherals

Window Switch

Window Motor
Example – Device Driver Module

Methodology:

- Understand device characteristics (internal or external)
  - Read device user manual, datasheet and application notes
  - Become familiar with the device (within the family)

- Identify and extract the characteristics that describe the device behavior

- Define component data types (needed for encapsulation)

- Design software component structures and interfaces

- Component implementation and testing

- Incorporate software component into software libraries
Example – MCU SPI™ Device Driver Module

- Data Abstraction – Extract common device characteristics
  - Operation mode (Master / Slave)
  - Data width
  - Clock polarity
  - Clock edge
  - Baudrate
  - Device selection (chip select - CS)

- MCUs can incorporate additional features, not part of SPI™ standard, that need to be addressed appropriately if used (not within the scope of this example)

- Data processing configuration
  - Enable/disable interrupts
  - Polling mode
Example – MCU SPI™ Device Driver (simple case)

- Define Interfaces
  - Driver Initialization
    - Init port pins
    - Init SPI physical channel
    - Init SPI interrupts (enable/disable)
  - Data transmission
    - Tx character
    - Rx character
    - Tx data buffer
    - Rx data buffer
Module Example – MCU SPI™ Device Driver (code snippets)

Module Data types

typedef struct Spi_hldChnlCfg
{
    Spi_hldChnlIdType channelId;
    Spi_hldCfgFlagType cfgFlags;
    Spi_hldIntCfgFlagType intCfgFlags;
    Spi_hldBaudRateType baudRate;
   Spi_hldCallbackCfgType cfgFnctnPtr;
    Spi_hldCallbackIntCfgType intCfgFnctnPtr;
} Spi_hldChnlCfgType;

typedef struct Spi_hldCsCfg
{
    Spi_hldcsIndexType csId;
    Spi_hldCsEnType csEnable;
    Spi_hldCsPolType csPolarity;
    Spi_hldCsDlyEnType csDelayEn;
    Spi_hldCsDlyTimeType csDelayTime;
    Spi_hldCsPortType csPort;
} Spi_hldCsCfgType;

Module Configuration

const Spi_hldCsCfgType Spi_hldCsConfig[SPI_MAX_CS_NUMBER] =
{
    {SPI_CS0, SPI_CS_ENABLE, SPI_CS_POL_LOW, SPI_CS_DLY_DISABLE, 0, EE_CS},
    {SPI_CS1, SPI_CS_DISABLE, SPI_CS_POL_LOW, SPI_CS_DLY_DISABLE, 0, ADS124x_CS},
    {SPI_CS2, SPI_CS_DISABLE, SPI_CS_POL_LOW, SPI_CS_DLY_DISABLE, 0, GPIO_NULL},
    {SPI_CS3, SPI_CS_DISABLE, SPI_CS_POL_LOW, SPI_CS_DLY_DISABLE, 0, GPIO_NULL}
};
Module Example – MCU SPI™ Device Driver (code snippets)

Module Configuration (Cont...)

```c
const Spi_hldChnlCfgType Spi_hldChannelConfig[SPI_MAX_CHANNEL] =
{
    { 
        SPI_CHANNEL_1,       /* MCU SPI Channel Id */
        SPI_CFG_CHANNEL1,    /* Configuration Flags */
        SPI_INT_CFG_CHANNEL1,/* Interrupt Flags */
        SPI_BAUD_CHANNEL1,   /* Baudrate */
        Mcu_lldSpiCfg,
        /* Channel Configuration Function Pointer */
        Mmu_lldSpiIntCfg,    /* Interrupt Configuration Function Pointer */
    },
    { 
        SPI_CHANNEL_2,       /* MCU SPI Channel Id */
        SPI_CFG_CHANNEL2,    /* Configuration Flags */
        SPI_INT_CFG_CHANNEL2,/* Interrupt Flags */
        SPI_BAUD_CHANNEL2,   /* Baudrate */
        NULL_FUNCTION_SPI_CFG_PTR, /* Channel Configuration Function Pointer */
        NULL_FUNCTION_SPI_INT_PTR /* Interrupt Configuration Function Pointer */
    },
    #if ((TARGET_PROCESSOR == CPU_PIC32MX6X_MICROCHIP) ||
         (TARGET_PROCESSOR == CPU_PIC32MX7X_MICROCHIP))
    { 
        SPI_CHANNEL_3,       /* MCU SPI Channel Id */
        SPI_CFG_CHANNEL3,    /* Configuration Flags */
        SPI_INT_CFG_CHANNEL3,/* Interrupt Flags */
        SPI_BAUD_CHANNEL3,   /* Baudrate */
        NULL_FUNCTION_SPI_CFG_PTR, /* Channel Configuration Function Pointer */
        NULL_FUNCTION_SPI_INT_PTR /* Interrupt Configuration Function Pointer */
    },
    #endif
    { 
        SPI_CHANNEL_4,       /* MCU SPI Channel Id */
        SPI_CFG_CHANNEL4,    /* Configuration Flags */
        SPI_INT_CFG_CHANNEL4,/* Interrupt Flags */
        SPI_BAUD_CHANNEL4,   /* Baudrate */
        Mmu_lldSpiCfg,
        /* Channel Configuration Function Pointer */
        Mmu_lldSpiIntCfg,    /* Interrupt Configuration Function Pointer */
    }
};
```

© 2012. Vector CANtech, Inc.. All rights reserved. Any distribution or copying is subject to prior written approval by Vector.
Module Interfaces - Initialization

```c
void Spi_hldChannelInit(const Spi_hldChnlCfgType *spiCfgPtr)
{
    Spi_hldChnlCfgType      cfgPtr = (Spi_hldChnlCfgType *)spiCfgPtr;
    uint8                   cfgIdx;

    if(NULL != cfgPtr)
    {
        for(cfgIdx = 0; cfgIdx < SPI_MAX_CHANNEL; cfgIdx++)
        {
            if(NULL_SPI_FUNC_CFG_PTR != cfgPtr[cfgIdx].cfgFnctnPtr)
            {
                cfgPtr[cfgIdx].cfgFnctnPtr(
                    &cfgPtr[cfgIdx].cfgFd,  // cfgPtr[cfgIdx].channelId,
                    &cfgPtr[cfgIdx].cfgFlags,
                    &cfgPtr[cfgIdx].baudRate);

                Spi_SetStatusFlag(cfgPtr[cfgIdx].channelId, SPI_CHANNEL_ENABLE);
                Spi_SetStatusFlag(cfgPtr[cfgIdx].channelId, SPI_TX_READY);
            }
            if(NULL_SPI_INT_FUNC_CFG_PTR != cfgPtr[cfgIdx].intCfgFnctnPtr)
            {
                cfgPtr[cfgIdx].intCfgFnctnPtr(
                    &cfgPtr[cfgIdx].channelId,
                    &cfgPtr[cfgIdx].intCfgFlags);
            }
        }
    }

    Spi_hldCsInit();
}
```

© 2012. Vector CANtech, Inc., All rights reserved. Any distribution or copying is subject to prior written approval by Vector.
Module Interfaces – Rx Buffer

```c
void Spi_hldGetBuffer(Spi_LogicChannelIdType id, Spi_DataWidthType *destBuffer, uint16 size)
{
    uint16 spi = Spi_hldGetChannelId(id);
    uint16 cs  = Spi_hldGetCs(id);
    Spi_DataWidthType dummy = 0;

    Spi_hldSetStatusFlag(spi, SPI_RX_IN_PROGRESS);
    Spi_hldClearStatusFlag(spi, SPI_RX_READY);

    /* Start Communication */
    Spi_hldAssertCs(cs);

    if(NULL != destBuffer)
    {
        while(size--)
        {
            /* Keep Clock active during reception */
           Spi_hldTransmitChar(spi, dummy);

            *destBuffer++ = Spi_hldReceiveChar(spi);
        }
    }

    /* End Communication */
    Spi_hldDeAssertCs(cs);

#if !defined(SPI_INTERRUPT_MODE)
    Spi_hldSetStatusFlag(spi, SPI_RX_READY);
    Spi_hldClearStatusFlag(spi, SPI_RX_IN_PROGRESS);
#endif
}
```
Module Interfaces – LLD SPI Rx interface

```c
void Mcu_lldSpiTxChar(Mcu_hwSpiChnlIdType id, uint16 data)
{
    #if (TARGET_PROCESSOR == CPU_PIC32MX3X_MICROCHIP)
    if((SPI_CHANNEL_1 != id) || (SPI_CHANNEL_2 != id))
    {
        Mcu_SetErrorFlag(MCU_SPI_CHANNEL_ID_NOT_FOUND);
    }
    else
    {
        SpiChnPutC(id, data);
    }
    #else
        SpiChnPutC(id, data);
    #endif

}
```
Example – Module interaction

Inter-module interaction - EEPROM Module (External device)
Example – Module Interaction (code snippets)

Inter-module interaction - EEPROM Module (External device)

EEPROM Driver Initialization

```c
const EEDRV_cfgType EEDRV_Config[EEPROM_MAX_NUMBER] =
{
    {
        EEDRVDEVICE1,
        EEDRV_SPI,
        EEDRV_SPI_CHANNEL1,
        EEDRV_256KBIT,
        EEDRV_PAGE_SIZEDEVICE1
        /* EEPROM Device ID */
        /* EEPROM Device Interface Type */
        /* EEPROM Comm Channel */
        /* EEPROM size */
        /* EEPROM Page Size */
    },
    {
        EEDRVDEVICE2,
        EEDRV_I2C,
        EEDRV_I2C_CHANNEL3,
        EEDRV_SIZE_512KBIT,
        EEDRV_PAGE_SIZEDEVICE2
        /* EEPROM Device ID */
        /* EEPROM Device Interface Type */
        /* EEPROM Comm Channel */
        /* EEPROM size */
        /* EEPROM Page Size */
    }
};
```
Inter-module interaction - EEPROM Module (External device)

EEPROM Driver – Read Memory Block

```c
void EEDRV_readBlock(EEDRV_IdType id, EEDRV_addressType srcAddress, (EEDRV_addressPtrType)* destAddr, uint16 size)
{
    if(NULL_EEDRV_FNCTN_CFG_PTR != EEDRV_ConfigPtr)
    {
        while(index++ < size)
        {
            *destAddr++ = EEDRV_readByte(id, srcAddress++);
        }
    }
}

EEPROM_dataType EEDRV_readByte(EEDRV_IdType id, EEDRV_addressType address)
{
    if(NULL_EEDRV_FNCTN_CFG_PTR != EEDRV_ConfigPtr)
    {
        if(EEDRV_I2C == EEDRV_Config[id]->eeInterface)
        {
            EEDRV_ProcessI2CReadCmd(id, address);
        }
        else if(EEPROM_SPI == EEDRV_ConfigPtr[id]->eeInterface)
        {
            EEDRV_ProcessSpiReadCmd(id, address);
        }
    }
}
```
**Example – Module Interaction (code snippets)**

### Interaction with other modules - EEPROM Module

**EEPROM Driver – Write Cycle Status**

```c
EEDRV_StatusTypeDef EEDRV_isBusy(EEDRV_IdType id)
{
    EEDRV_status status = EEDRV_IDLE;

    if(NULL_EEDRV_FNCTN_CFG_PTR != EEDRV_ConfigPtr)
    {
        if(EEPROM_I2C == EEDRV_ConfigPtr[id]->eeInterface)
        {
            I2C_requestWriteCmd(EEDRV_DeviceAddress());
            I2C_Start(EEDRV_ConfigPtr[id]->commId, I2C_START);
            I2C_putchar( EEDRV_ConfigPtr[id]->commId, 
                        I2C_getAddress(EEDRV_DeviceAddress(), I2C_ADDRESS_7BIT) 
                    );

            if(I2C_ServeAsk(EEPROM_ConfigPtr[id]->commId))
            {
                EEDRV_status = EEDRV_BUSY;
            }
            I2C_Stop(EEPROM_ConfigPtr[id]->commId);
        }
        else if(EEPROM_SPI == EEDRV_ConfigPtr[id]->eeInterface)
        {
            if(EEDRV_readStatus(EEPROM_ConfigPtr[id]->commId))
            {
                EEDRV_status = EEDRV_BUSY;
            }
        }
    }

    return(status);
}
```
Example – Module interaction

Interaction with other modules - EEPROM Module

EEPROM Manager

© 2012. Vector CANtech, Inc.. All rights reserved. Any distribution or copying is subject to prior written approval by Vector.
Inter-module interaction - EEPROM Manager

EEPROM Manager - Executive

```c
void EEMNGR_Exec(void)
{
    ...  
    switch (eeMngrState)
    {
        ...  
        case EEMNGR_UPDATE:
        {
            if (EEMNGR_WRITE_DONE == EEDRV_writeByte(EEMNGR_GetDeviceId(EEMNGR_ramAddress), EEMNGR_EEaddress, *EEMNGR_ramAddress))
            {  
                eeMngrState = EEMNGR_VERIFY;
            }
            else
            {  
                EEMNGR_SetFault(EEMNGR_COMM_FAILURE);
                eeMngrState = EEMNGR_FAIL;
            }
        }
        break;
        case EEMNGR_VERIFY:
        {
            if(*EEMNGR_ramAddress == EEPROM_readByte(EEMNGR_GetDeviceId(EEMNGR_ramAddress), eeAddress))
            {  
                eeMngrState = EEMNGR_IDELE;
               ClrBitUS(EEMNGR_eeEnduranceTable[EEMNGR_updateByte], EEMNGR_updateBit);
            }
            else
            {  
                eeMngrState = EEMNGR_RETRY;
            }
        }
        break;
        ...  
    }
}
```
Module Example – External Software Driver

Sigma-Delta ADC TI ADS124x Family

Source: ADS1246, ADS1247, ADS1248 datasheet, August 2008, Revised October 2011
Module Example – External Software Driver

Sigma-Delta ADC TI ADS124x Family

Source: ADS1246, ADS1247, ADS1248 datasheet, August 2008, Revised October 2011
Module Example – External Software Driver

Sigma-Delta ADC TI ADS124x Family

Source: ADS1246, ADS1247, ADS1248 datasheet, August 2008, Revised October 2011
Module Example – External Software Driver

App_readExternalTemp(&App_Temperature)

HALSYS_readTemp(ptr)

HAL_readAds124x(ptr)

Ads124x_readOneShot(id)

SPI Driver

GPIO Driver

MCU LLD

SPI

GPIO

ADC_SPI_CLK

ADC_SPI_SIMO

ADC_SPI_SOMI

ADC_SPI_CS

ADC_Rdy

EXT INT

ADC_START

ADS1246
Module Example – External Software Driver

File Structure
(Simplified view)
Module Data Types

typedef struct Ads124x_hldCfg
{
    Ads124x_muxInputType   muxInput;
    Ads124x_pgaType        pga;
    Ads124x_samplingType   sampling;
    Ads124x_gpioType       startPin;
    Ads124x_gpioType       resetPin;

#if(ADS124X_CALIBRATION_ENABLE)
    Ads124x_calCoeffType   *ofcPtr;
    Ads124x_calCoeffType   *fscPtr;
#endif

#if(_ADS1247_ || _ADS1248_)
    Ads124x_gpioMaskType   gpioMask;
#endif
} Ads124x_hldCfgType;
Module Initialization

```c
const Ads124x_hldCfgType Ads124x_hldInit =
{
    ADS124X_SINGLE_INPUT,
    SYS0_PGA_8,
    SYS0_SPS_320,
    GPIO_NULL,
    GPIO_NULL,
#if(ADS124X_CALIBRATION_ENABLE)
    NULL,
    NULL,
#endif
#if(_ADS1247_ || _ADS1248_)
    ADS124x_NULL_GPIO_MASK
#endif
};
```
Module Initialization (cont.)

```c
void Ads124xInit( Ads124x_IdType id, const Ads124x_hldCfgType *cfgPtr)
{
    if(GPIO_NULL != resetPin)
    {
        Ads124x_ResetCycle(resetPin);
        /* Reset is needed prior to start conversion */
        Ads124x_Delay(ADS124X_RESET_DELAY_uS);
    }
    if(NULL != cfgPtr)
    {
        Ads124x_writeRegister(id, SYS0_ADDRESS, (Ads124x_regType)(cfgPtr->pga + cfgPtr->sampling));
        Ads124x_registersDump(id);
    }
#if(ADS124X_CALIBRATION_ENABLE)
    if(NULL != cfgPtr->ofcPtr)
    {
        Ads124x_Write_OFC_Coefficients(id, cfgPtr->ofcPtr);
    }
    if(NULL != cfgPtr->fscPtr)
    {
        Ads124x_updateFscCoeff(id, cfgPtr->fscPtr);
    }
    Ads124x_systemOffsetCal(id);
    Ads124x_systemGainCal(id);
    Ads124x_selfOffsetCal(id);
#endif
```
Module Initialization (cont.)

```c
#if(_ADS1247_ || _ADS1248_)
    if(ADS124x_NULL_GPIO MASK != cfgPtr->gpioMask)
        Ads124x_builtinGpioCfg(channel, (uint8)(cfgPtr->gpioMask & 0x0F))
    
#else
    if(GPIO_NULL != startPin)
        Ads124x_stopConversion(startPin);
#endif
```
Example – External Driver development example

Module Interfaces Examples

```c
void Ads124x_readOneShot(Ads124x_IdType id)
{
    /* Ignore dummy read data */
    (void)Spi_putCharacter(id, ADS124x_RDATA, SPI_CS_ACTIVE);

    /* Read Data */
    Ads124x_Data[2] = Spi_putCharacter(id, ADS124x_NOP, SPI_CS_ACTIVE);
    Ads124x_Data[1] = Spi_putCharacter(id, ADS124x_NOP, SPI_CS_ACTIVE);
    Ads124x_Data[0] = Spi_putCharacter(id, ADS124x_NOP, SPI_CS_DEACTIVATE);
}
```
Inter-Module Interaction

Application:

```c
void App_readExternalTemp(void)
{
    if(HALSys.TempSensorReadAvailable())
    {
        HALSys.readTemp(&App_Temperature);
    }
}
```

HAL System Interface

```c
#define HALSys_readTemp(ptr)\
    do{\
        HAL_readAds124x(ptr);\
        HAL_ClearTempSensorReadAvailable();\
    } while(0)
```
Conclusions

- The more complex requirements the more complex software implementation

- The evolvement of embedded software requires the application of software engineering concepts

- Reusable software components demands a higher amount of MCU resources (measures in memory size and execution cycles)

- A well defined software architecture allows the creation of truly reusable software components that can be effectively ported to different hardware architectures

- The effective implementation of software architecture requires a culture of discipline and commitment
Thank you for your attention.

For detailed information about Vector and our products please have a look at:

www.vector.com

Author:

Salvador Almanza, Special Projects(PES)
salvador.almanza@vector.com

Vector CANtech, Inc.