• What is S32DS?
• KEA Overview
• S32 Design Studio Overview
• Hardware Enablement – Frescale Freedom+KEA Board
• Getting Started with S32 Design Studio for ARM
• Arduino/FRDM Shields
• BLE Shield Example Project
• Project Customization
• Q&A
What is S32DS: Product Name Description

S32 = new unified automotive processors brand
e.g. Freescale S32K (Kinetis)

Existing Products
ARM®-based processors
MPC57xx, MPC56xx, MPC55xx MCUs
Power Architecture®-based processors
S12(X) MCUs
S12 MagniV mixed-signal MCUs
Image Cognition Processors
Kinetis auto MCUs
S08 MCUs
ARM Cortex®-based MCUs
MAC57Dxx MCUs
Others …

New Products

DS = Design Studio
for Automotive & Ultra-Reliable Industrial Processors
KEA Overview
Industrial Megatrends

Robust
- Proven Quality
- Ultra-Reliable
- Product Longevity
- High Temp Support
- Functional Safety (no SPF)

Secure Connectivity
- IoT / Big Data
- Sensor Fusion
- Detection
- Encryption
- Authentication
- Trust

Energy Efficient
- Balanced Performance/Power
- Higher Integration
- Space Constrained

Easy to Use
- Faster Market Launch
- Low Cost s/w
- Application RDs
- Open Source
- Trusted Partners
- Strong Support
Targeting Ultra-Reliable

**Best in class MCU quality, reliability and safety**
- 30 year history delivering industrial and automotive quality
- Zero defectivity = no field failures
- Advanced design for manufacturing (DFM) and test (DFT)

**Industry Benchmark in Longevity**
- 15 years minimum product longevity support
- 20 year service lifetime and 5 years shelf-life
- Demonstrated <1ppm defect quality

- **Application Target** = those which operate in harsh environments, >105C or have expensive repair costs
Ultra-Reliable Applications
URI Application Examples

**Industrial Appliances**
- Convection chamber
- Washer
- Refrigeration chamber
- Induction module
- HVAC

**Motor Control**
- E-Bikes
- AC motor
- DC / BLDC motor
- Sensorless motor
- HVAC
- Fluid pumps

**Building Automation**
- Circuit breakers
- Relays
- Safety alarms
- Fire alarms
- Safety lights
- Robot arm

**General Purpose**
- Power supply
- Power management
- HVAC
- Industrial HMI
- Lighting
- DC/DC...

**Industrial Metering**
- Pressure meters
- Oil pumps
- Coolant pumps
- Programmable logic controller (PLC)
Ultra-Reliable Applications – Use Case Examples

- **Coolant Pump** – S12ZVM to monitor pressure and temperature

- **Elevator** – MPC5744P as safety module, digital and analog inputs are evaluated and system put into safe state in case of error, Motor driven by separate MCU

- **Medical Pump** – MPC5645S as secondary MCU for HMI/ Safe Display, Asian language characters are stored in external QSPI Flash

- **3 Phase Solar Inverter** – MPC5675K dual core with External Memory Interface and Ethernet

- **Industrial Diesel control unit up to 20 cylinders** – MPC5777C with external Memory Interface; long TPU device legacy down to MPC565, ASIL-C

- **Elevator, SIL3 according to IEC61508** – MPC5744P plus Power SBC. Concept approved by TÜV Nord.

- **Medical Infusion Pump** – MPC5645S, SIL 2 plus medical safety requirements
Motor Control: Total Support

1. ARM (KEA, S32K)
   MagniV (S12ZVM)
   PA (MPC564xL, MPC5644P, …)

2. Development Kits

3. Simplified Functional Safety Support

4. Motor Control s/w Libraries

IEC 61508
ISO 26262
Ultra-Reliable Products
Ultra-Reliable ARM MCU Roadmap

### Graphics
- **SVF5/3R (Vybrid)**
- **MAC57D5xx (Halo)**
  - A5/M4/M0+, 2D-ACE GPU, 355pin

### High-Performance
- **KFA2M**
  - 120MHz M4, CAN-FD, 176pin
- **KFA1M**
  - 120MHz M4, CAN-FD, 144pin

### Mainstream
- **S32K144 (512 KB)**
  - 120MHz M4, CAN-FD, FlexIO, 64-100pin
- **S32K142 (256KB)**
  - 80MHz M4, CAN-FD, FlexIO, 64pin

### Entry
- **KEA128/64**
  - 48MHz M0+, LIN, CAN, 64-80pin
- **KEA32/16**
  - 40MHz M0+, LIN, 32-64pin
- **KEA8**
  - 40MHz M0+, LIN, 16-24pin

### Product Values
- Low power and performance
- Fast time to market
- ARM ecosystem
- Rich connectivity

---

*Last Updated May 15, 2018*
KEA Family

1st ARM based Ultra-Reliable MCU targeting connectivity and motor control applications in harsh environments

**Performance and Low-Power**
- Up to 48 MHz ARM M0+ core w/ up to 128kB Flash
- Single-cycle 32-bit x 32-bit multiplier
- <2µA in stop mode

**Reliability**
- AEC-Q100 Grade 1 Quality,
- Enhanced ESD/EMC performance up to 6KV
- Full compliance with IEC 61000-4
- Temperature support up to 125°C ambient

**Connectivity and Control**
- CAN, SCI, SPI, I2C, Analog comparators
- Multiple Timers with PWM functionality

**Enablement**
- Fast time-to-market with S32DS and FRDM+
- Low Level drivers, Reference designs, example code

**Applications**
- CAN and LIN nodes
- Peripheral Gateway
- Lighting Control
- BLDC and DC Motor Control
- Pump/Fan Controller
- Battery Management
- HVAC
S32K Family

Broadens ARM Ultra-Reliable MCU portfolio and offers scalability from 8k – 2MB

**Performance and Low-Power**
- Up to 120 MHz ARM M4 core w/ 256 – 2MB Flash
- DMA and single-cycle 32-bit x 32-bit multiplier
- <25uA sleep mode w/ RAM powered

**Reliability**
- AEC-Q100 Grade 1 Quality, up to 125C
- Enhanced ESD/EMC performance up to 6KV
- Full compliance with IEC 61000-4
- Supports ASIL-B safety and hardware security

**Connectivity and Control**
- Up to 5 x CAN (2xFD), FlexIO (4xSCI, 3xSPI, 2xI2C)
- Multiple Timers with PWM functionality

**Enablement**
- Full SDK driver suite for rapid development
- Reference designs, AUTOSAR MCAL, example code

**Applications**
- CAN and LIN nodes
- Industrial HVAC
- Lighting Control
- BLDC and PMSM Motor Control
- Pump/Fan Controller
Enablement for Faster Time-to-Market
New FRDM+ Platform

**Low-cost**
- FRDM+ kits start at only $29/ea

**Arduino UNO compatibility**
- Leverage global development support

**Ease of Use**
- Plug & Play using Arduino UNO shields
  - LCD, Motor Control, Bluetooth, Sensor, Communication…Wi-Fi
- New iQuick Start Guides (QSG)
- New iOne-sheets (samples/spec/tools)
- Peripheral drivers and code examples
- Example code embedded into MCU
- Demo example code using smartphone

**Shield Plans**
- 1 BLDC Shield → many FRDM+ MC kits
- Communication Shield → multi-CAN, SCI

**Feature Rich**
- I/O headers to support up to 176pins
- 12V supply, 5V and 3.3V operation
- OpenSDA USB (drag/drop capability)
- CAN/LIN SBC - Vreg, CAN/LIN PHY
- RGB LED and Potentiometer

**Availability**
- FRDM+KEA available in Aug’15
- Additional FRDM+ kits starting in mid Q3’15
S32 Design Studio

- Premium Compiler Support
- Basic GCC Compiler
- Base Eclipse Framework
- Industry Standard
- Basic GNU Debugger
- Tool Integration
- Software Integration
- Premium Debugger Support
Enablement Support

Customer Application Software

Professional Services for porting customer applications

Freescale Tools

- MCAT
  Motor Control Tool
- Motor Control Dev. Kit:
  Reference Software
- S32 Design Studio / CW
  Compiler & Debugger
- Math Motor Control Lib. (AMMCLIB)
- Partners (Cosmic, LB, GHS, iSys, PE, Keil, IAR)
- Software
- Hardware
  (FRDM+, EVB, Reference Designs)
- Software / Tools
- Hardware
- Middleware
- Virtual Modeling / Simulator (performance benchmarking)

- Low-level Drivers
- CAN/LIN/NVM Drivers
- Initialization
- Processor Expert
  Graphical Init Tool
- Pin Wizard

- FreeMASTER
  (GUI real-time monitor)
- AUTOSAR MCAL
- AUTOSAR OS
- Vector CAN Driver

Reduces Development Time
S32 Design Studio Overview
What is S32DS: Vision

Example code
Bootloaders
Init Tools
Low-level Drivers
Ref Design SW
Calibration/Tuning/Instrumentation
Motor Control Algorithms
Code generation tools
Comm stacks

One software and tools suite to save FSL sales and customers precious time AND expertise
What is S32DS?

**IS**
- Low-cost / No-cost Software Development Tool Strategy
- Low-Cost to develop and maintain
- No-Cost to our customers
- GCC Base Compiler Performance
- Intended to complement and enable Third-Party Partner Tools (Compilers & Debuggers)
- Support for Premium compilers addressing Automotive Safety Requirements (GHS, Cosmic,…)
- Integration of Freescale software in easy to use environment for application development (AMMCLIB, MCAT…)

**IS NOT**
- KDS Equivalent
- AUTOSAR software
- CodeWarrior Replacement
- Premium Tool Replacement (w/ Advanced Debugging, Best in class code performance, etc…)

---

#FTF2015
What is S32DS: Basic Tool Frame Work

- **GNU C/C++ Compiler**
  - GCC from **ARM Cortex-M** and **A cores**
  - GCC for **Power Architecture**
  - Standard Solid Compilers

- **Basic Eclipse Backplane**
  - C/C++ Development Tools
  - Integrated Editor with C/C++ tools
  - Managed Make Facility
  - Over 10,000 Eclipse Plug-in Available

- **GNU GDB Debugger**
  - Basic Debugger interface
to low cost JTAG debugger
  - (Pemicro, Segger, and OpenOCD)
What is S32DS: Premium Compiler/Debugger Support

**Third Party Premium Compiler Support**
ISO Certified compilers to support

Best in Class Compilers for Code Density and Code Performance

Examples: **GHS** and **IAR** both have ISO 26262 certifications and Certification Kits

**Third Party Premium Debugger Support**
When required for Trace of code execution and advanced debugging **Lauterbach** and **iSystems** are among Industry leaders.

These debuggers plug-in to the S32DS seamlessly integrated for use when the most difficult software problems need to be solved.
What is S32DS: Freescale Tools Integration and Enablement

Freescale Tools software enablement integrated for easy application inclusion

**Processor Expert** Initialization Configuration
Tool Configuration and Code Generation

**FreeMASTER** Signal Tuning and Data Monitoring

**Bootloader** Utility
- Flash Based Boot loader
- Desktop GUI and Command Line Tools

**Motor Control Application Tuning**

Embedded side software already integrated into a new project by S32DS project wizard

This integration ability allows users to easily leverage these Freescale tools
What is S32DS: Roll-out Strategy, Roadmap

- Establish base core tools chain **V1.0 Release**
  - Eclipse IDE with Editor
  - GCC Compiler
  - GNU Debugger
  - Basic Run Control P&E to hardware interface
  - New Project wizard for new NPI support
  - Initialization tool?

- **Quickly Add in Software Enablement and Third-Party Integration Pieces V1.+**
  - AMMCLib software libraries integrated in IDE
  - FreeMASTER embedded integration
  - Bootloader embedded integration
  - GHS/IAR Integration
  - Lauterbach/iSystems Integration
## What is S32DS: Timing & Schedules

### ARM

<table>
<thead>
<tr>
<th>Release</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha SP for SW team (KFA512 v1.0)</td>
<td></td>
</tr>
<tr>
<td>GCC 4.9 for M0+, M4</td>
<td></td>
</tr>
<tr>
<td>P&amp;E Debug</td>
<td></td>
</tr>
<tr>
<td>Service Pack Update (KFA512 Final SP)</td>
<td></td>
</tr>
<tr>
<td>Processor Expert (KEA/KFA)</td>
<td></td>
</tr>
<tr>
<td>KDS Import for KEA</td>
<td></td>
</tr>
<tr>
<td>AMMC Library</td>
<td></td>
</tr>
<tr>
<td>GCC 4.9</td>
<td></td>
</tr>
<tr>
<td>P&amp;E Debug</td>
<td></td>
</tr>
<tr>
<td>MCAT</td>
<td></td>
</tr>
<tr>
<td>GHS Plug-in</td>
<td></td>
</tr>
<tr>
<td>FreeMaster</td>
<td></td>
</tr>
<tr>
<td>iSystem Plug-in</td>
<td></td>
</tr>
<tr>
<td>Lauterback Plug-in</td>
<td></td>
</tr>
<tr>
<td>OpenOCD</td>
<td></td>
</tr>
<tr>
<td>Bootloader</td>
<td></td>
</tr>
<tr>
<td>Core Self Test</td>
<td></td>
</tr>
</tbody>
</table>

### Products

<table>
<thead>
<tr>
<th>S32DS for ARM</th>
<th>2015</th>
<th>2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEA128</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>KEA64</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>KEA32</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>S32K1x (KFA512) *Alpha SP</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>S32K1x (KFA256)</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

- **Major Release**
- **Developers Release**
- **Customer-driven Production Release**
- **Beta Release**
- **Maintenance Update Release**
What is S32DS: Current Release – v1.0

- Eclipse IDE – 4.4.2 (Luna)
- GNU Tools for ARM Embedded Processors 4.9 (Q4 2014)
- Project wizard supports derivatives:
  - KEA 128/64/32
  - KFA 512 (alpha SP)
- GNU Debugger
- Run Control P&E interface v11.1.1
- Open SDA version 1.14
Hardware Enablement: Freescale Freedom+KEA Board
Hardware Enablement – Freescale Freedom+ for Endless Possibilities

- Hardware compatibility with Arduino shield community modules
  - LCD, Motor Control, Wi-Fi®, Bluetooth®, Analog/Sensors
- element14 community support
- Interactive and simplified QSG (Quick Start Guide)
- Arduino Shields adapters examples:
Hardware Enablement – Freescale Freedom+ in Detail

- CAN Bus
- External Power Supply (5-12 V)
- System Basis Chip MCZ33903
- KEA SWD
- Kinetis KEA MCU
- RGB LED
- LIN Bus
- OpenSDA USB
- Reset Button
- OpenSDA MCU
- OpenSDA JTAG
- User Buttons
- Potentiometer
Getting Started with S32 Design Studio for ARM
Agenda

• Create New Project
• Debugger Configuration
• Basic Debug Session
• Startup Code
• GPIO
• Header Files
Create New Project
Create New Project: First Time – Select a Workspace

- Start program: Click on “S32 Design Studio for Power v1.0” icon
- First time only: Select workspace
  - Choose default (see below example) or specify new one
  - Suggestion: Uncheck the box “Use this as the default and do not ask again”
  - Click OK
Create New Project: Top Menu Selection

- File – New – S32DS Project
Create New Project: **S32DS Project**

- **Project Name:**
  - Example: FirstProject
- **Project Type:**
  - Select from inside executable or library folder
- **Next**
Create New Project: Target Processor

- KEA target: select micro
- Float ABI: default or none
- I/O Support options (for any console msgs)
  - UART
  - Debugger console
  - No I/O (None)
- Library Support options (ARM libraries)
  - newlib
  - newlib_nano (suggested starting point)
  - ewl
- Language
- Click “Next”
Create New Project: Target Debug

- Run Control Support:
  - PE Micro GDB server
  - Click “Finish”
Create New Project: Build Project

• Select project
• If code was altered, save before building
• Project – Build Project
• Console tab has message that build finished
Debug Configuration
Debug Configuration: Select Interface & Project

- Connect target to PC
- Select: Run – Debug Configurations
- Main tab displays selections.
  - Select target as below.
Debug Configuration: Verify Debugger Settings

- Select Debugger tab
- Verify Interface is correct or make proper selection
- Verify Port is found. If the target is disconnected from USB port, connect target and hit refresh
Debug Configuration: **First Breakpoint**

- Select Startup tab. Modify breakpoint if desired.
Debug Configuration: **Start debugger**

- Select Debug.
Debug Basics
Debug Basics: Starting the Debugger

• Debug configuration is only required once. Subsequent starting of debugger does not require those steps.
• Three options to start debugger:
  – **FIRST TIME**: If the “Debug Configuration” has not been closed, click on “Debug” button on bottom right
  
  – Select Run – Debug (or hit F11)

  – **Recommended Method**: Click on pull down arrow for bug icon and select …_debug.elf target
Debug Basics: Step, Run, Suspend, Resume

- Step Into (F5)
- Step Over (F6)
- Step Return (F7)
- Run
- Suspend
- Resume (F8)
Debug Basics: View & Alter Variables

- View variables in “Variables” tab.
- Click on a value to allow typing in a different value.
Debug Basics: View & Alter Registers

• View CPU registers in the “Registers” tab
• Click on a value to allow typing in a different value

• View peripheral registers in the EmbSys Registers tab
Debug Basics: View & Alter Memory

• Add Memory Monitor

• Select Base Address to Start at: 40000000

• View Memory
Debug Basics: Breakpoints

Add Breakpoint: Point and Click

- light blue dot represents debugger breakpoint
Debug Basics: Reset & Terminate Debug Session

• Reset program counter

• Terminate Ctl+F2()
LAB 1: Bare Metal Blinking LEDs
Simple Lab 1

• Setup a simple state machine
  – Use the RGB LED on the KEA FRDN+ board
  – Toggle between two states (RED and BLUE)
  – Set LED Color in each state to be different
  – Use a counter to determine when to transition between states
  – Create and update a state variable
Lab 1: RGB LED on KEA FRDM+ Board

- RGB_RED on port PTH0 of the MCU
- RGB_GREEN on port PTH1 of the MCU
- RGB_BLUE on port PTE7 of the MCU
Lab 1: State Machine Example

Container State

- state = 1
- counter = 0
- ports = output
- LED = GREEN

counter++

- LED = RED
  - State = 2
  - counter > MAXCNT/2

- LED = BLUE
  - State = 3
  - counter > MAXCNT
  - counter = 0

- counter > MAXCNT/2
Lab 1: Code the State Machine

• Use new project wizard to create a bare metal project

• “main” from new project wizard

• simple counter variable in an infinite loop
Lab 1: Code the State Machine

- Add macro to identify the ports to be used
- PTH0, PTH1, and PTE7

```c
/* Create port Macros */
#define PTE7 7    /* Blue LED port */
#define PTH0 24   /* Red LED port */
#define PTH1 25   /* Green LED port */
```

- Add macro for MXCNT to test counter variable against

```c
/*Create Max count MACRO 500,000 counts*/
#define MAXCNT 500000
```
Lab 1: Code the State Machine

- Initialize variables, set port direction, set “state” variable

Turn LED to GREEN

```c
int main(void)
{
    int counter = 0; /* counter is initialized to 0 */
    int state = 0; /* uninitialized state */

    /* Initialization State LED is GREEN */

    /* Initialize Port Direction */
    GPIOB_PDDR |= 1<<PTE7; /* Setup PTE7 as an output for Blue LED */
    GPIOB_PDDR |= 1<<PTH0; /* Setup PTH0 as an output for Red LED */
    GPIOB_PDDR |= 1<<PTH1; /* Setup PTH1 as an output for Green LED */

    /* Initialize LED to GREEN */
    GPIOB_PSOR |= 1<<PTE7; /* Set output port PTE7*/
    GPIOB_PSOR |= 1<<PTH0; /* Set output port PTH0*/
    GPIOB_PCOR |= 1<<PTH1; /* Clear output port PTH1*/
    state = 1; /* state set to initialized */
}```
Lab 1: Code the State Machine

- Add two states with entry condition code
- In each state set LED to proper color
- Set state variable to correct number

```c
for(;;)
{
    counter++;
    if(counter > MAXCNT) /* 50% of count LED is to be BLUE */
    {
        /* Set to BLUE State */
        state = 2;
        /* Set LED to BLUE */
        GPIOB_PCOR |= 1<<PTE7; /* Clear output port PTE7 */
        GPIOB_PSOR |= 1<<PTH0; /* Set output port PTH0 */
        GPIOB_PSOR |= 1<<PTH1; /* Set output port PTH1 */
        counter = 0;
    }
    else if (counter > MAXCNT/2) /* 50% of count LED is to be RED */
    {
        /* Set to RED State */
        state = 3;
        /* Set LED to RED */
        /* Set output port PTE7 */
        GPIOB_PSOR |= 1<<PTE7; /* Set output port PTE7 */
        GPIOB_PCOR |= 1<<PTH0; /* Clear output port PTH0 */
        GPIOB_PSOR |= 1<<PTH1; /* Set output port PTH1 */
    }
}
return 0;
```
Lab 1: Build, Debug, Run Code on target
Alternate Lab 1: Code with input & output

#include "derivative.h" /* include peripheral declarations SKEAZ128M4 */
#define PTH0 0 /* Port PTH0, 1 bit: output to LED_RED */
#define PTE4 4 /* Port PTE4, 5th bit: input from SW2*/

int main(void)
{
    int counter = 0;
    GPIOH_PDDR |= 1 << PTH0; /* Port Data Direction= output on port H0 */
    GPIOE_PIDR &= ~(1<<PTE4); /* Port Input Disable= disabled (input enabled) on E4 */
    for(;;) {
        counter++;
        if (((GPIOE_PDIR >> PTE4) & 1) { /* if Pad Data Input = 1 (BTN0 is pushed) */
            GPIOH_PSOR |= 1<<PTH0;
            /* Set Output (0) on port H0 */
        }
        else { /* if BTN0 was not pushed */
            GPIOH_PCOR |= 1<<PTH0;
            /* Clear Output (0) on port H0 */
        }
    }
    return 0;
}
Bluetooth Smart Shield Lab
Bluetooth Smart in Automotive

- Replace cables to further car weight reduction
- Tire pressure monitoring
- Remote car telematics
- Keyless entry
- Connection with Wearables - monitor a person's vital signs while driving
Bluetooth Smart Shield Lab: Embedded Side Components

Freescale Freedom+KEA (Freescale)

BLE Shield (RedBearLab.com)

S32DS for ARM (Freescale)
Control of Freescale Freedom+KEA Board Over Bluetooth (BLE)

- BLE shield by RedBearLab (Nordic Semiconductor nRF8001)  

- BLE Controller Smartphone app (Android / iOS App)
Lab 2: Bluetooth Smartphone Preparation
Lab 2: BLE Simple Chat – Check if Your Phone Supports BLE

• Step 1: Install BLE Controller into a mobile device
• Step 2: Run BLE controller and change mode to “Simple Chat”
• Step 3: Scan for BLE devices
• Step 4: Connect to an available device and send a message
• Step 5: Disconnect (no simultaneous connections possible)
Lab 2: Step 1 - Install BLE Android / iOS Application

- Search and Install and Run RedBearLab app – “BLE Controller”
- You should be asked to enable Bluetooth
Lab 2: Step 2, 3 – Connect to a BLE Device

- Select Simple Chat from the app setup.
- Return and scan for the list of available devices
- Press blue arrow to connect to one of the Boards detected
Lab 2: Step 4, 5 – Send a Message and Disconnect

- Invitation message sent by FRDM+KEA displayed
- Type a message to check board is working.
- Your message should display into a serial terminal and projected to audience
- Disconnect from the device to allow others to connect
Lab 3: Control Freescale Freedom+ Pins Over BLE
Lab 3: Control FRDM+ Pins Over BLE

- Step 1: Stack up Freescale Freedom+KEA, BLE shield, Prototyping Shield
- Step 2: Open S32DS and Import BLE project template
- Step 3: Change the shield name (up to 10 characters)
- Step 4: Build/Debug
- Step 5: Connect to your board using “BLE Controller” app
- Step 6: Control the pins by your mobile device
- Step 7: Prototyping Shield LEDs and Switch
Lab 3: Step 1 – Stack up the Board and Shields

- Freescale Freedom+KEA (Freescale)
- BLE Shield (RedBearLab.com)
- Prototyping Shield (3rd party)

Change SPI jumpers J11–J13 to side position
Jumper 9 - REQN [default]
Jumper 8 – RDYN [default]
Lab 3: Step 2 – Open S32DS and Project Import

1. Select a workspace
2. Import...
3. Existing Projects into Workspace
4. Copy projects into workspace

[Image of S32DS interface with import options and selected projects]

#FTF2015
Eclipse TIP - Code Completion feature

• CTRL + Space or defined time delay (0.5s) shows the selection dialog
Lab 3: Step 3 – Change the Shield Name

- Search `ble_set_name()` in `void setup()`
- Change name (up to 10 characters)
Eclipse TIP: Add a variable into Watch View

- Select symbol to be added into watch while debug session is active
- Right click and select “Add Watch Expression…”
Lab 3: Step 4 – Build/Debug
Lab 3: Step 5 - Connect to Your Board Using “BLE Controller” App
Lab 3: Step 6 - Control FRDM+Board over mobile phone App

- Pin 11: RED
- Pin 12: GREEN
- Pin 13: BLUE
- Pin 14: Button SW2
- Pin 15: Button SW3
- Pin 22: Potentiometer

RGB LED (Pins 11, 12, 13)
Button SW3 (Pin 15)
Button SW2 Potentiometer (Pin 22)
Lab3: Step 7 - Prototyping Shield LEDs and Switch

- Connect LED1 to Digital Pin 0
- Connect LED2 to Digital Pin 3 (PWM feature)
- Connect Switch S1 to Digital Pin 1
- Optional (connect a POT at breadboard to Analog input 0 (Pin 16 in App)
Lab 3: Step 7 - Connect LED1 to Digital Pin 0 (Output), LED2 to Digital Pin 3 (PWM feature), Switch S1 to Digital Pin 1 (Input)
Lab 4: FreeMASTER integration
Lab 4: Check that added Folder is not excluded from build

1. Right-click on the folder "freemaster_driver" in the project window.
2. Select "Resource Configurations" from the context menu.
3. Click on "Exclude from Build..."
4. Select "Debug" or "Release" in the Exclude from build window.
5. Click on the folder "freemaster_driver" in the project window.
Lab 4: Add FMSTR_Init() and FMSTR_Poll()

```c
#include <stdbool.h>
#include <boards.h>
#include "arduino.h"

#include "freemaster.h"  // FreeMASTER header file
#include "drv_uart.h"     /* UART driver */

#define PROTOCOL_MAJOR_VERSION 0 //
#define PROTOCOL_MINOR_VERSION 0 //
#define PROTOCOL_BUGFIX_VERSION 2 // bugfix

#define PIN_CAPABILITY_NONE 0x00
#define PIN_CAPABILITY_DIGITAL 0x01
#define PIN_CAPABILITY_ANALOG 0x02
#define PIN_CAPABILITY_PWM 0x04
#define PIN_CAPABILITY_SERVO 0x08
#define PIN_CAPABILITY_I2C 0x10

volatile uint32_t test;

msf_init(0);  // msf_hw drivers init.
arduino_init();  // Internal function which must be called by main before calling setup().
setup();  // Arduino setup()

Driver_UART2.Initialize(BD115200, null);
FMSTR_Init();  // Initialization of FreeMASTER driver

for (; ;)
{
    loop();  // Arduino loop()
    FMSTR_Poll();  // FreeMASTER Polling routine
}
return 0;
```

- Include header file “freemaster.h”
- Include header file “drv_uart.h” (initializeUART – non FMSTR function)
- Initialize UART2 driver
- Add FMSTR_Init() prior to main loop
- Add FMSTR_Poll() into main loop
Eclipse TIP: Spilt Editors vertically/horizontally
Lab 4: Check config file [freemaster_cfg.h]

```c
/* @file freemaster_cfg.h.example */

#ifndef __FREEMASTER_CFG_H
#define __FREEMASTER_CFG_H

#include "SKEAD2B04.h"

/**
 * Select interrupt or poll-driven serial communication
 */

#define PMSTR_LONG_INTR 0 /* complete message processing in interrupt */
#define PMSTR_SHORT_INTR 0 /* SCI FIFO-queuing done in interrupt */
#define PMSTR_POLL_DRIVEN 1 /* no interrupt needed, polling only */

/**
 * Select communication interface (SCI, CAN, USB CDC or Packet Driven BDM)
 */

#define PMSTR_SCI_BASE UART2_BASE_PTR /* UART2 base on KEA */
#define PMSTR_SCI_INTERRUPT UART2_IRQn /* UART2 interrupt vector on KEA */
#define PMSTR_CAN_BASE 0x40024001u /* MSCAN base on KEA */
#define PMSTR_CAN_INTERRUPT 30 /* MSCAN1 interrupt vector */
#define PMSTR_DISABLE 0 /* To disable all Freemaster functionalities */
#define PMSTR_USE_SCI 1 /* To select SCI communication interface */
#define PMSTR_USE_FLEXCAN 0 /* To select FlexCAN communication interface */
#define PMSTR_USE_USB_CDC 0 /* To select USB CDC communication interface */
#define PMSTR_USE_PDBDM 0 /* To select Packet Driven BDM communication interface (optional) */,

#define PMSTR_FLEXCAN_TXMB 0
#define PMSTR_FLEXCAN_RXMB 1

/**
 * Input/output communication buffer size
 */

#define PMSTR_COMM_BUFFER_SIZE 0 /* set to 0 for "automatic" */

/* clear Freemaster */
```

- **Polling mode**
- **UART2 Base address**
- **FRDM+KEA**
- **Use SCI/UART as comm. interface**
Eclipse TIP: Speed up build process – enable parallel build

1. BEFORE
   - CDT Build Console: Executing target #35 LAB4_BLE_Control_FreeMASTER_KEA64.siz
   - Build Finished: 16:19:33 Build Finished (took 26.180ms)

2. BEFORE
   - Build Behavior: Enable parallel build
     - Use optimal jobs (4)

3. AFTER
   - CDT Build Console: Executing target #35 LAB4_BLE_Control_FreeMASTER_KEA64.siz
   - Build Finished: 16:28:09 Build Finished (took 15.445ms)

4. AFTER
   - Build Behavior: Enable parallel build
     - Use parallel jobs: 4
     - OK
Lab 4: Build the project and run Debugger

1. Build ‘Debug’ for project ‘LAB4_BLE_Control_FreeMASTER_KEA64’
2. Click ‘Debug’ in the top menu bar
3. Select ‘LAB4_BLE_Control_FreeMASTER_Debug_KEA64’ from the list
4. Click ‘Debug’ button to run the project in Debugger
Lab 4: Launch PC side FreeMASTER

C:\Freescale\FreeMASTER 1.4\pcmaster.exe
Or
"C:\Program Files (x86)\Freescale\FreeMASTER 1.4\pcmaster.exe"
Lab 4: Setup communication port – Serial (COM)

- FreeMASTER will communicate via COM port
- If you FRDM+KEA board is plugged to USB – CDC serial port is available
- Setup Options\Comm Tab
Lab 4: Load Symbol File (ELF)

• Browse for the elf file generated by S32DS

Eclipse TIP: Use Alt + Shift + “W” on the file and select “System Explorer” in S32DS Project to get to the file on your Hard Drive
Lab 4: Adding Variables to FreeMASTER

- Add Device Name - global variable “device_name”

1. Click on the ‘Variables’ option.
2. Select the ‘Generate’ button.
3. Check the ‘device_name’ symbol.
4. Click on ‘Generate single variables.’
5. Click on ‘Close.’
Lab 4: Adding Variables to Variable Watch List

- Add Available variable “device_name” into Watched Variable list

1. Watch Properties...
2. Add ->
3. OK
Lab 4: Change/Adjust Variable properties

- Change displayed name to “BLE Shield Name”
- Change variable type to zero terminated string
Lab 4: Start Communication with FreeMASTER

- Application on the board must be running!
- Click on Start/Stop button to toggle the communication port on/off
- Communication details displayed at the bottom of “Variable Watch”
- Value column should show up-to-date content
Lab 4: FreeMASTER Summary

- Just basic variable monitoring demonstrated
- FreeMASTER offers much more
  - RealTime charts, Recorder, Visualization (HTML + JavaScript)
  - UART, BDM, CAN, LIN, USB
  - TSA enables variables to be defined without having/parsing .elf file
  - Free tool on a wide array of Freescale silicon
- Embedded Side integration with S32DS studio is coming soon
Bluetooth Smart Shield Lab: Software Layers

- **Application that calls Arduino API**
- **MSF lite s/w library** (www.codeproject.com)
- **Hardware (KEA)**
- **RBL Shield**
  - **SPI +handshake (ACI)**

**Nordic Semiconductor driver** (nRF8001)

**RedBearLab BLE library**

**Arduino API**

**MSF global functions**

**MSF drivers**

**MSF CMSIS – style drivers**

**CMSIS Core definitions**

- Arduino API
- MSF global functions
- MSF drivers
- MSF CMSIS – style drivers
- CMSIS Core definitions