RPP Test Software

Final report

Bc. Michal Horn 2012

1 Input and output interface

Communication between user and device is done through serial line. Software on the device expects commands in text form followed by eventual parameters (see 4.2), which are separated by spaces. An output is sent on serial line in text form as well. The output structure is not yet defined in actual implementation state.

1.1 Serial line configuration on control computer

To get proper data transfer functionality, it is necessary to set correct parameters for serial port on a computer that is used for sending commands and receiving outputs from the device. Right parameters are:

- Baudrate: 9600 Hz
- Parity: none
- Bits: 8
- Stopbits: 1
- Flow control: none

1.2 Command processing on the device

The device expects commands on the SCI port and uses a cmdProc[1] library for its processing. The cmdProc library processes command with its parameters and calls an appropriate function according to the commands definition (see 4.1).

1.2.1 SCI configuration

The SCI port on the device (do not mistake with SCI/LIN or SCI2) have to be activated. The proper data format have to be configured correctly (see 1.1) and TX INT and RX INT interruptions are supposed to be enabled.

1.2.2 Implementation receiving data from SCI

A FreeRTOS task is created after device reset. This task is reading commands in an infinite loop. The cmdProc reads characters from an input buffer until it gets the end of line character. Characters are inserted to the input buffer in a RX interrupt service routine (ISR). The input (and output as well) buffer is implemented as FreeRTOS queue data type, which is internally implemented as ring buffer.

When the cmdProc tries to read from empty buffer, it blocks itself until new character comes into the buffer.

When a situation occurs, that ISR tries to store a character into full buffer – that means that characters are processed slower than they are received from the SCI – this character is discarded and an error counter is incremented.

Once the cmdProc reads the end of line character, it processes the string, analyzes operators and in case of equality with some of the commands in the list, an appropriate function is called.

After the return from the function this cycle continues.

The most important in this process is, that all original interrupt service routines in sci.c file had to

1 Available from repository: git://ulan.git.sourceforge.net/gitroot/ulan/ulan-top

be reimplemented, because the original code from the Texas Instruments came with race conditions.

1.2.3 Implementation of sending data on SCI

When user needs send some information from the device, the easiest way is to call function print, which expects just a string as its parameter.

long print(const uint8_t* string);	
string	String to be sent on SCI
Return value	pdPASS, errBUFFER_NOT_INITIALIZED, errQUEUE_FULL

Print function prototype

The print function gives a right for writing just to one task at time. This is achieved by using mutex for the buffer. Other tasks are blocked until the first one finishes its printing.

After the print function obtains the right for writing, it tests an output buffers flag BUF_TRANSFER_IN_PROGRESS at first. This flag specifies, whether some SCI transfer is running or not. If no transfer is running, the function sets the flag and initiates new SCI transfer by invoking TX interruption. Than the ISR is sending characters from the output buffer one by one to the SCI while the print function is storing new characters designated to be sent to the output buffer.

In the case of slower sending to SCI than storing characters to the buffer, the print task is blocked until buffer is free again.

When the last character was sent from the buffer, the ISR clears the BUF_TRANSFER_IN_PROGRESS flag, which causes termination of the transfer until flag is set again.

2 Implemented peripherals

Peripheral	**Status**
ADC1	Working on HDK
CAN	Working on HDK
DIN 8 – 15	Not yet tested
HOUT	Not yet tested
MOUT	Not yet tested
SCI	Working, is used for sending and receiving commands
SDRAM	Particularly working, probably needs correct SDRAM configuration
VBAT Enable	Working, but for some reason two signals of four saturate only at 1.6V on prototype board

3 Implementation details

Currently there is implemented support for such peripheral devices, which are accessible directly from CPU (without an intermediate IC driven by SPI). Support for external SDRAM memory testing is implemented in addition.

3.1 ADC1

Analog digital converter connected to port ADC1 has 12 channels. Each channel measures voltage difference on High and Low pins. Each channel is connected directly to the CPU ADC pin.

3.1.1 HALCoGen settings

- Enable driver ADC1
- In ADC1 Group1 tab
  - Tick Enable channel ID conversion result
  - Set data resolution 12 bit
  - Enable Software triggering
  - Enable channels 0 – 11
- Enable VIM channel 15: ADC1 Group1

3.1.2 Software initialization

ADC is initialized by calling adcInit(). For proper function the IRQ have to be enabled by using a macro _enable_IRQ().

Before conversion starts, it is necessary to enable notification by adcEnableNotification() function for ADC1 and GROUP1. After the conversion finishes, it is good to disable those notification.

3.1.3 Measurement

Measurement is started by calling adcStartConversion() function. The calling task blocks itself on a semaphore and waits until conversion finishes. The end of the conversion invokes an interrupt request, which ISR reads converted data and unblocks the task again so it can subsequently process the data.

3.2 CAN

The device supports two CAN buses described as CAN1 and CAN2.

3.2.1 HALCoGen settings

- Enable drivers CA1 and CAN2
- Enable VIM channel 16: CAN1 HIGH
- Enable VIM channel 35: CAN2 HIGH

Loopback test is assigned to message 1.

- In setting CAN1 tab CANMsg 1-8

    - Activate Message1, direction TX, enable interrupts.
- In setting CAN2 tab CANMsg 1-8
    - Activate Message1, direction RX, enable interrupts.

3.2.2 Software initialization

CAN bus is initialized by calling canInit() function. For proper function, the IRQ have to be enabled by using a macro _enable_IRQ().

Before transmition starts, it is good to enable error notification by calling canEnableErrorNotification(). It should be disabled again at the end by canDisableErrorNotification().

3.2.3 Data transfer

Messages can be sent using canTransmit() function. Task than blocks itself on a semaphore and waits until the whole transaction finishes. The end of the transaction invokes interrupt request, which ISR unblocks the task

Received data can be read by canGetData() function.

3.3 DIN

DIN is a 16 channel digital input. Channels 0 – 7 are accessible through subsidiary integrated circuit MC33972 driven by SPI, channels 8 – 15 are, beside SPI, accessible through GPIO as well. This second group of channels have implemented data manipulation support.

3.3.1 HALCoGen settings

- Enable driver GIO
- In settings GIO tab PORT A
    - Set all bits as input with pull down resistor connected and deactivated interrupts.

3.3.2 Software initialization

GIO ports are initialized by calling gioInit() function.

3.3.3 Reading values

Reading from DIN is realized by direct reading values from PORTB GIO input register. There are constants and functions defined in file din.h, which facilitate access to DIN.

3.4 HOUT

H-bridge is an output realized by circuit AUIR33401S.

The input pin IN has two functions. In running state it is used as an input for PWM signal and its output value copies this input signal. In case of fault it is grounded to low.

IFBK output pin can be used for analog measurement of load on the out pin.

3.4.1 HALCoGen settings

Input IN and output DIAG is connected to HET. IFBK is connected to ADC.

- Enable driver HET1
- Enable driver ADC2
- In settings HET1 tabs Pin 16-23 and Pin 24-31 set
  - Bit 16, 18, 20, 22, 25, 29 as output with pull down resistor.
  - Bit 17, 19, 21, 23, 27, 31 as input with pull down resistor.
- In settings ADC2 tab Group1
  - Tick Enable Channel ID Conversion Results
  - Set software trigger
  - Enable channels 2 – 7
- Enable VIM channel 51: ADC2 Group1

3.4.2 Software initialization

HET is initialized by hetInit(), ADC by adcInit(). For proper function of reading IFBK the IRQ has to be enabled by using a macro _enable_IRQ(). Before IFBK reading starts, it is necessary to enable notification by adcEnableNotification() function. After the conversion finishes, it is good to disable those notification.

3.4.3 Reading and writing values

Writing values to IN pins and reading from DIAG pins is done by direct access to input and output register of HET port.

Reading values from IFBK is done similary to reading from ADC1 (see 3.2.3).

There are constants and functions defined in file hout.h, which facilitate access to HOUT.

3.5 MOUT

MOUT connected to subsidiary integrated circuits VNH5019A provides motor drivers. Inputs INA and INB determine direction of motor rotating. Pins ENA and ENB have to be connected to pull up resistor and are used to enable or disable circuit A and B or for diagnostic fault state.

In case of fault ENA and ENB are driven low by the IO itself and are not therefor reacting on external values changes.

3.5.1 HALCoGen settings

- Enable driver GIO
- Enable driver MIBSPI5
- Enable driver HET1
- In settings GIO tab PORTB set:
  - Bits 0, 1, 4, 5 as input with pull up resistor.
  - Bits 2, 3, 6, 7 as output with pull down resistor.

- In settings MIBSPI5 Port:
  - Set SIMO 1, 2 as GIO, input and with pull up resistor.
- In settings HET1 Pin 8 – 15:
  - Set bits 9, 14 as output with pull down resistor.

3.5.2 Software initialization

It is necessary to initialize GIO, HET and MIBSPI by calling functions gioInit(), hetInit() and mibspiInit().

3.5.3 Reading and writing values

Writing values to IN and EN pins is done by direct writing to the output GIO register of appropriate port.

Reading values from DIAG pins is done by direct reading from the input GIO register.

There are constants and functions defined in file mout.h, which facilitate access to MOUT.

3.6 SCI

Serial communication interface used for communication between the device and computer.

3.6.1 HALCoGen settings

- Enable driver SCI
- Enable VIM kanál 64: SCI Level 0 Interrupt
- Enable in SCI global config RX INT and TX INT
- Data format settings:
  - Baudrate: 9600 Hz
  - Parity: none
  - Bits: 8
  - Stopbits: 1
  - Flow control: none
- Set SCI Port TX as output and RX as input.

3.6.2 Software initialization

SCI is initialized by calling function sciInit(). For proper function, the IRQ has to be enabled by using a macro _enable_IRQ(). The cmdproc library is initialized by calling initCmdProc(), which includes also buffers initialization (initIoBuffer()).

3.6.3 Data transmission

Data receiving needs to be initiated by calling function sciReceive() at first. After that, receiving continues in an ISR.

Data transfer, with the use of interruptions, is done by direct interrupt invoking by setting flag in

SETINT register (simultaneously with setting BUF_TRANSFER_IN_PROGRESS flag) and by writing characters into the output buffer, from which the characters are cutted and sent.

There are functions defined in file cmdio_tisci.c, which facilitate access to SCI.

3.6.4 SDRAM

Processor TMS570LS3137 is able to address up to 128 MB additional memory of type SDRAM. The device has SDRAM slot and current software implementations supports memory initialization and testing.

3.6.5 HALCoGen settings

- Enable driver EMIF
- In settings EMIF general
    - Tick only Enable EMIF SDRAM
- In settings EMIF SDRAM
    - Fill memory chip timing according its documentation

3.6.6 Software initialization

External memory is initialized by calling function emif_SDRAMInit().

3.6.7 Accessing the memory

External memory is mapped to an address space 0x80000000 – 0x87FFFFFF. It is necessary to define volatile pointer on the address 0x80000000 and with that pointer we can access the external memory in similar way as we access internal memory.

4 Commands

4.1 Command definition

The list of commands with their logic is defined in file commands.c. The cmdProc is able to walk through the list composed of sublists, command descriptors or termination elements NULL.

Each command is defined by a descriptor. The most important parameters of descriptor is a pointer to a function and text showed as a help.

The function contains the commands logic and is defined by that prototype:

```
int cmd_do_commandname(cmd_io_t *cmd_io, const struct cmd_des *des, char *param[])
```

Command function prototype.

By implementing the function, creating the descriptor and inserting the pointer to the descriptor into the list before NULL element, new command is created.

4.2 List of commands

4.2.1 ADCCON

ADCCON

Tests connection of all ADC channels on ADC1 port. For each channel checks, whether it is shorted to REFHI or REFLO or it is all right. REFHI and REFLO are referential voltages.

4.2.2 CANLOOPBACK

CANLOOPBACK

Tests CAN bus loopback. Sends test message from CAN1 to CAN2 and prints number of errors that occurred during transmission or OK.

4.2.3 GETPIN

GETPIN* PIN

Suffix can be ADC1, DIN, HOUTD, HOUTF, MOUTE, VBAT.

Reads input pin PIN of port according to the suffix.

4.2.3.1 Examples

GETPINADC1 3 – Reads value of 3rd channel AD converter ADC1.

GETPINDIN 7 – Reads value of 3rd pin of port DIN

GETPINMOUTE 9 – Writes error message, MOUT does not have 9 pins.

4.2.4 HELP

HELP <CMD_NAME>

If no parameter is given, prints all commands and their help texts. Else prints help text to given command name

4.2.4.1 Examples

HELP – prints whole help

HELP ADCCON .– prints help for ADCCON command.

4.2.5 HOUTFAIL

HOUTFAIL

Tests if HOUT is in fault state.

4.2.6 MOUTFAIL

MOUTFAIL

Tests if MOUT is in fault state.

4.2.7 READVAL

READVAL*

Suffix can be ADC1, DIN, HOUTD, HOUTF, MOUTE.

Reads values on input pins of port according to the suffix.

4.2.7.1 Examples

READVALADC1 – Reads values from AD converter

READVALDIN – Reads values from Digital Input

READVALHOUTD – Reads values from pins HOUT_DIAG

READVALHOUTF – Reads values from pins HOUT_FBK

READVALMOUTE – Reads values from pins MOUT_EN

4.2.8 SETPIN

SETPIN* PIN VAL

Suffix can be MOUTI, HOUTI, VBAT.

Sets pin PIN of port according to suffix to value VAL.

4.2.8.1 Examples

SETPINMOUTI 3 1 – Sets pin 3 MOUT_IN of port MOUT on log 1

SETPINHOUTI 7 0 – Prints an error, bacause pin HOUT_IN does not have 7 pins. If it does, it would set 7th HOUT_IN pin of port HOUT to log 0

4.2.9 TESTSDRAM

TESTSDRAM

Tests if SDRAM module is connected, if it does, calculate its capacity.

4.2.10 VBATSET

VBATSET

Sets the value of global enable VBAT_EN to 1.

4.2.11 VBATGET

VBATGET

Gets the value of global enable VBAT_EN

4.2.12 VBATCLR

VBATCLR

Sets the value of global enable VBAT_EN to 0.