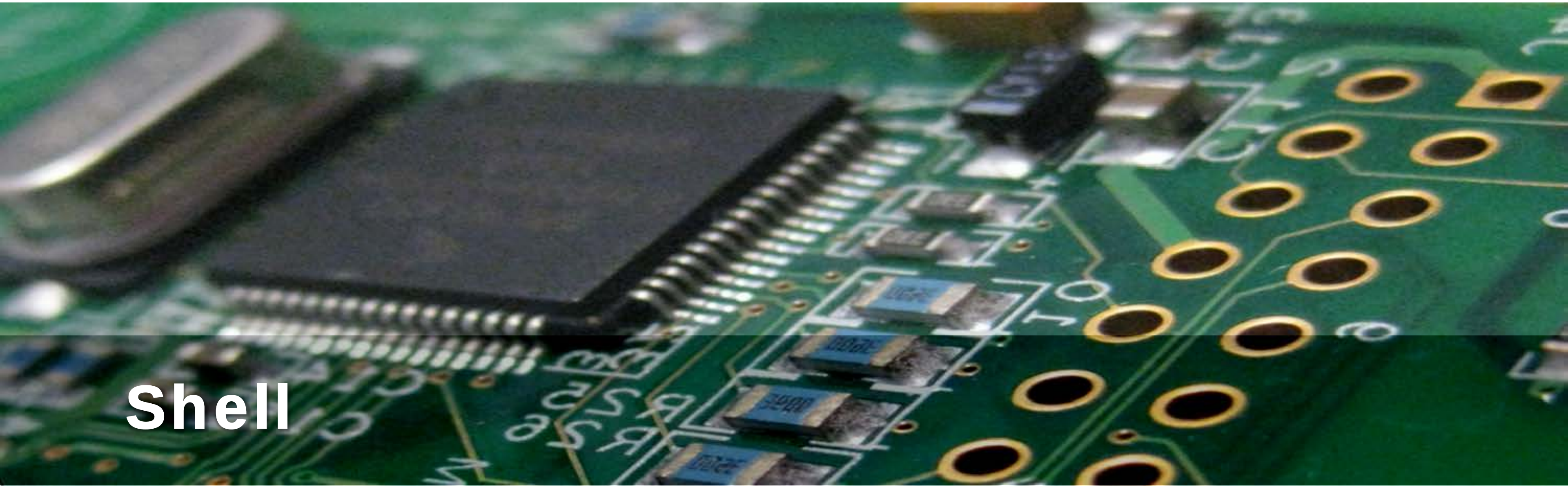
Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur



Shell

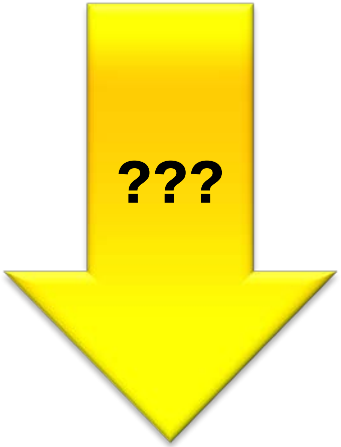
“We need an easy and universal way to configure and inspect our target, even if we are not debugging.”

Prof. Erich Styger
erich.styger@hslu.ch
+41 41 349 33 01

Learning Goals

- Problem: We need a simple user interface

- Goals
    - Shell (Console) Interface
    - UART
    - RTT
    - Create and use command handler

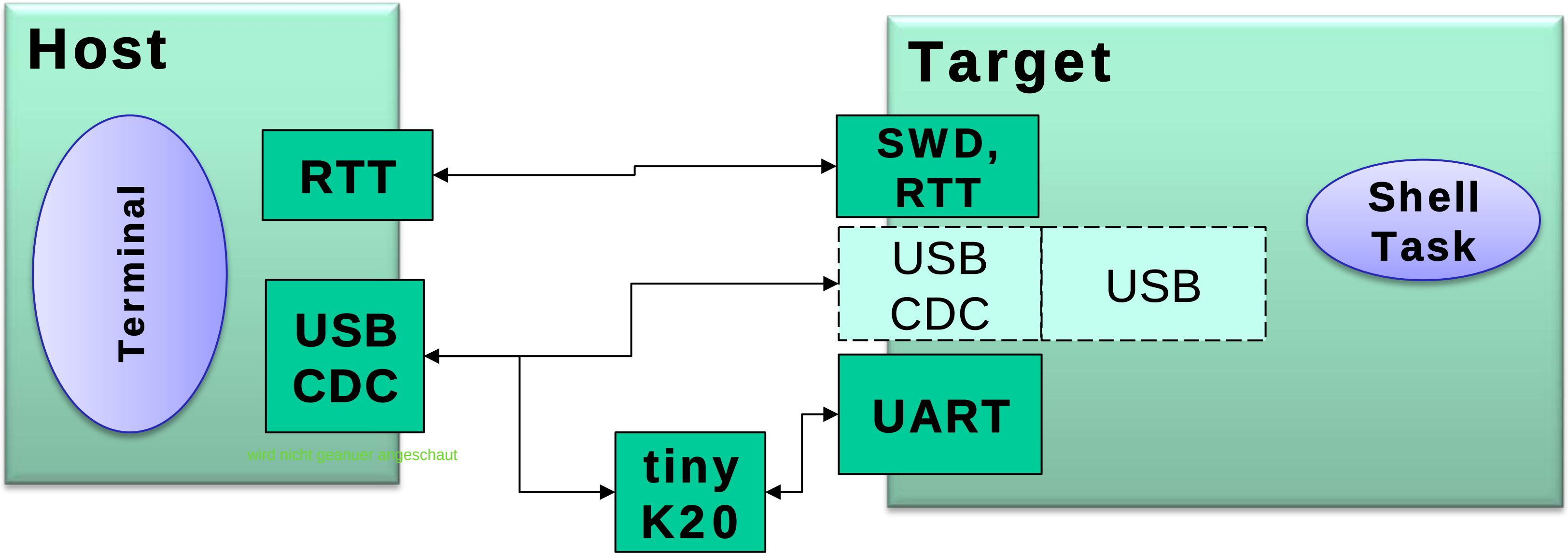


Hardware Connections

- Shell: Provides Command line Interface
- Robot V1: RTT, native USB CDC
- Robot V2: RTT, native USB CDC, UART over tinyK20
- Remote: RTT, native USB CDC, UART over tinyK20

- (later we will add a wireless channel)

RTT works both for the remote and the robot



wird nicht geanuer angeschaut

Shell Interface

```
/*!
 * \brief Sends a string to the shell/console stdout
 * \param msg Zero terminated string to write
 */
void SHELL_SendString(unsigned char *msg);

/*!
 * \brief Initializes the module and creates Shell task
 */
void SHELL_Init(void);

/*!
 * \brief Deinitializes the module.
 */
void SHELL_Deinit(void);
```

Shell Task

- ReadAndParseWithCommandTable()
  - Read in chars until '\n', appends to buffer reads from standard in
  - Uses StdIO as input/output channel
  - Uses table for parsers

add the string in the buffer, character by character

```
static void ShellTask(void *pvParameters) {
  static uint8_t localConsole_buf[48];
  CLS1_ConstStdIOTypePtr ioLocal = CLS1_GetStdio();

  localConsole_buf[0] = '\0';
  CLS1_ParseWithCommandTable(CLS1_CMD_HELP,
      ioLocal, CmdParserTable);
  for(;;) {
    CLS1_ReadAndParseWithCommandTable(localConsole_buf,
        sizeof(localConsole_buf), ioLocal, CmdParserTable);
    vTaskDelay(50/portTICK_RATE_MS);
  } /* for */
}
```

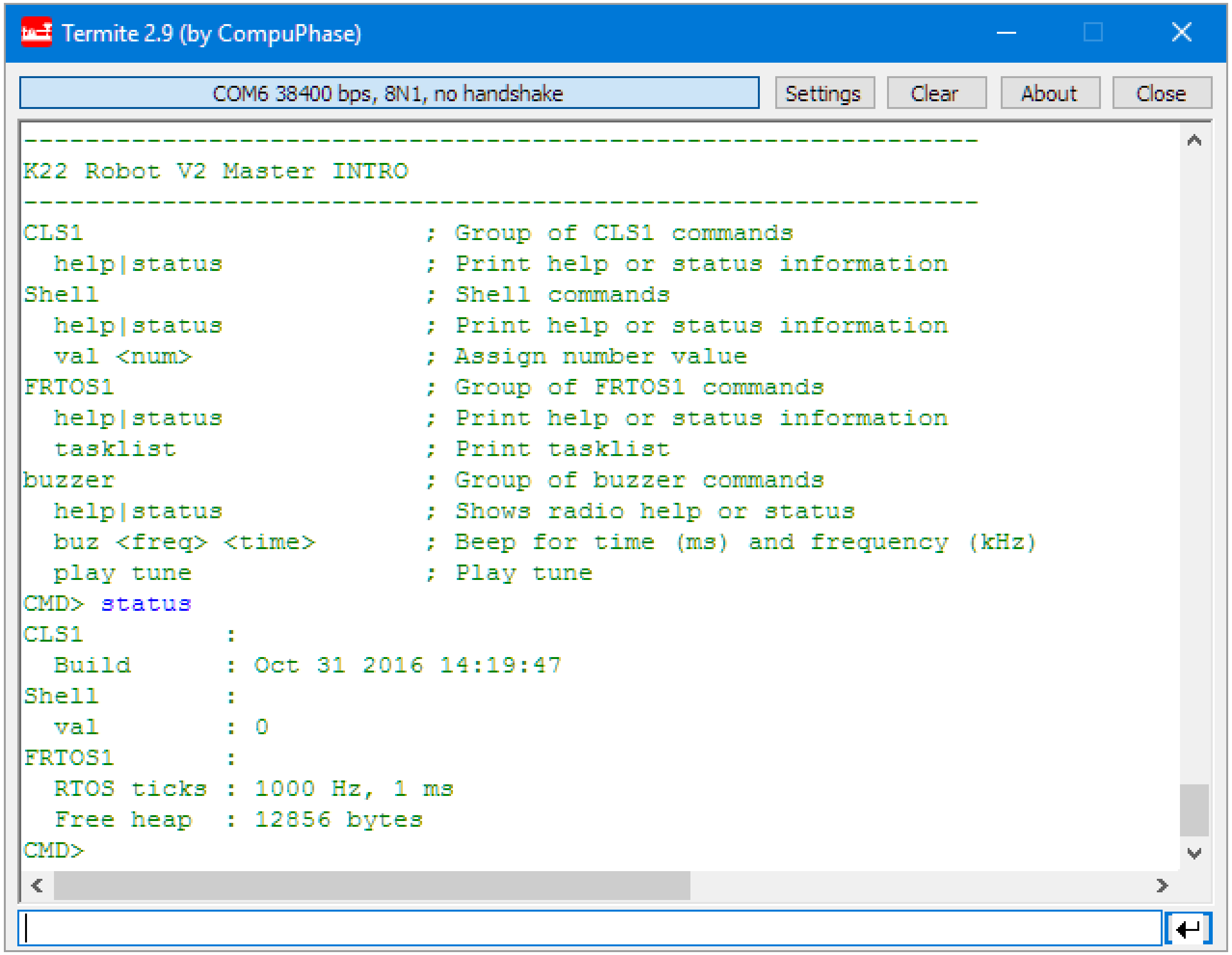
table explained later #11

every 50ms reads in the character

Example Console Session

default comments

- ‘help’ and ‘status’ default commands
- Shell status
- Shell help
- Shell val 5

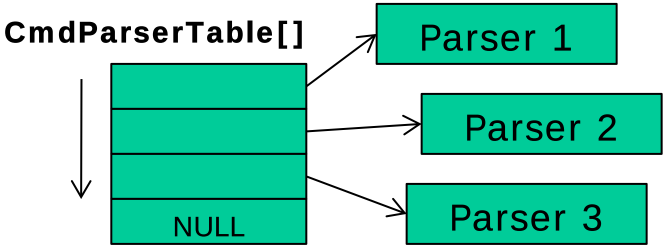

Termite 2.9 (by CompuPhase)
COM6 38400 bps, 8N1, no handshake
Settings
Clear
About
Close

```
--------------------------------------------------------------
K22 Robot V2 Master INTRO
--------------------------------------------------------------
CLS1                       ; Group of CLS1 commands
  help|status              ; Print help or status information
Shell                      ; Shell commands
  help|status              ; Print help or status information
  val <num>                ; Assign number value
FRTOS1                     ; Group of FRTOS1 commands
  help|status              ; Print help or status information
  tasklist                 ; Print tasklist
buzzer                     ; Group of buzzer commands
  help|status              ; Shows radio help or status
  buz <freq> <time>        ; Beep for time (ms) and frequency (kHz)
  play tune                ; Play tune
CMD> status
CLS1            :
  Build         : Oct 31 2016 14:19:47
Shell           :
  val           : 0
FRTOS1          :
  RTOS ticks    : 1000 Hz, 1 ms
  Free heap     : 12856 bytes
CMD>
```

Shell Command Parser Table

- Command Line Parser Table
  - **List / <u>Table</u>** of **Function Pointers** structure (list or table) -> we use the tabel
  - **<u>Static</u>** or **Dynamic** table an constant array in a dynamic table
  - **Size** argument or **<u>Sentinel</u>** (at the end of the table a special entry)
- Passed to shell parsing routine

```
static const CLS1_ParseCommandCallback CmdParserTable[] =
{
  CLS1_ParseCommand,
  . . .
  NULL /* Sentinel */
};
```



```
(void)CLS1_ParseWithCommandTable("help", CLS1_GetStdio(), CmdParserTable);
```

you can have multiple table if you want...

Example Command Parser

(implementation)

```
static uint8_t SHELL_ParseCommand(const unsigned char *cmd, bool *handled,
const CLS1_StdIOType *io) {
  uint32_t val;
  const unsigned char *p;

  if (UTIL1_strcmp((char*)cmd, CLS1_CMD_HELP)==0 || UTIL1_strcmp((char*)cmd,
           "Shell help")==0)
  {
    *handled = TRUE;
    return SHELL_PrintHelp(io);
  } else if (UTIL1_strcmp((char*)cmd, CLS1_CMD_STATUS)==0
            || UTIL1_strcmp((char*)cmd, "Shell status")==0) {
    *handled = TRUE;
    return SHELL_PrintStatus(io);
  } else if (UTIL1_strncmp(cmd, "Shell val ", sizeof("Shell val ")-1)==0) {
    p = cmd+sizeof("Shell val ")-1;
    if (UTIL1_xatoi(&p, &val)==ERR_OK) {
      SHELL_val = val;
      *handled = TRUE;
    }
  }
  return ERR_OK;
}
```

if you handled the comment or not

wrapper to the normal string compare

string compare byte by byte

extendend ascii to int

sizeof("abc"); -> 4, because abc0
strlen("abc"); -> 3

Help & Status

```
static uint8_t SHELL_PrintHelp(const CLS1_StdIOType *io) {
  CLS1_SendHelpStr("Shell", "Shell commands\r\n", io->stdOut);
  CLS1_SendHelpStr("  help|status", "Print help or status
                  information\r\n", io->stdOut);
  CLS1_SendHelpStr("  val <num>", "Assign number value\r\n",
                  io->stdOut);
  return ERR_OK;
}

static uint8_t SHELL_PrintStatus(const CLS1_StdIOType *io) {
  uint8_t buf[16];

  CLS1_SendStatusStr("Shell", "\r\n", io->stdOut);
  UTIL1_Num32sToStr(buf, sizeof(buf), SHELL_val);
  UTIL1_strcat(buf, sizeof(buf), "\r\n");
  CLS1_SendStatusStr("  val", buf, io->stdOut);
  return ERR_OK;
}
```

strncmp(), sizeof() & xatoi() Example

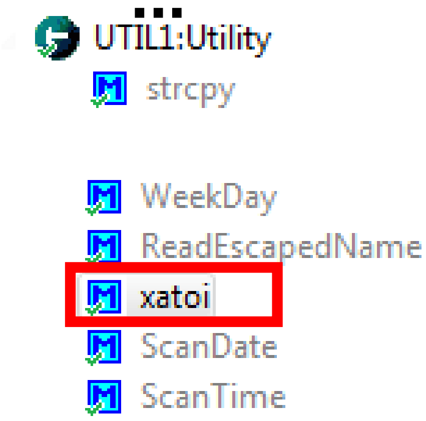
```
const unsigned char *p;
uint32_t val;
. . .
} else if (UTIL1_strncmp((char*)cmd,
 (char*)"val ", sizeof("val ")-1)==0)
{
  p = cmd+sizeof("val ")-1;
  if (   UTIL1_xatoi(&p, &val)==ERR_OK
      && val >=-100 && val<=100)
  {
   . . .
```

compares a number of characters

address

Parsing Numbers

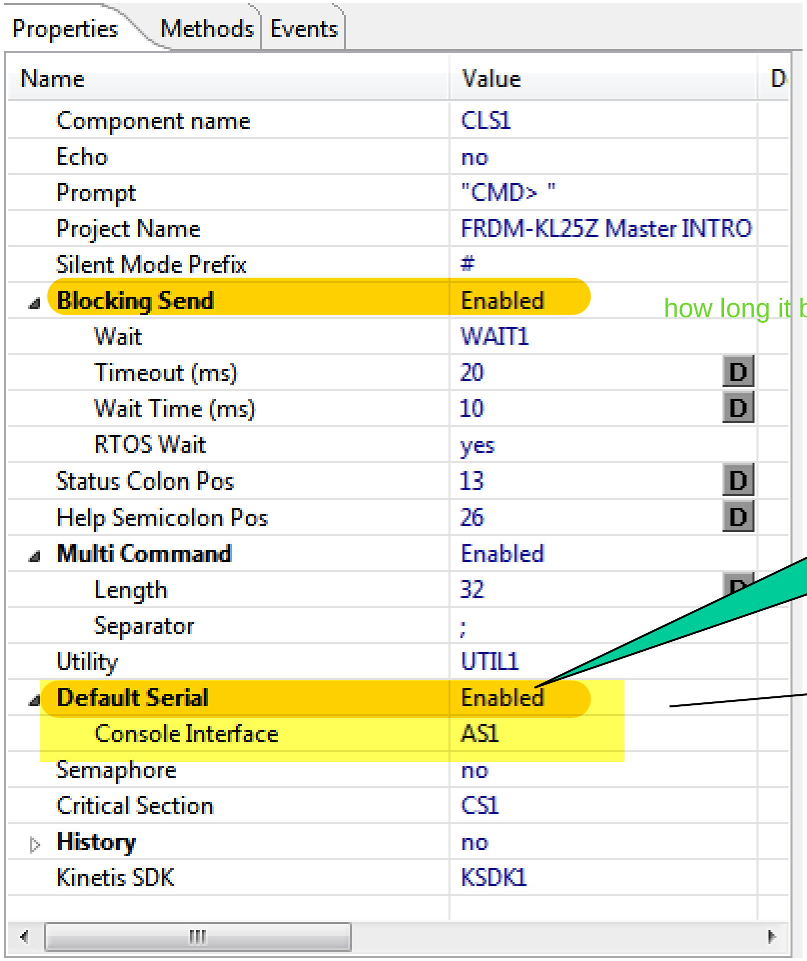
`val <number>`



```
uint8_t UTIL1_xatoi(const char **str, long *res)
{                       octal
/* 123 -5   0x3ff 0b1111 0377  w "
      ^                           1st call returns 123 and next ptr
         ^                        2nd call returns -5 and next ptr
                 ^                3rd call returns 1023 and next ptr
                        ^         4th call returns 15 and next ptr
                             ^    5th call returns 255 and next ptr
                                ^ 6th call fails and returns ERR_FAILED
*/
```

Shell and Default Communication Channel

- Shell usually has ‘default’ channel (stdio)



Name	Value	D
Component name	CLS1	
Echo	no	
Prompt	"CMD> "	
Project Name	FRDM-KL25Z Master INTRO	
Silent Mode Prefix	#	
Blocking Send	Enabled	
Wait	WAIT1	
Timeout (ms)	20	D
Wait Time (ms)	10	D
RTOS Wait	yes	
Status Colon Pos	13	D
Help Semicolon Pos	26	D
Multi Command	Enabled	
Length	32	D
Separator	;	
Utility	UTIL1	
Default Serial	Enabled	
Console Interface	AS1	
Semaphore	no	
Critical Section	CS1	
History	no	
Kinetis SDK	KSDK1	

how long it blocks

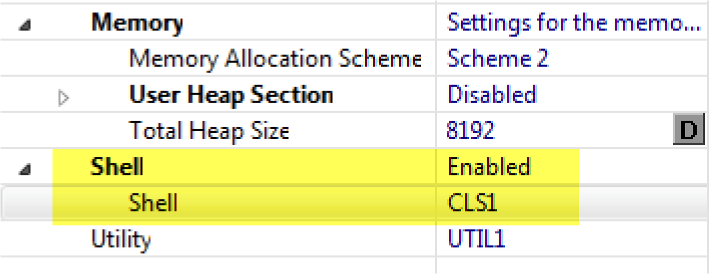
INTRO: Disabled, as we are going to use multiple communication channels

usefull if you use only one communication channel (RTT, UART,..)

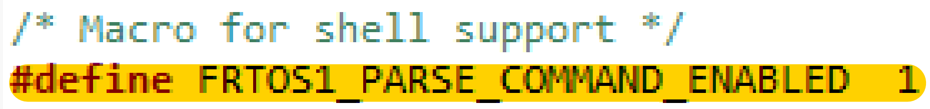
```
#define CLS1_DEFAULT_SERIAL  1
```

Components with Shell Support

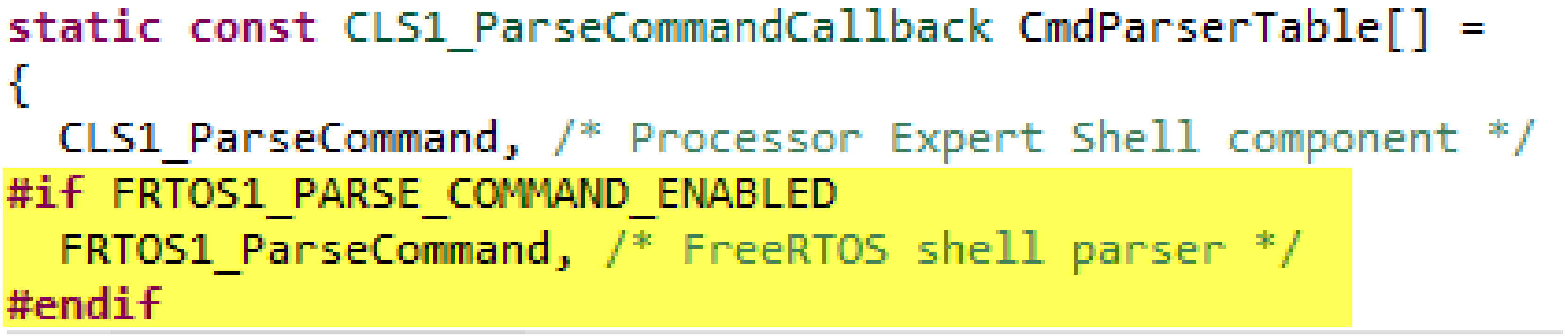
- Many Components have a Shell Parser implemented, e.g. FreeRTOS
- The setting configures a Macro which can be used in the parser table



◢ Memory	Settings for the memo...
Memory Allocation Scheme	Scheme 2
▷ User Heap Section	Disabled
Total Heap Size	8192 D
◢ Shell	Enabled
Shell	CLS1
Utility	UTIL1



```
/* Macro for shell support */
#define FRTOS1_PARSE_COMMAND_ENABLED  1
```



```
static const CLS1_ParseCommandCallback CmdParserTable[] =
{
  CLS1_ParseCommand, /* Processor Expert Shell component */
#if FRTOS1_PARSE_COMMAND_ENABLED
  FRTOS1_ParseCommand, /* FreeRTOS shell parser */
#endif
```

Custom Standard I/O

- Chaining/Redirecting Standard I/O
- Create your own handler struct
  - In: Reading Character
  - Out/Error: Sending Character
  - Pressed: if input is available
- Needs a buffer to read character stream (until ‘\r’,’\n’)
  - needs to initialized the first time with ‘\0’

```
 static CLS1_ConstStdIOType UART_stdio = {
    .stdIn = UART_ReceiveChar,
    .stdOut = UART_SendChar,
    .stdErr = UART_SendChar,
    .keyPressed = UART_KeyPressed,
  };

static uint8_t UART_DefaultShellBuffer[CLS1_DEFAULT_SHELL_BUFFER_SIZE];
```

UART I / O Handler

just for UART

```c
#include "AS1.h"

 static bool UART_KeyPressed(void) {
   return AS1_GetCharsInRxBuf()!=0;
 }

 static void UART_SendChar(uint8_t ch) {
   CLS1_SendCharFct(ch, AS1_SendChar);
 }

 static void UART_ReceiveChar(uint8_t *p) {
   if (AS1_RecvChar(p)!=ERR_OK) {
     *p = '\0';
   }
 }
```

true if something in the buffer

function to write something in the uart channel

AS1_SendChar(ch) -> non blocking

CLS1_SendCharFct(ch, AS1_SencChar) -> blocking with timeout

Shell I/O Handler Implementation

```
static void SHELL_SendChar(uint8_t ch) {
#if SHELL_CONFIG_HAS_SHELL_RTT
  RTT1_SendChar(ch);
#endif
#if SHELL_CONFIG_HAS_SHELL_UART
  UART_SendChar(ch);
#endif
}
```

```
static bool SHELL_KeyPressed(void) {
#if SHELL_CONFIG_HAS_SHELL_RTT
  if (RTT1_stdio.keyPressed()) {
    return TRUE;
  }
#endif
#if SHELL_CONFIG_HAS_SHELL_UART
  if (UART_stdio.keyPressed()) {
    return TRUE;
  }
#endif
  return FALSE;
}
```

```
static void SHELL_ReadChar(uint8_t *p) {
  *p = '\0'; /* default, nothing available
*/
#if SHELL_CONFIG_HAS_SHELL_RTT
  if (RTT1_stdio.keyPressed()) {
    RTT1_stdio.stdIn(p);
    return;
  }
#endif
#if SHELL_CONFIG_HAS_SHELL_UART
  if (UART_stdio.keyPressed()) {
    UART_stdio.stdIn(p);
    return;
  }
#endif
}
```

why do need to check if keyPressed?
-> wird abgefragt ob etwas im Input-Buffer ist (by R.K)

there is no serializing in RTT (help -> hsaelp)

```
CLS1_ConstStdIOType SHELL_stdio =
{
  (CLS1_StdIO_In_FctType)SHELL_ReadChar, /* stdin */
  (CLS1_StdIO_OutErr_FctType)SHELL_SendChar, /* stdout */
  (CLS1_StdIO_OutErr_FctType)SHELL_SendChar, /* stderr */
  SHELL_KeyPressed /* if input is not empty */
};
```

Shell I / O Descriptor Array

```
typedef struct {
  CLS1_ConstStdIOType *stdio;
  unsigned char *buf;
  size_t bufSize;
} SHELL_IODesc;

static const SHELL_IODesc ios[] =
{
    {&SHELL_stdio, SHELL_DefaultShellBuffer, sizeof(SHELL_DefaultShellBuffer)},
#if SHELL_CONFIG_HAS_SHELL_RTT
    {&RTT1_stdio, RTT1_DefaultShellBuffer, sizeof(RTT1_DefaultShellBuffer)},
#endif
    /*! \todo Extend as needed */
};
```

size of the whole thing (struct) divide sizeof "one line"

Shell Task

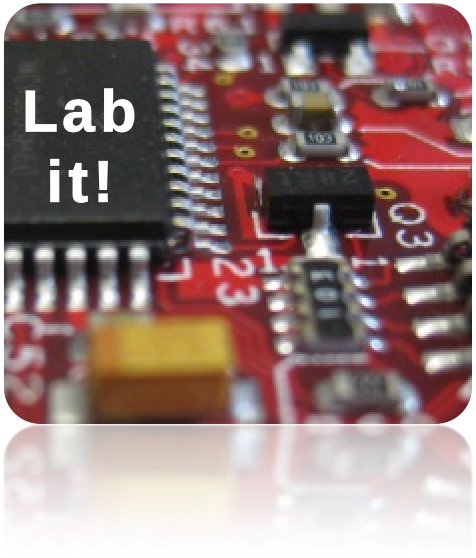
```c
static void ShellTask(void *pvParameters) {
  int i;

  /* initialize buffers */
  for(i=0;i<sizeof(ios)/sizeof(ios[0]);i++) {
    ios[i].buf[0] = '\0'; initialize the buffer
  }
                                    default start "help"
  CLS1_ParseWithCommandTable(CLS1_CMD_HELP, ios[0].stdio, CmdParserTable);
  for(;;) {
    /* process all I/Os */
    for(i=0;i<sizeof(ios)/sizeof(ios[0]);i++) {
      CLS1_ReadAndParseWithCommandTable(ios[i].buf, ios[i].bufSize,
          ios[i].stdio, CmdParserTable);
    }
    vTaskDelay(pdMS_TO_TICKS(10));
  } /* for */
}
```

Lab Task: Shell

- Integrate
  - Shell.c
  - Shell.h

- Communication Channels
  - Robo: Bluetooth
  - Both: Segger RTT

- Use shell for your own modules (status/help/commands)



Summary

- *Problem: We need a simple user interface*

- UART, RTT, (later: nRF24L01+ 2.4 GHz)
- Settings
  - Driver structure

- Shell
  - Configuration
  - Status / Help
  - Command Line Interface