
TECHNICAL UNIVERSITY
SOFIA


medis

Material Resources

- The course requires classroom-laboratories with the following equipment:
  - Computer workstations (one for every workplace)
  - Specialized and standard microcontroller development kits (two for every workplace). One of the kits will act as a controller and the other as the simulated object.
  - For programming the applications is used a free kickstart edition of the development environment, which will allow reducing the cost of the project.

Material Resources

- The resources required for the “Process controllers and simulators” module are:
  - Personal computer.
  - Power source.
  - Oscilloscope (optional)
  - Multimeter (optional)
  - Two standard microcontroller development kits - STM32Discovery
  - Two specialized extension motherboards for STM32Discovery
  - IAR Embedded Workbench for ARM software – free kickstart edition
  - Internet access.

Hardware description

- Base development kit – STM32Discovery kit
- Features
  - 32-bit ARM Cortex-M4F core, 1 MB Flash, 192 KB RAM
  - Board power supply: through USB bus or from an external 5 V supply voltage
  - Integrated STLink debugger onboard
  - USB OTG FS with micro-AB connector
  - Extension header for all LQFP100 I/Os for quick connection to prototyping board and easy probing
  - IAR Embedded Workbench for ARM development environment – kickstart edition

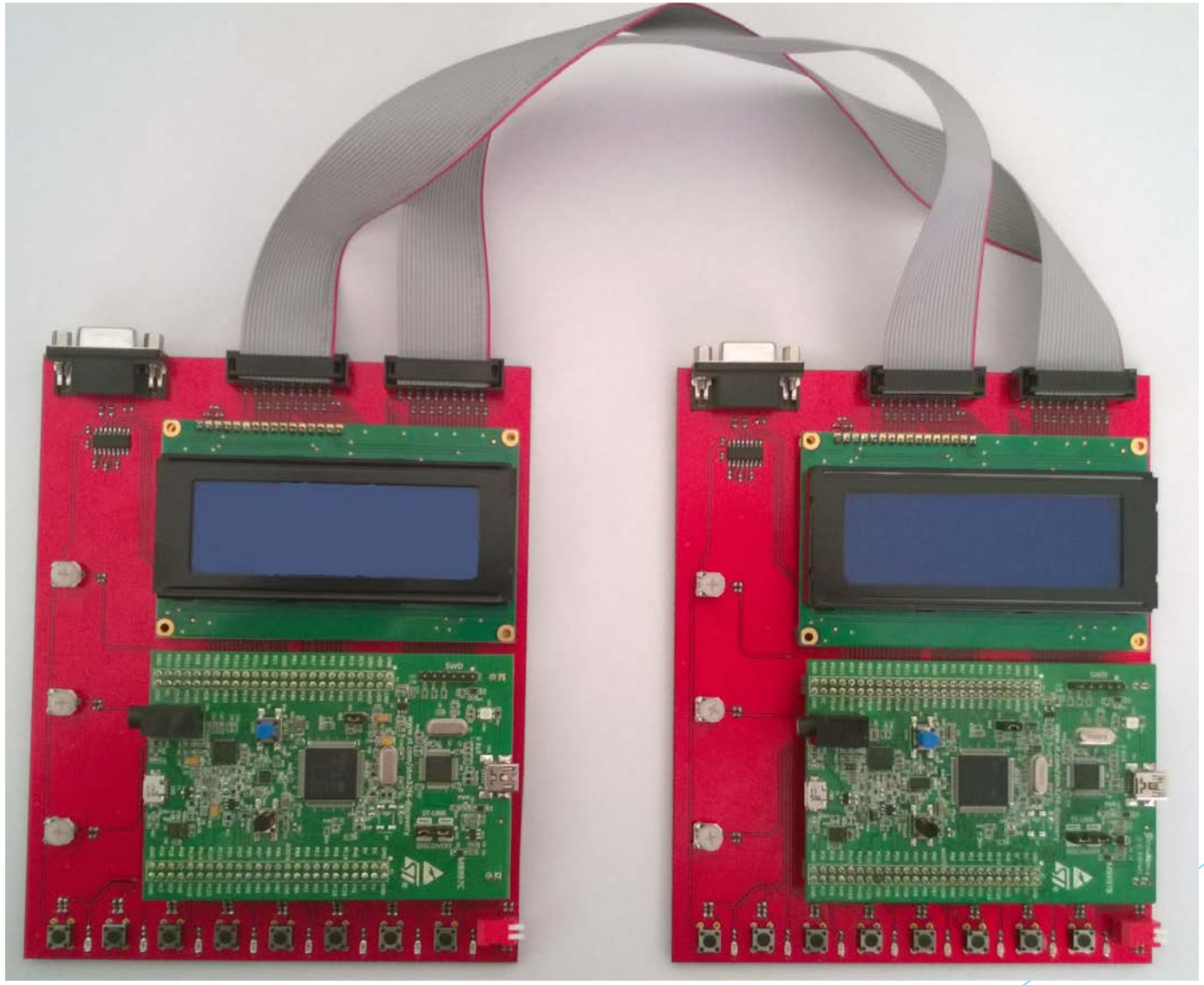
Hardware description

- Custom extension board main features:
  - 8 buttons and 2 switches
  - 8 LED
  - 3 trimmers
  - RS232 port for PC connection
  - LCD character display 20x4

Hardware description

- Two 20 pin headers for connection between the controller and simulator board
  - 16 digital inputs
  - 16 digital inputs
  - 2 analog inputs
  - 2 analog outputs
  - 1 USART interface

Hardware description



Software description

- The software used for making the simulations is based on STM32Discovery library and further simplified to allow the students to concentrate their efforts into designing and implementing the simulation models rather than programming the microcontroller peripherals.
- Each of the microcontroller peripherals used by the extension board has predefined simple name (BUTTON_X, DIN_X, DOUT_X, SWITCH_X etc. with X corresponding the device number). Each of the above devices has additional definition allowing to be renamed in order to achieve more readable code. For example:
  - #define BUTTON_1 BUTTON_START_CYCLE
  - #define INPUT_1 INPUT_SCALE_READY

Software description - Manually operating peripherals definitions

BUTTONS	LEDS	TRIMMERS	SWITCHES
BUTTON_1	LED_1	TRIMMER_1	SWITCH_1
BUTTON_2	LED_2	TRIMMER_2	SWITCH_2
BUTTON_3	LED_3	TRIMMER_3	
BUTTON_4	LED_4		
BUTTON_5	LED_5		
BUTTON_6	LED_6		
BUTTON_7	LED_7		
BUTTON_8	LED_8		

Software description – Headers peripheral definitions (for simulation)

DIGITAL INPUTS		DIGITAL OUTPUTS		ADC	DAC
INPUT_1	INPUT_2	OUTPUT_1	OUTPUT_2	ADC_1	DAC_1
INPUT_3	INPUT_4	OUTPUT_3	OUTPUT_4	ADC_2	DAC_2
INPUT_5	INPUT_6	OUTPUT_5	OUTPUT_6		
INPUT_7	INPUT_8	OUTPUT_7	OUTPUT_8		
INPUT_9	INPUT_10	OUTPUT_9	OUTPUT_10		
INPUT_11	INPUT_12	OUTPUT_11	OUTPUT_12		
INPUT_13	INPUT_14	OUTPUT_13	OUTPUT_14		
INPUT_15	INPUT_16	OUTPUT_15	OUTPUT_16		

Universal defined constants

- To achieve further code readability the following defines are used:
  - ON and OFF – used by the library functions for operation of the digital inputs, digital outputs, switches and LEDs.
  - PRESSED and NOT_PRESSED – used by the library function returning the button states.
  - MIN_ANALOG_VALUE and MAX_ANALOG_VALUE – reconfigurable for 8 bit, 10 bit or 12 bit ADC and DAC mode.
  - ELAPSED or NOT_ELAPSED – used for timer checks.

Software library description

- GET and SET functions for buttons, switches, LEDs, digital inputs, digital outputs, ADCs, DACs.
- Implemented simple serial communication.
- Implemented MODBUS protocol for communication between the controller and the simulator boards. Controller board is configured as MODBUS master and the simulator board is configured as MOSDBUS slave.
- Implemented simplified timer library with one shared physical timer working with interrupts on 1ms and 16 virtual timers.

Software library description

- int GetButtonState(int ButtonNumber) - Returns the button state (PRESSED or NOT_PRESSED). Receives the button number.
- int GetDigitalInput(int InputNumber) – Returns the digital input state (ON or OFF). Receives the input nuber.
- int GetSwitchState(int SwitchNumber) – Returns the switch state (ON or OFF). Receives the switch number
- int GetOutputState (int OutputNumber) – Returns the output state for debug purposes. (ON or OFF). Receives the output number.
- int GetTrimmerValue(int TrimmerNumber) – Returns the analog value from the trimmer as an integer between 0 and 1023 (10 bit ADC configuration for easier scaling and debugging). Receives the trimmer number.

Software library description

- int GetAnalogInput(int InputNumber) – Returns the analog value as an integer between 0 and 1023. Receives the analog input number.
- int GetDigitalOutput(int OutputNumber) – Returns the DAC value as an integer between 0 and 1023. Receives the DAC number.
- void SetDigitalOutput(int OutputNumber, int state) – Sets the digital output state. Receives the output number and desired state (ON or OFF).
- void SetAnalogOutput(int AnalogOutput, int value) – Sets the DAC state. Receives the DAC number and the desired value as an integer between 0 and 1023.

Software library description – timers subsystem

- Predefined timer constants TIMER_1 – TIMER_16.
- void DisableVTimers(void);
- void EnableVTimers(void);
- void SetVTimerValue(int TimerNumber, int ticks) – Starts or reloads the corresponding virtual timer. Receives the timer number and the given number of ticks in milliseconds.
- int GetVTimerValue(int TimerNumber) – Returns the number of ticks in milliseconds since the start of the virtual timer. Receives the timer number. If the timer is not starter returns 0.
- int IsVTimerElapsed(int TimerNumber) – Returns ELAPSED or NOT_ELAPSED. Receives the timer number.

Software library description – timers subsystem

- void ClearVTimer(int TimerNumber) – Disables the desired virtual timer. Receives timer number.
- int GetTimerCounter(void) – Returns the timer counter for debug purposes.
- int ResetTimerCounter(void) – Resets the timer counter for debug purposes.

Software library description – communication subsystem

- Both slave and master boards have standard communication settings that can be changed to illustrate the influence of the communications on simulator and controller behavior.

Implemented simple library for serial communications.

- void InitSerial(void) – Initializes the serial communications with the predefined parameters.
- unsigned char USARTGetByte(void) – reads single byte from the USART receive buffer.

Software library description – communication subsystem

- int outString(unsigned char *Str, int len) – sends a string or array of bytes. Receives a pointer to the string or array or string constant and the length in bytes. In case of sending strings the terminating zero should be counted in the string length.
- int inString(unsigned char *Str, int len) – Receives the desired number of bytes in the string pointer. Receives the string pointer, and the expected number of bytes. Returns the number of bytes read.

Software library description – MODBUS communication subsystem

- Implemented simple MODBUS library both for the master and the slave.

Function name	Command number
Read Coils	1
Read Discrete Inputs	2
Write Single Coil	5
Write Multiple Coils	15
Read Holding Registers	3
Write Multiple Holding Registers	16

Software library description – MODBUS communication subsystem

- Implemented possibility one simulator board to emulate multiple devices on the communication bus, allowing simulation of system with multiple smart sensors or actuators.
- The simulator board can support up to 10 slaves each of them having the following MODBUS map.

Device name	Number	Start address
Output coils	16	0
Discrete inputs	16	0
Holding registers	100	0

Software description

- Display – 2 modes
  - 4 up to 10 symbol labels and up to 10 digit numbers on each display row (table 4x2)
  - 8 up to 9 digit numbers and 1 symbol label (table 4x4)
  - Custom *printf* function for each mode
    - printf4(“Label”, Numeric_value, row)
    - printf8(“Label”, Numeric_value, row, column)

Exercises

- Exercises cover design and development of 3 different simulators
  - Elevator model – shows the students how to simulate a discrete system.
  - Liquid tank model – shows the student how to simulate analog system.
  - Concrete plant model – shows the students how to emulate a complex discrete and batching system.
- Each simulator is built during 4 exercises
  - Exercise one – Design of the simulator behavior. Students helped by the tutor should identify all simulator subsystems, specify inputs and outputs and the simulator behavior.
  - Exercise 2 and 3 - Development and test of the different simulator subsystems.
  - Exercise 4 – Integration of all the subsystems and test of the simulator behavior.

The END