1. Foundations

1.1. DeComSys NODE<RENESAS> Starterkit

As hardware target for the integration of TDL with the FlexRay protocol we used a FlexRay prototyping package by DeComSys called the NODE<RENESAS> Starterkit (current version: R2.0.2). It consists of 2 prototyping boards with a Renesas M32C/85 host CPU (24 Mhz, 2MB RAM, 2MB Flash) and a Bosch E-Ray FlexRay controller, a USB programming interface and a software package including the GNU C compiler, the AES operating system and drivers for the FlexRay controllers called COMMSTACK.

Refer to the getting started guide for using NODE<RENESAS> together with DESIGNER PRO.

1.1.1. Hardware

This section describes the hardware contents of the DeComSys NODE<RENESAS> Starterkit.

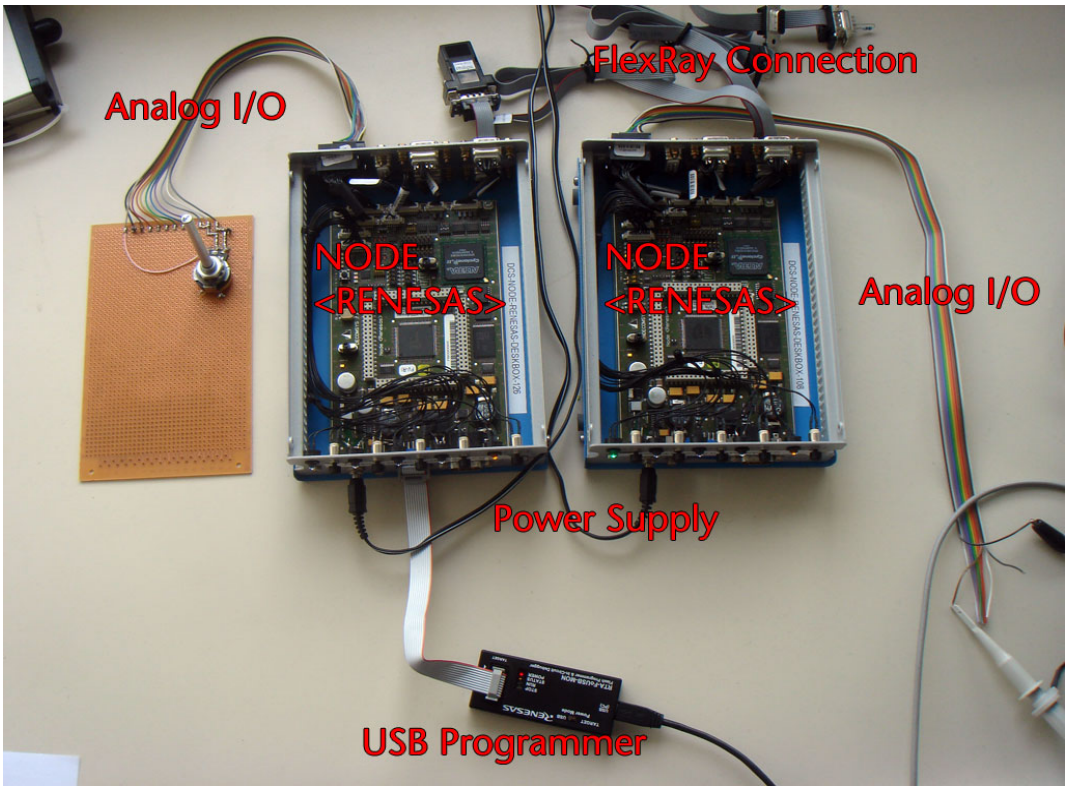

Figure 1 - Hardware overview

Figure 1 shows the typical setup of the prototyping hardware. Both nodes are connected to the power supply at the front of the node and to the FlexRay bus via a connector at the back. Analog I/O is connected by a special connector at the back named "AIO" for which the plug had to be built manually by using the pins

supplied with the starter kit. We use analog outputs to visualize the incrementer - decrementer example on the oscilloscope. The USB programmer needs to be connected to the blue socket at the front of the boards. The front panel of the board also contains digital I/O in form of 4 LEDs and 4 buttons. For detailed hardware information and pinouts of all connectors refer to.

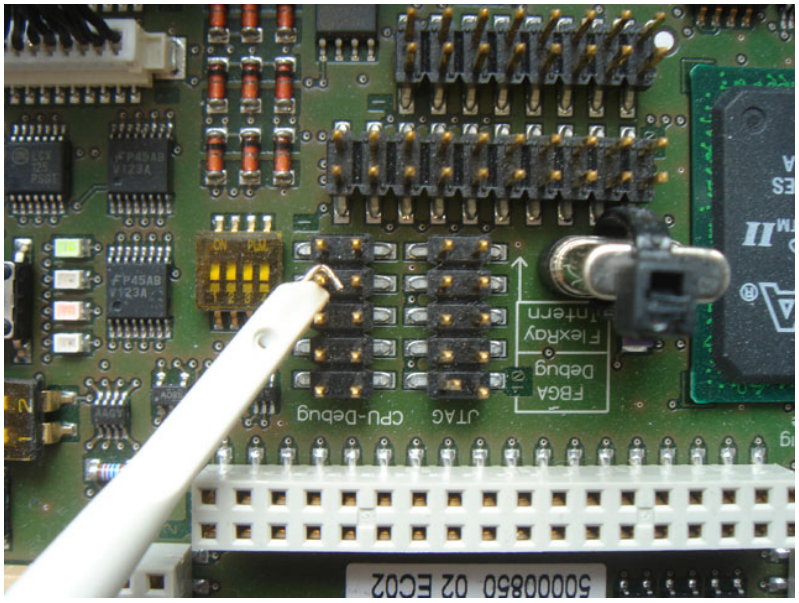

Figure 2 - Debug pin location

An undocumented but useful feature is the AES dispatcher debug pin that is high when a task is executed and low when the CPU is idle. It is located on the CPU-Debug connector at pin 3 as shown in Figure 2 where a probe is connected to the pin. It can be used to find out about the time instants at which tasks are activated and task execution times.

1.1.2. AES Operating System

AES is an abbreviation for Application Execution System and is an ANSI-C software library which enables executing periodic time driven application tasks. It is pre-runtime configurable and supports synchronization with the FlexRay communication system. The configuration describes the task execution by means of a dispatch table where an entry specifies the task and its invocation time with regard to the application period. Furthermore the synchronization behavior is described by additional configuration parameters.

Instead of the typical `main()` function which is normally the initial entry point in a C program, AES uses 3 hooks for initialization, idling and shutdown. In addition to that a dispatch table containing task functions and time instants is executed periodically.

The following example shows the complete configuration for AES which invokes the function `periodicTask()` every 5ms by executing it twice within an application period of 10ms. Note that all entities are mandatory for both single node and distributed systems.

```
static void periodicTask(void) {
  [...]
}
```

```
/* This task runs when there is no running time-triggered task. */
void skAES_ApplIdleTask (void) {
}

/* This function is called at system start up */
void skAES_ApplInitHook (void) {
}

/* This function is called when the system (skAES) shuts down. */
void skAES_ApplShutdownHook (skAES_ErrorType skAES_ErrNo) {
}

/* AES dispatch table */
const skAES_TaskDescriptionType skAES_TaskDescription[] = {
  /* Offset in us, Run only if synchronized with cluster, Task function */
  {0U, SK_AES_FALSE, periodicTask},
  {5000U, SK_AES_FALSE, periodicTask}
};

/* Number of tasks in the dispatch table */
const uint8 skAES_NumberOfTasks = sizeof(skAES_TaskDescription) /
sizeof(skAES_TaskDescription[0]);

/* Length of the application cycle */
const skAES_TimeType skAES_ApplCycleLenUs = 10000U;

/* FlexRay synchronization parameters */
const skAES_TimeType skAES_MaxDecreaseUs = 50U;
const skAES_TimeType skAES_MaxIncreaseUs = 50U;
const skAES_SyncModeType skAES_SyncMode = SK_AES_SYNCMODE_HARD;
```

The application cycle length skAES_ApplCycleLenUs is the length of the dispatcher round, i.e. the dispatch table is executed exactly once every application cycle. This means that a task in the dispatch table is at least executed every application cycle. The application cycle length can differ between nodes but always needs to be a 2^n multiple of the FlexRay cycle period.

For detailed information on the configuration constants and functions refer to the AES user manual.

1.1.3. Additional DeComSys Libraries

The COMMSTACK FlexRay controller driver provides frame based access to FlexRay controllers. It can only be used together with corresponding COMMSTACK configuration files which are generated by DESIGNER PRO.

The FlexRay Sync Handler (FrSh) manages COMMSTACK controller states.

The OS Synchronization Handler (OsSh) is used to synchronize the AES operating system to the time base of a running FlexRay cluster.

The Target Platform Infrastructure (skTPI) provides basic hardware access utilities (see 1.2 below).

1.1.4. Build Process

The DeComSys NODE<RENESAS> Starterkit includes a version of the free GNU C compiler, linker and make tool suitable for the Renesas M32C CPU. The build environment is described in.

Note that unfortunately in some rare cases GCC crashes during linking with an internal error. The reason is unclear and can normally be circumvented by e.g. reordering some lines of code which where changed recently. It also helps

sometimes to use the *NODE RENESAS Starterkit Shell* which can be launched from the Windows Start Menu, but this still did not solve all occurrences of the problem.

1.1.5. Download Tool

We used the KD3083 download and debug tool in version 3.30. It can be obtained from http://eu.renesas.com//fmwk.jsp?cnt=/upgrades_kd3083.jsp&fp=/support/downloads/download_results/C2002801-C2002900. Figure 3 shows its user interface.

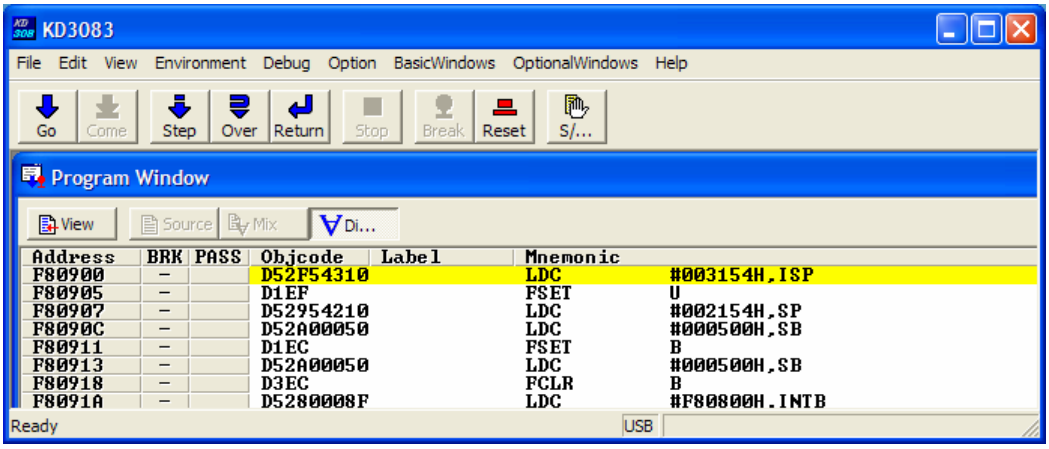

Figure 3 - KD3083 download application

The debug functionality does not work on source code level with the GCC compiler. DeComSys later supported a new version of the Renesas High-performance Embedded Workshop (HEW) to enable this, but we never used source code level debugging anyway.

Note that the KD3083 tool supports downloading of .elf files (via File > Download > Load Module...) and .mot files (via File > Download > Memory Image...). The DeComSys build tools produced both type of files and it is highly recommended to use the .mot files as the KD3083 tool sometimes crashes when .elf files are used!

The following settings need to be made upon tool start up:

- Init > MCU:
  - MCU: M30855.MCU
  - USB
- Init > Debugging Information:
  - Compiler: IAR EWM32C
  - Object Format: ELF/DWARF2.0
- Emem > Status:
  - Processor Mode: Memory Expansion Mode

1.2. NODE<RENESAS> I/O Access

This section lists some useful functions to drive input and output hardware. The complete API documentation of all functions provided by the NODE<RENESAS> StarterKit can be found in.

LEDs

Location: On the front panel

Function to drive the LEDs where PinNumber is SKTPI_LED_1..SKTPI_LED_4:

```
void skTPI_DebugPinOut(unsigned int PinNumber, boolean PinValue)
```

Buttons on the front panel

Location: On the front panel

This function returns the status of the buttons where PinNumber is SKTPI_BUTTON_1..SKTPI_BUTTON_4:

```
boolean skTPI_DebugPinIn(unsigned int PinNumber)
```

Analog I/O

Location: AIO connector on the back panel

The following functions are provided by the files AnalogIO.h/.c written by us using sample code from the DeComSys support team.

Initialization function for DA and AD converter:

```
void AnalogIO_init(void);
```

Function to get analog input values (8 bit). pin must be in range 0..7 corresponding to the CPU pins AN0..AN7:

```
uint8 AnalogIO_get(uint8 pin);
```

Function to output analog values (8 bit). pin must be in range 0..1 corresponding to the CPU pins DA0..DA1:

```
void AnalogIO_set(uint8 pin, uint8 value);
```

Serial I/O

Location: RS232 serial port on the front panel

Our file SerialIO.h/.c contains a simple function to write to the serial port in printf style:

```
void SerialIO_print (const char*, ...);
```