Implementing CPDev runtime in a target controller

A short summary how to implement the CPDev runtime in your target platform

The CPDev runtime interprets and executes the binary intermediate code produced by the CPDev compiler as .XCP file. Commands in the binary code are platform-independent and the runtime is called the CPDev Virtual Machine (VM).

The VM source is provided as a set of C code files. Usually you will not need to modify the files (consult us if you feel you should). Instead, you will implement the platform interface functions in a way that is specific to your solution. You will be provided with sample implementation of the interface. Please refer to the sample during reading of this document.

The VM requires two memory areas (64kbytes each by default) to be reserved for operation: data memory (RAM) and program memory (ROM or RAM). Setting up the VM requires pointers to these areas (*pgmData*, *pgmCode*) to be set properly. CPDev VM can also work with smaller or larger code and data memory areas if required.

The runtime uses a set of platform interface functions (declared in *vmplatform.h*). You should provide the body of these functions. The following summary describes the most important platform functions.

int VMP_LoadProgramAndData(const char* filename, int datasize)

The function is used to setup the VM for execution. In the body, you should set *pgmData* and *pgmCode* pointers to the data and code memory areas. Using the *filename* and *datasize* parameters is not mandatory. If the binary program is available on the target in the form of .XCP file, you can pass its name upon call to the function. If the binary is loaded directly to the program memory (e.g. via a wire connection), you can skip this. The parameter *datasize* determines the required size for data memory area and can be set according to the information provided by the CPDev compiler in the .DCP file. You should call VMP_LoadProgramAndData before starting the VM program.

void VMP_PreCycle(void)

The function is called automatically before performing calculation cycle (when VM executes a program). This is the place where you will read inputs and provide their values to the CPDev program by storing them in the data memory. Usually you will require the controller to execute a PLC program cyclically with a defined constant cycle. To achieve this, you have to store the system clock value at the beginning of the cycle. The information will be used later to determine the time when the next cycle should start (see below).

void VMP_PostCycle(void)

The function is called automatically after performing calculations in the PLC calculation cycle. This is the place where you will set outputs by reading the calculated values from the data memory. Here you will also read the system clock and compare it with the value from VMP_PreCycle. To keep the cycle constant (to some extent), you should wait (sleep) for the amount of time that is left until the full cycle is reached. Use the global variable *task_cycle* for comparison. Single-task environments can use the remaining time to perform some other operations such as communications, hardware testing, etc. Without the waiting, the program will be executed as endless loop.

void VMP_PreProgram(void)

The function is called automatically once before the program starts. It is usually used to set seed for pseudo-random generator.

WM_TIME VMP_CurrentTime(void)

The function is used by the VM to get current value of the system clock. You should provide the value in milliseconds. This function is crucial for time-dependent operations (such as delayed timers).

void VMP_ReadRTC(WM_DATE_AND_TIME *dt)

The function is used by the VM to get current value of the real-time clock. You should convert the value returned by your RTC clock to the WM_DATE_AND_TIME structure (refer to the sample implementation).

WM_BOOL VMP_WriteRTC(WM_DATE_AND_TIME *dt)

The function is used by the VM to set current value of the real-time clock. This can be useful if you want the CPDev program to set the clock programmatically, e.g. to handle daylight saving. You should convert the WM_DATE_AND_TIME structure to the format your RTC clock accepts. The result of the function indicates whether the setting has succeeded.

WM_REAL VMP_GetRandom(void)

The function is used by the VM to get a random value. The value should be a REAL number.

Other functions

The file *vmplatform.h* declares some other platform-dependent functions. They should be implemented for more sophisticated scenarios, such as debugging, implementing native function blocks or maintaining flash memory. You can leave them empty for now. However if you would like to implement the advanced features, the sample implementation code can help you with this.

Handling inputs and outputs

The data memory area is read or written by the VM during operation. This is also the place where you should put your input values or get output values (as said in *VMP_PreCycle* and *VMP_PostCycle* above). The CPDev compiler provides information (.DCP file) on locations (addresses) of program variables so you can use the information to map I/Os to variables. More information about possible options of I/O handling can be found in our corresponding paper “Interfacing Inputs and Outputs with IEC 61131-3 Control Software”.

Starting execution

The following steps are necessary to start execution of the CPDev program on the target:

1. Load the contents of XCP file to the target, move it to the code memory indicated by *pgmCode* (this can be done in *VMP_LoadProgramAndData*).
2. Make sure that *pgmData* points to a free block of RAM memory.
3. Set the *task_cycle* variable to the required PLC cycle (in milliseconds).
4. Call WM_Run(WM_MODE_FIRST_START | WM_MODE_NORMAL) (see vm.h for other run flags).

If you have more questions about the runtime, please write a message to btrybus@prz.edu.pl.