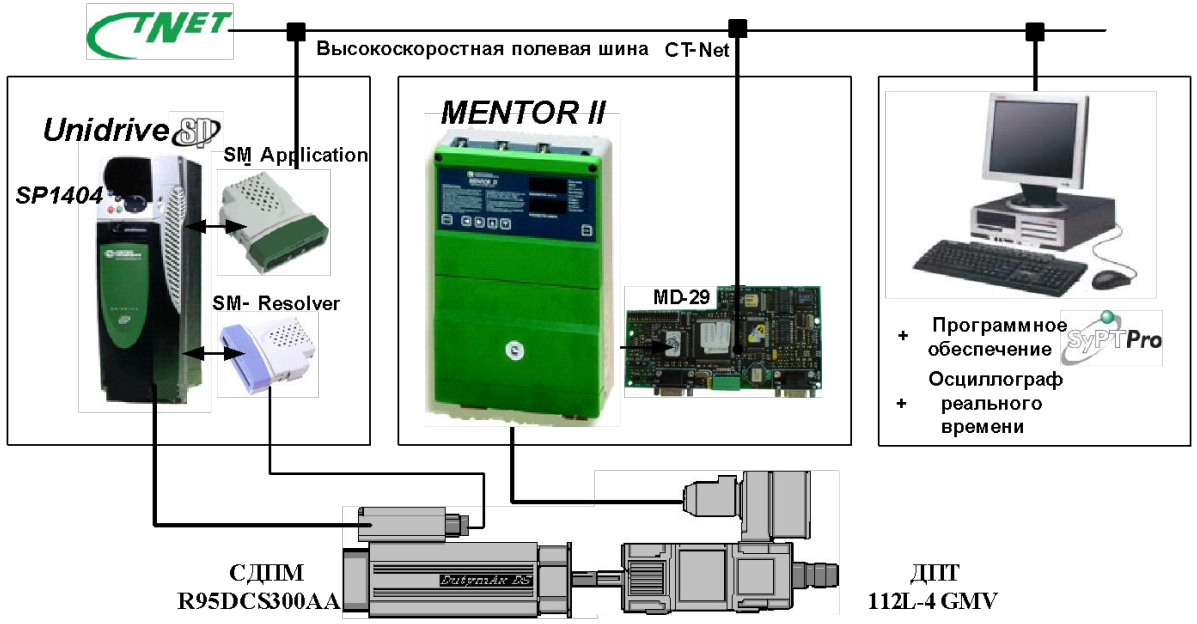
Программирование сопроцессорного модуля SM-Application преобразователя частоты Unidrive SP с помощью программного обеспечения SYPT Pro

В состав лабораторного комплекса входят две электрические машины: СДПМ *Dutymax DS* типа *R95DCS300AAA* и двигатель постоянного тока (ДПТ) независимого возбуждения *112L-4MGV,* установленные на одном валу и жестко соединенные между собой. ДПТ используется в качестве нагрузочной машины для синхронного электродвигателя. Для управления СДПМ используется частотный преобразователь *Unidrive SP1404*, а для управления ДПТ – тиристорный преобразователь *MENTOR II.*



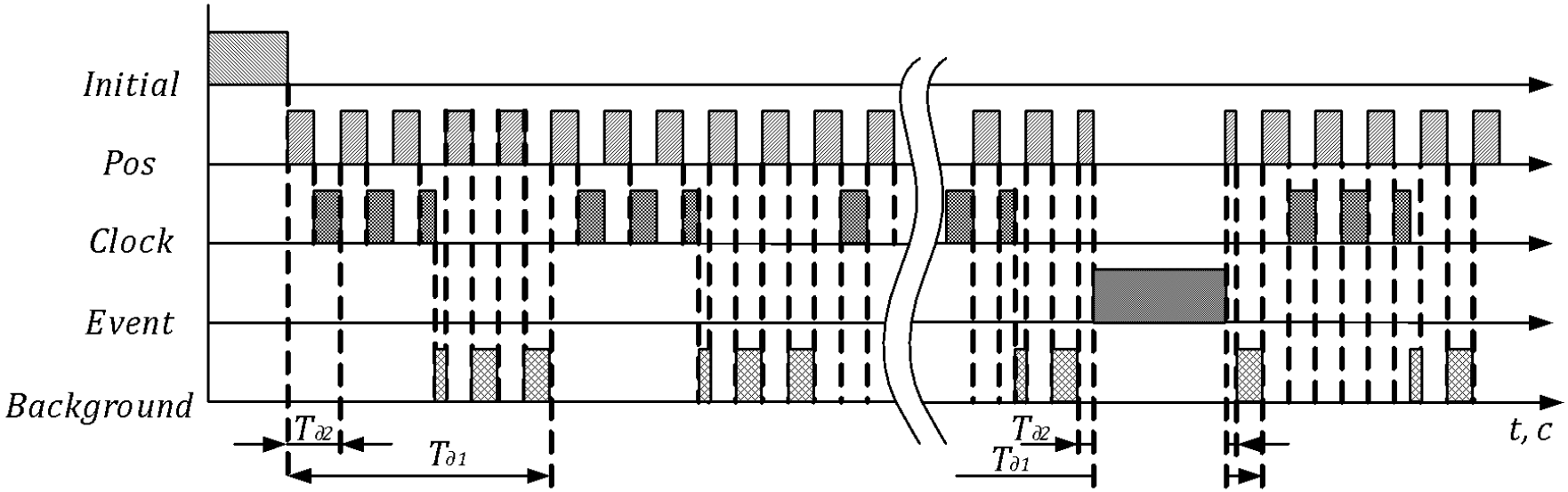
ПЧ укомплектован также сопроцессорным модулем *SM-Application*, а тиристорный преобразователь – модулем *MD29*. Оба этих дополнительных модуля предназначены для реализации приложений пользователя. Модуль *SM-Application* построен на основе высокопроизводительного специализированного процессора, оснащен *Flash*-памятью емкостью 384кБ, оперативной памятью емкостью 80кБ. Модуль *MD29*, реализованный на 32-х битовом RISC-процессоре INTEL 960; имеет 96 кбайт флэш-памяти и 8 кбайт оперативной памяти. В корпусе синхронной машины установлен датчик положения типа резольвер. Связь датчика положения с преобразователем частоты осуществляется через дополнительный модуль *SM-Resolver*. С помощью высокоскоростной шины *CT-Net* обеспечивается обмен данными между преобразователями и компьютером, также входящим в состав стенда и предназначенным для программирования дополнительных модулей в среде *SyPT-Pro* и обеспечения работы виртуального осциллографа *ST-Scope* в режиме реального времени.

Управление модулем *SM-Application*

Пользовательское приложение состоит из отдельных разделов (задач), которые выполняются в строго определенной последовательности. К таким разделам относятся (в порядке приоритета): *«Initial»*, *«Event»*, *Pos»*, *«Clock»* и *«Background»*. При подаче питания на ПЧ первыми выполняются инструкции, записанные в разделе *«Initial»*, в котором задаются значения констант и начальные значения сигналов системы управления, а также определяется ее конфигурация. После этого начинают выполняться задачи реального времени разделов *«Pos»* (их может быть несколько, например *«Pos0»* и *«Pos1»*) и *«Clock»*. Инструкции, помещенные в данные разделы, циклически повторяются через фиксированные интервалы времени (периоды дискретности). Период дискретности для задачи *«Clock»* (*Тд1*) может принимать целочисленные значения от 1 до 200 мс, а для задач *«Pos0»* и *«Pos1»* (*Тд2*) – строго фиксированные значения: 250 мкс, 500 мкс, 1 мс, 2 мс, 4 мс и 8 мс.

Инструкции разделов *«Event»* имеют самый высокий приоритет, поэтому их задачи содержат очень малое число инструкций. Они прерывают работу разделов *«Pos»* и *«Clock»*, и только по окончании их выполнения программа управления продолжает прерванные инструкции разделов *«Pos»* и *«Clock».* Таким образом, в разделы *«Event»* целесообразно помещать алгоритмы обработки определенных событий, например, аварийных ситуаций.

«Background» – является фоновой задачей. Она выполняется только в паузах между выполнениями инструкций других разделов. Эта задача организуется в виде бесконечного цикла. Если она будет завершена, то она больше не будет выполняться.



Как видно из временной диаграммы, выполнение инструкций раздела *«Clock»* прерывается выполнением раздела *«Pos»* (он имеет более высокий приоритет, чем *«Clock»,* и меньший период дискретности). Поэтому разработчики рекомендуют в разделы *«Pos»* помещать инструкции, связанные с корректировкой контуров регулирования скорости и (или) положения, а в раздел *«Clock»* - инструкции, не требующие такого быстрого выполнения как задачи раздела *«Pos»*, например формирование задатчика положения.

Все инструкции программы должны располагаться только внутри определенной задачи. Инструкции разделов *«Pos* и *«Clock»*, должны быть выполнены за время, меньшее, чем их периоды дискретности; иначе задачи с меньшими приоритетами (выполняемые в паузах этих задач) не получат времени для своего выполнения. Это может привести к отключению процессора по перегрузке.

Среда разработки пользовательских программ управления *Sypt-Pro* содержит библиотеку стандартных команд и функций, а также поддерживает несколько языков программирования (DPL – язык инструкций, LD – язык лестничных диаграмм, FBD – язык функциональных блоков).

В рассматриваемой среде разработки определены 3 типа переменных: целочисленная, на которую в памяти отводится 32 бита (из них 1 бит знаковый); переменная с плавающей запятой обычной точности, также занимающая 32 бита (1 бит – знак, 8 – порядок, 23 – мантисса); переменная с плавающей запятой с двойной точностью (64 бита, из которых 1 бит – знаковый, 11 – порядок, 52 – мантисса).

Параметры, инициализируемые в разделе *«Initial»*, можно разделить на три группы:

1) параметры, инициализирующие подключение различных устройств;

2) глобальные переменные, значения которых изменяются в дальнейшем в процессе работы частотного преобразователя *Unidrive SP*;

3) переменные, не предусмотренные проектировщиками (локальные переменные сопроцессорного модуля *SM-Application*).

Имена переменных, относящихся к первым двум группам, имеют следующий формат #<*Номер-меню*>.<*Номер-параметра*>. Размерность их значений определена проектировщиками и не может быть изменена пользователем. Перечень меню и их параметров, а также диапазон и размерность возможных значений приведены в документации [1-5]. Например, оператор «*#2.02=1*» предписывает подать задание на РС от ЗИ, а оператор «*#2.02=0*» – подать задание на РС скачком; оператор «*#2.11=0.2*» определяет темп разгона как время в секундах, затрачиваемое на изменение скорости на 1000 об/мин.

Имена переменных, относящихся к третьей группе, задаются пользователем по следующим правилам: состав имени – латинские буквы, цифры и знак подчеркивания; первый символ – буква. Тип переменной определяется ее именем. По умолчанию все переменные считаются переменными с плавающей запятой с двойной точностью. Для того чтобы сделать локальную переменную целочисленной необходимо дополнить ее имя в конце знаком «%».

Кроме переменных, рассмотренных выше в примерах, в разделе *«Initial»* рассматриваемой программы заданы: период такта выполнения раздела *«Clock»*, времена характерных точек желаемой тахограммы и желаемой нагрузочной диаграммы *t_1%*, …, *t_5%*, коэффициенты обратных связей *L1* и *Kc*, начальное значение интегратора *W_c*, на котором построен НС и его постоянная интегрирования *Ti*. Все времена заданны в долях периода дискретности.

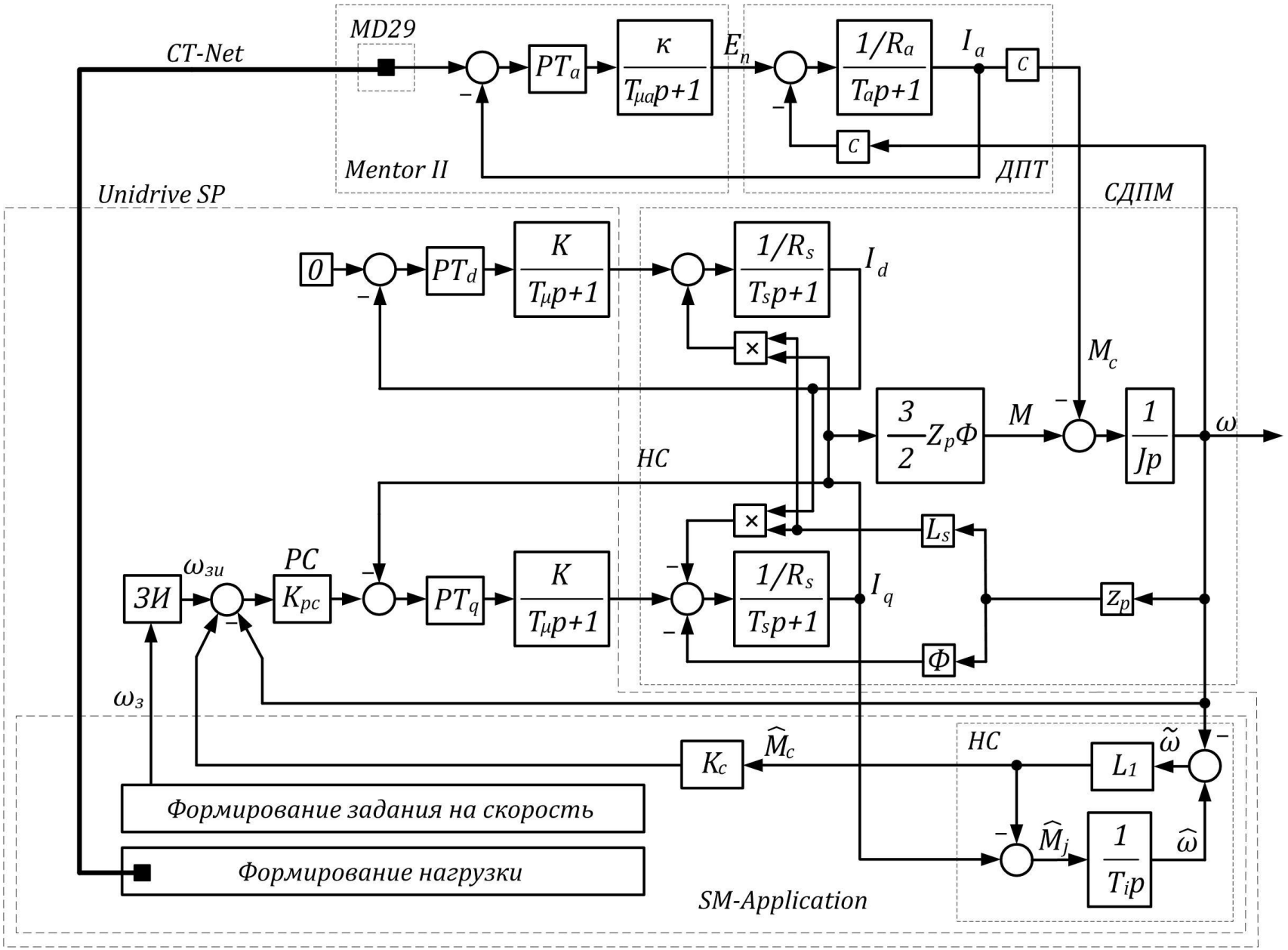
В разделе *«Clock»* выполнена программная реализация наблюдателя состояния, сформированы сигналы задания на скорость, на нагрузку и корректирующая связь по идентифицированному моменту сопротивления, а также выполнена запись сигналов, необходимых для визуализации переходных процессов. Здесь использованы следующие глобальные и локальные переменные: *#1.18, #3.2* – заданная и измеренная скорость двигателя [об/мин] соответственно, *#4.2* – моментообразующая составляющая тока статора [А], *#3.22* – корректирующая обратная связь [об/мин], заведенная на вход РС, *#6.30* – разрешение вращения вперед, *#6.32* – разрешение вращения назад, *load%* – разрешение формирования нагрузки (1 – разрешение, 0 – запрет), *t%* – счетчик выполняемых тактов.

*И*нтегратор, входящий в состав НС, реализован при помощи стандартной функции интегрирования *ITerm(Error%, iGain%, Limit%, Pset%, PsetE%, Reset%, Hold%)*», где *Error%* – интегрируемое значение, *iGain%* –умноженный на 1000 коэффициент усиления интегратора, *Limit%* – ограничение интегратора, *Pset%* – начальное условие интегратора, *PsetE%* – сброс интегратора в состояние *Pset%*, *Reset%* – сброс интегратора в 0, *Hold%* – удерживание текущего значения интегратора.

В разделе *«Background»* записаны инструкции, формирующие задание на контур тока якоря нагрузочной машины. Бесконечный цикл реализуется с помощью метки «*top:»* и оператора безусловного перехода «*goto»*. Здесь использованы следующие сетевые функции:

1) *Q% = CheckNode(Node%)* – функция, проверяющая наличие устройства в сети *CT-Net*, где *Node%* – адрес вызываемого устройства, которым в данном случае является модуль *MD29* с сетевым адресом 11.

2) *status% = WRNET(Node%, Menu%, Parametr%, Value%, PosD%, WaitCode%)* – функция, записывающая данные в другое устройство по сети, где *Node%* – адрес принимающего устройства, *Menu%*, *Parametr%* – номера меню и параметра, из которых формируется имя глобальной переменной для записи информации, *Value%* – пересылаемая информация, *PosD%* – количество знаков в дробных частях пересылаемых чисел, *WaitCode%* – предельное время, предоставляемое функции для передачи информации и подтверждения завершения этой операции [мс].

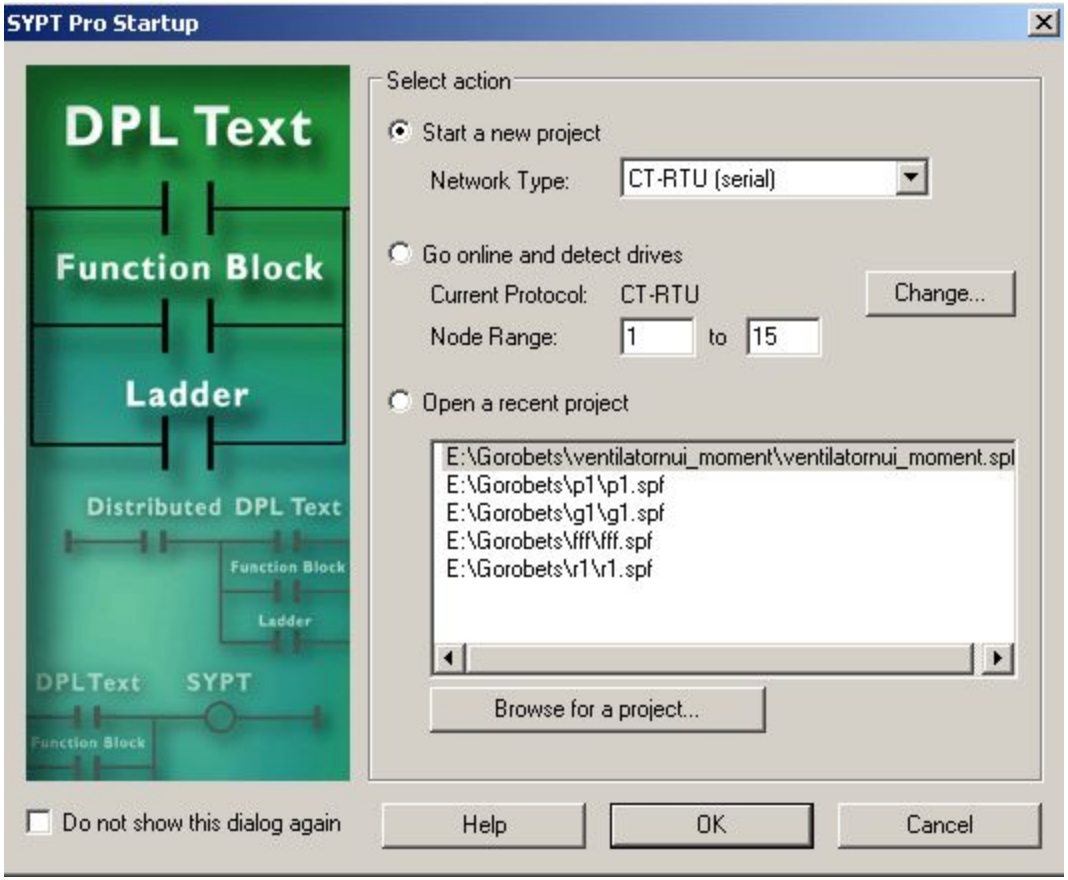


Структурная схема

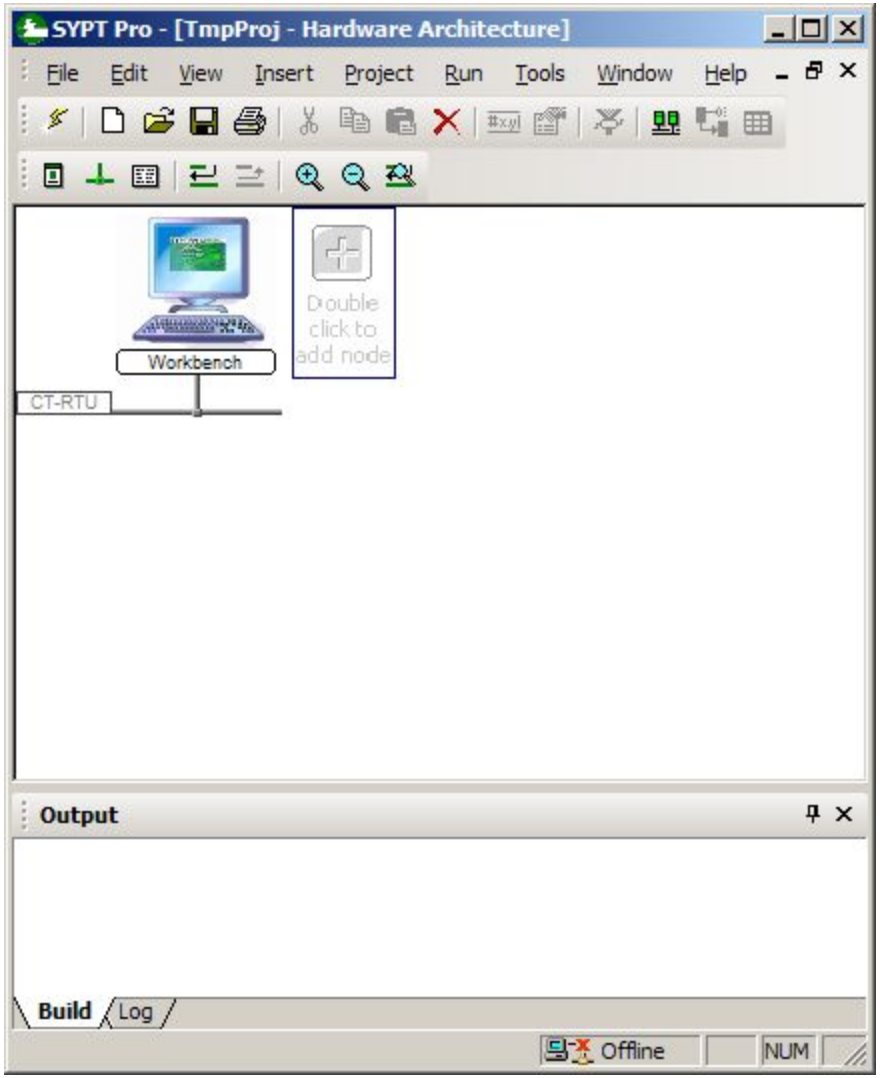
```
Initial{                                    // Открытие раздела «Initial»
#1.14=5                                     // Выбор источника задания скорости (0 – аналоговый вход, … , 5 – точное задание)
#15.11=5                                    // Такт выполнения раздела «Clock» = 5 мс
#2.02=1                                     // Включение ЗИ
#2.11=0.2                                   // Темп разгона (увеличение скорости на 1000 об/мин происходит за 0,2с)
#2.21=0.2                                   // Темп торможения (уменьшение скорости на 1000 об/мин происходит за 0,2с)
#3.23=1                                     // Разрешение корректирующей связи
t_1%=200                                    // Время начала разгона
t_2%=700                                    // Время от начала разгона до начала торможения
t_3%=300                                    // Время от начала торможения до обнуления счетчика
t_4%=200                                    // Время от начала разгона до наброса нагрузки
t_5%=200                                    // Длительность воздействия нагрузки
L1=0.025                                    // Параметр НС
Кс=1.7                                      // Коэффициент усиления корректирующей связи
Ti=627.0                                    // Постоянная интегрирования модели объекта
Nmax%=2147483647                            // Максимальная величина целочисленной переменной (2^31-1)
W_cn=0                                      // Нулевое начальное значение интегратора
}                                           // Закрытие раздела «Initial»

Clock{                                      // Открытие раздела «Clock»
M_cn=(#3.2-W_cn)*L1                         // Вычисление
M_jn=(#4.2-M_cn)                            // наблюдателя
W_cn=(ITerm(((#4.2-M_jn)*Ti), 1000, Nmax%, 0, 0, 0))   // состояния,
#3.22=M_cn*Кс                               // Формирование корректирующей связи
load%=0                                     // Запрет на формирование нагрузки
if ((#6.30=1)or(#6.32=1)) then              // Проверка разрешения на вращение
        if ((t%>t_1%)and(t%<t_1%+t_2%)) then    // t_1%<t%<t_1%+t_2%
        #1.18=1500                          // Задание на скорость =1500
        If ((t%>t_1%+t_4%)and(t%<t_1%+t_4%+t_5%)) then   // t_1%+t_4%<t%<t_1%+t_4%+t_5%
            load%=1                         // Разрешение формирование нагрузки
        endif
        else
            #1.18=0                         // Задание на скорость =0
        endif
        if (t%>t_1%+t_2%+t_3%) then         // t%>t_1%+t_2%+t_3%
            t%=0                            // Обнуление переменной t%
        endif
        t%=t%+1                             // Приращение переменной t%
else
        t%=0                                // Обнуление переменной t%
endif
#18.11=#4.2*1000                            // Занесение в память сигналов скорости,
#18.12=M_cn*1000                            // идентифицированного статического момента,
#18.13=#3.2*4                               // полного тока двигателя
}                                           // Закрытие раздела «Clock»
```

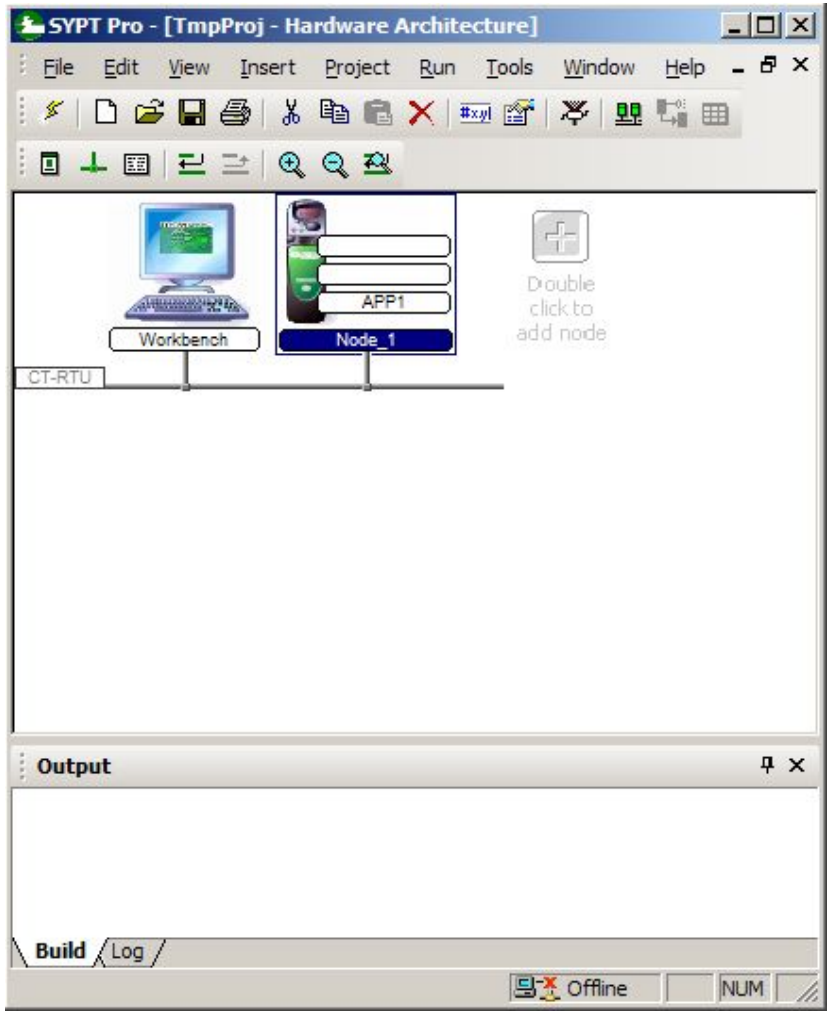
```
Background{                                    // Открытие раздела «Background»
top:                                           // Метка
if (CheckNode(11)=1) then                      // Проверка наличия устройства в сети
        if (load%=1) then                      // Пересылка данных по сети
           status%=WRNET(11,4,8,45,0,100)      // Пересылка данных по сети
        else                                   // Оператор безусловного перехода
           status%=WRNET(11,4,8,0,0,100)       // Закрытие раздела «Background»
        endif
endif
goto top:
}
```



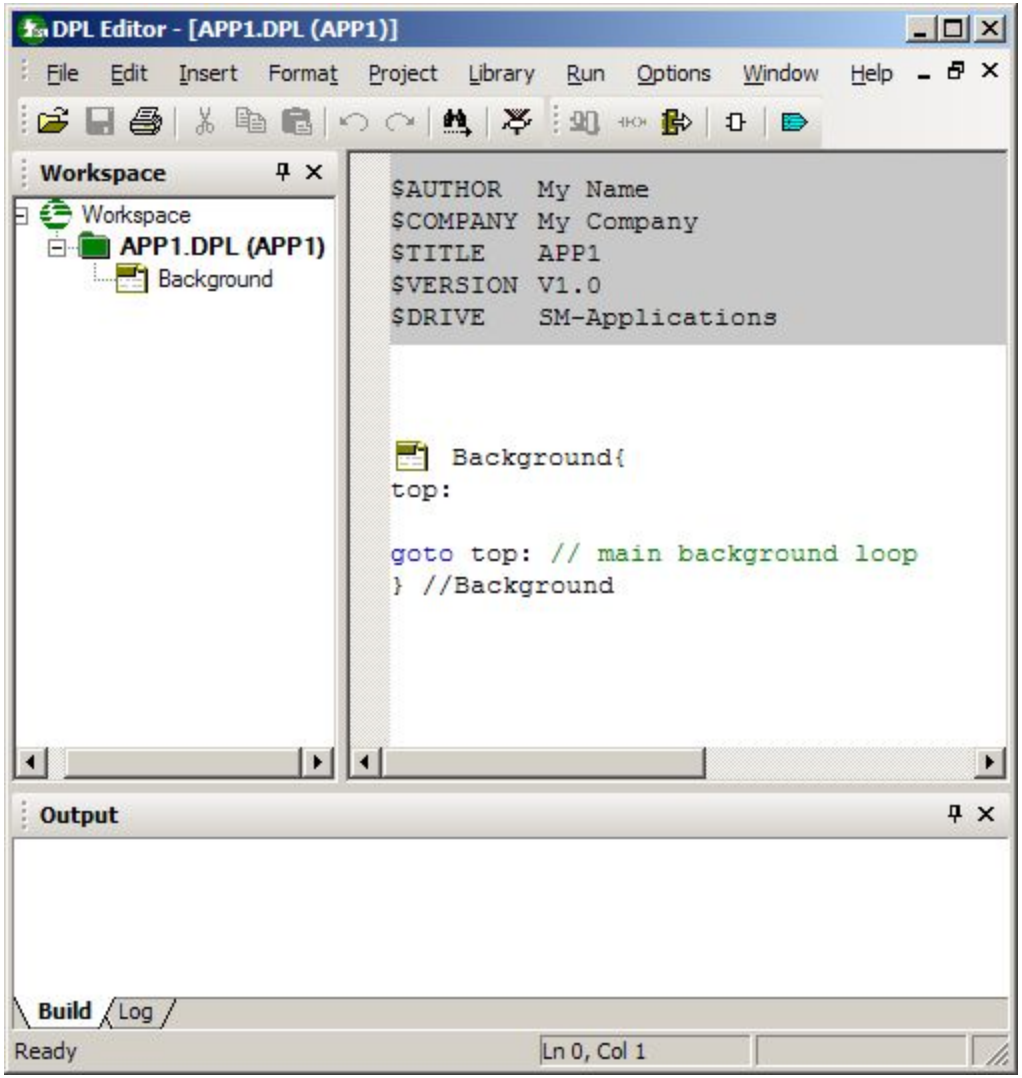
Стартовое окно SYPT Pro



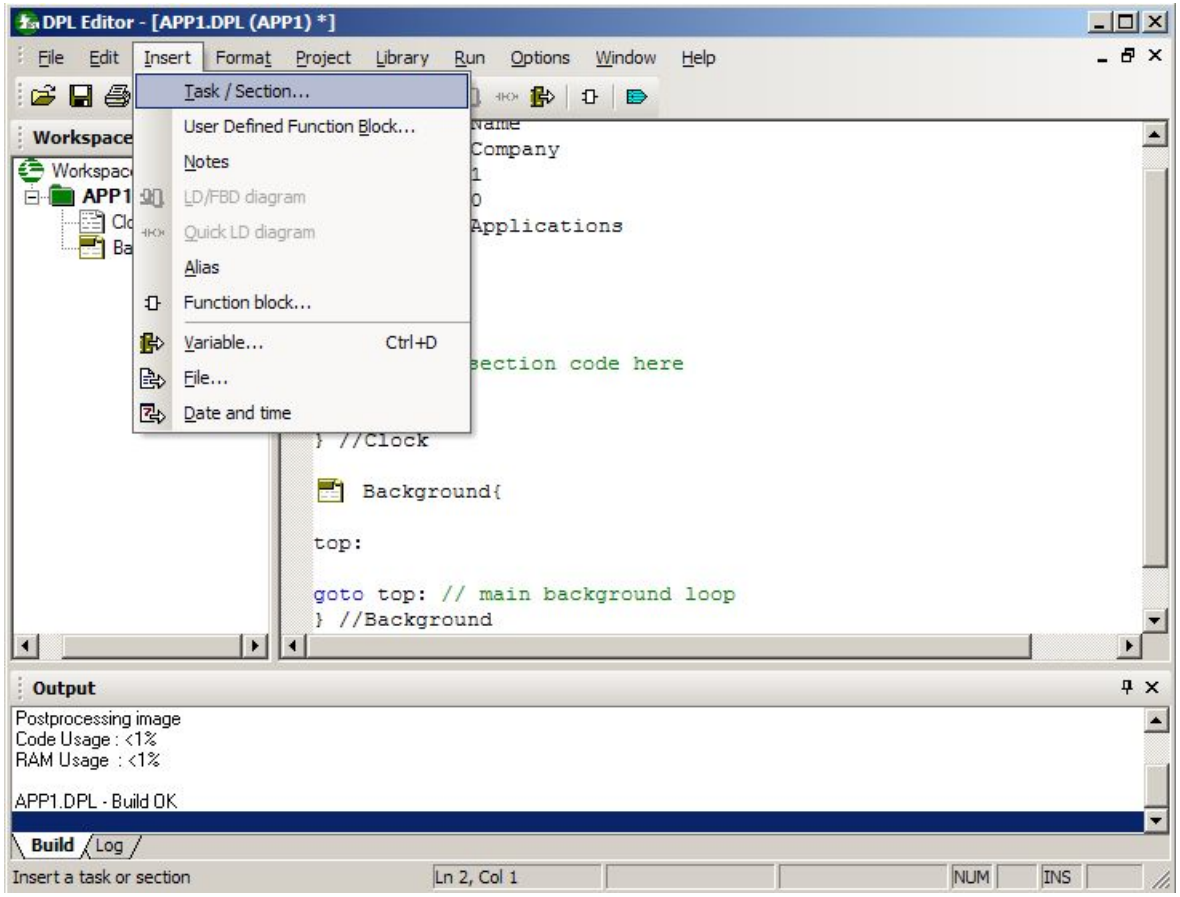
Окно архитектуры оборудования



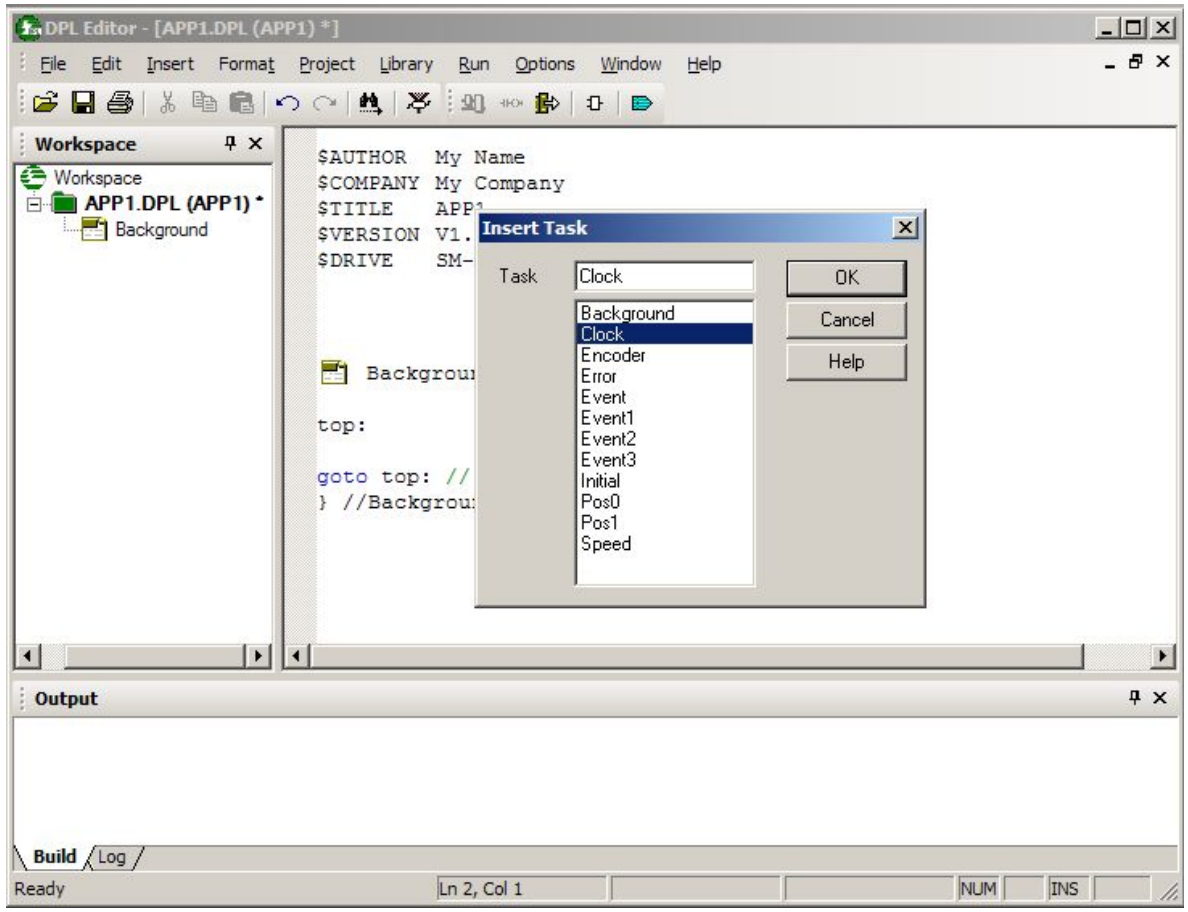
Архитектура оборудования (добавлено новое устройство)



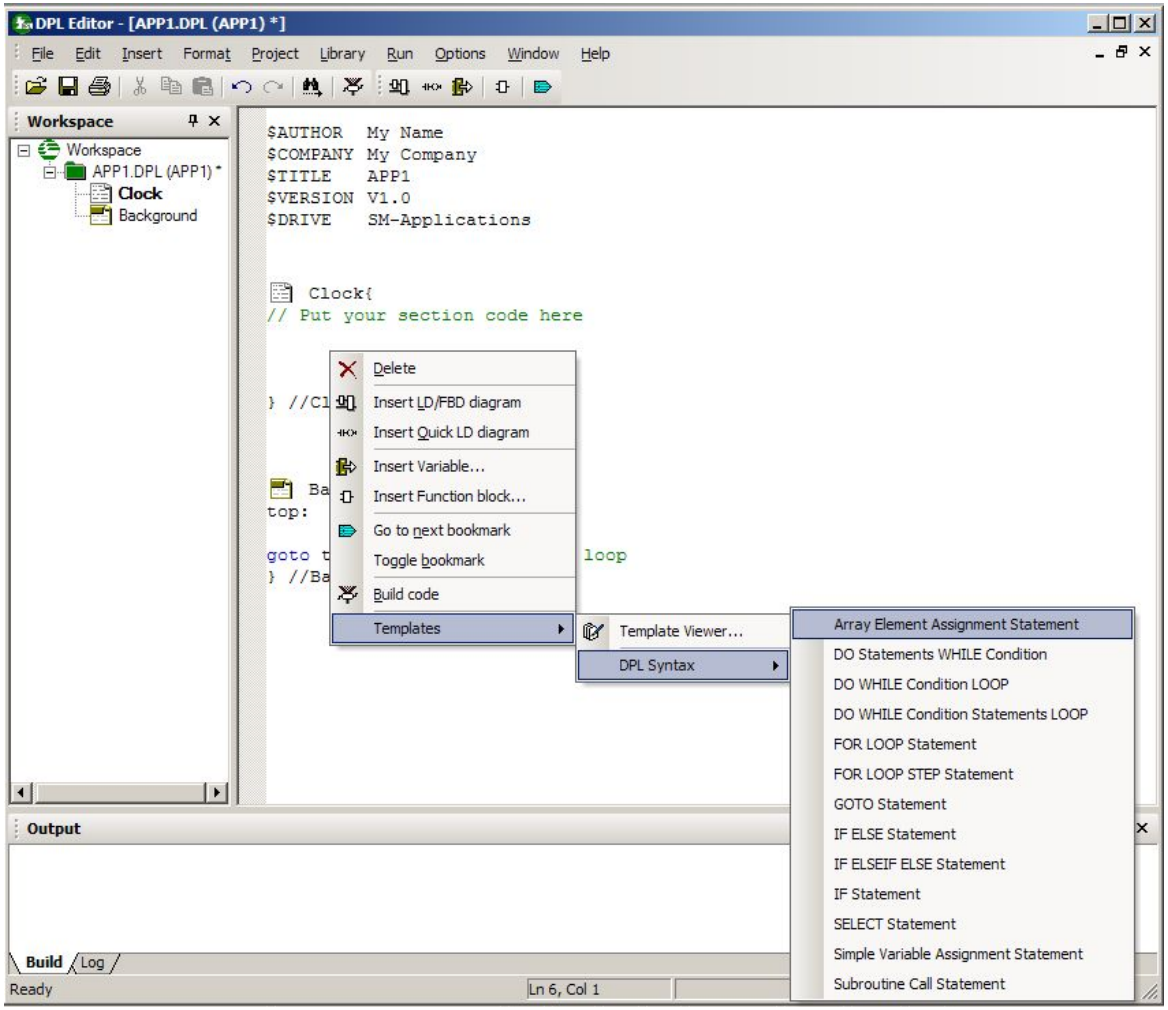
Окно редактирования и отладки программы



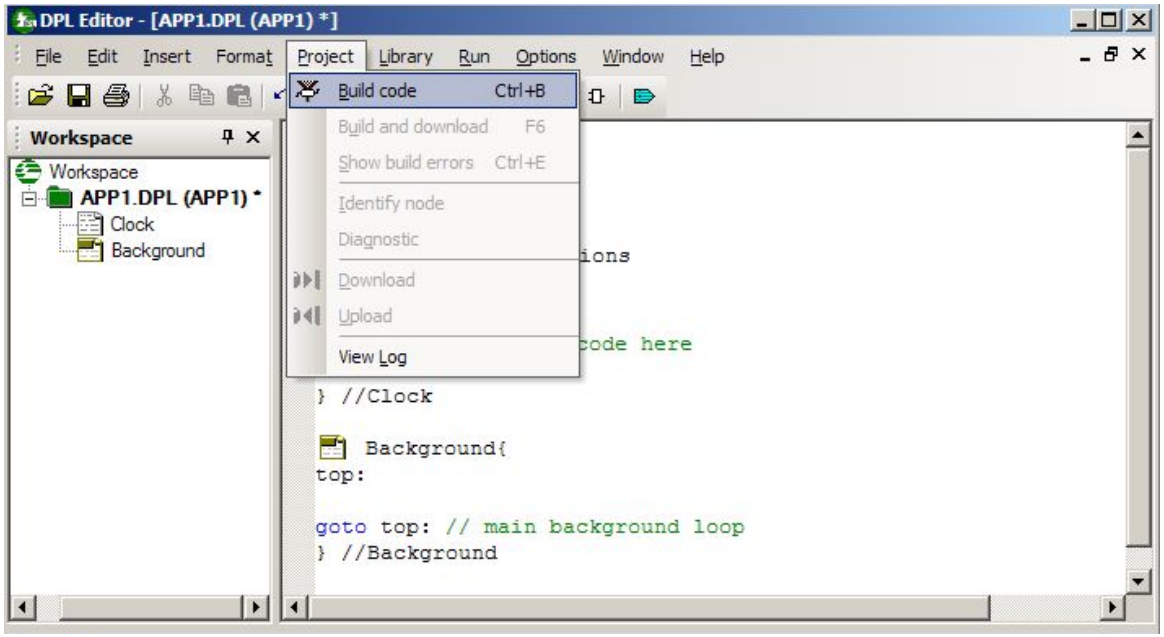
Добавление в программу нового раздела



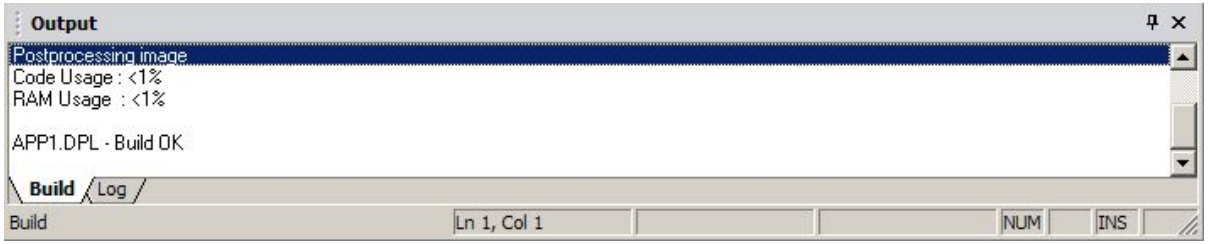
Окно выбора
раздела



Пример добавления
шаблона



Компиляция
программы



Окно вывода с сообщением об успешной
компиляции

DPL Editor - [APP1.DPL (APP1) *]

File Edit Insert Format Project Library Run Options Window Help

Workspace

- Workspace
  - APP1.DPL (APP1) *
    - Initial
    - Clock
      - **Diagram 1**

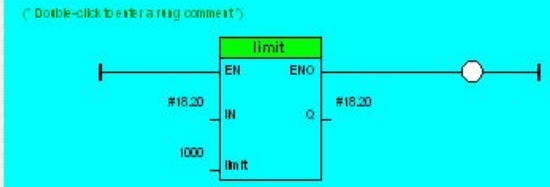
```
$AUTHOR   MainUser
$COMPANY  My Company
$TITLE    APP1
$VERSION  V1.0
$DRIVE    SM-Applications

Initial{
    // Set RAMP up and down rates.
    UpRate%       = 500
    DownRate%     = 500

    // Default the limits for the ramp to 2500
    #18.20        = 2500

    // Make the Clock task run once every 10 milliseconds.
    // For targets other than SM Applications or SM Applications Lite
    // this line of code can be left out.
    #81.11        = 10
    reinit
} //Initial


Clock{
// Limit the RAMP to +/- 1000 using a Function Block diagram.
Diagram 1
```

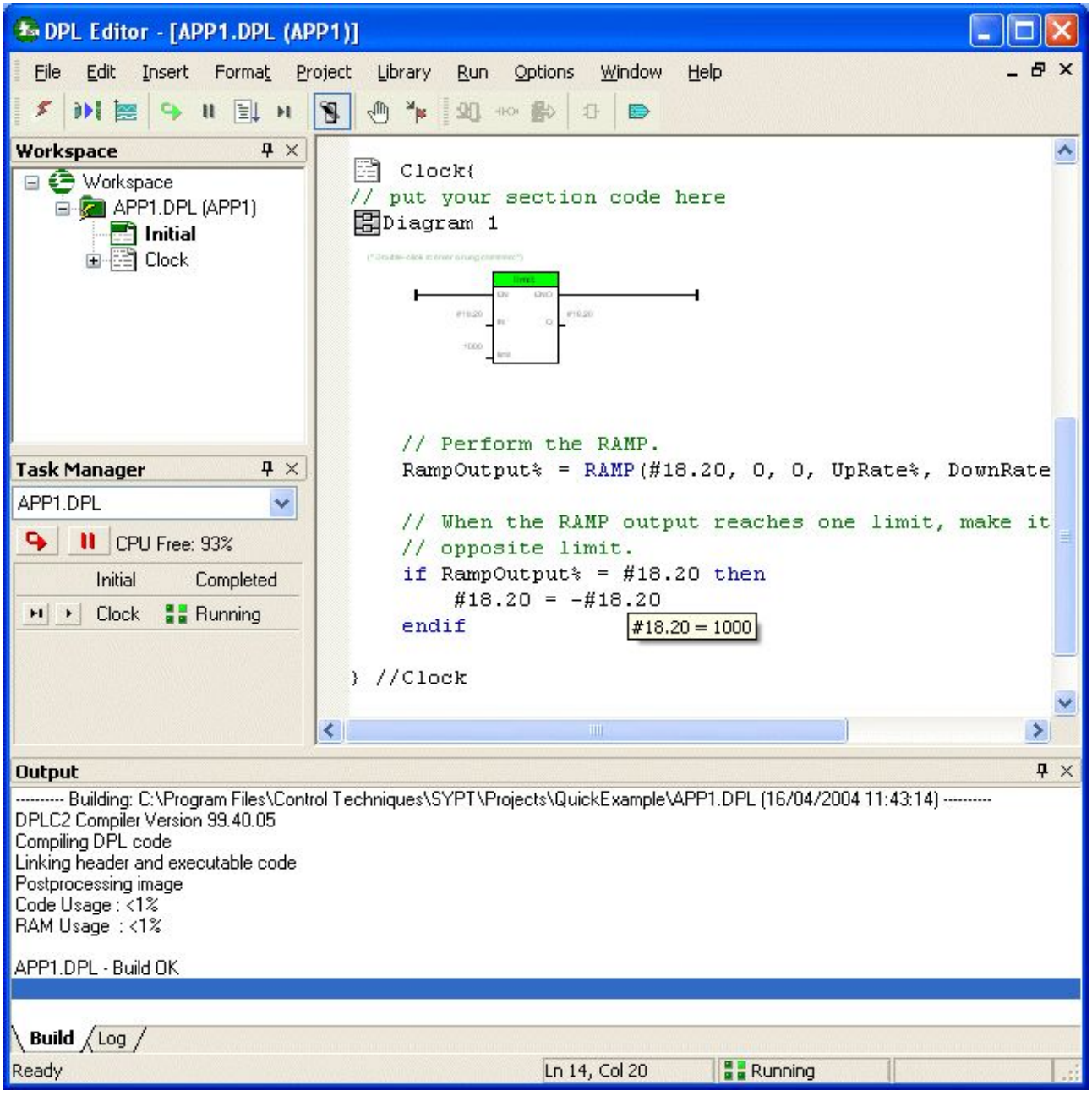


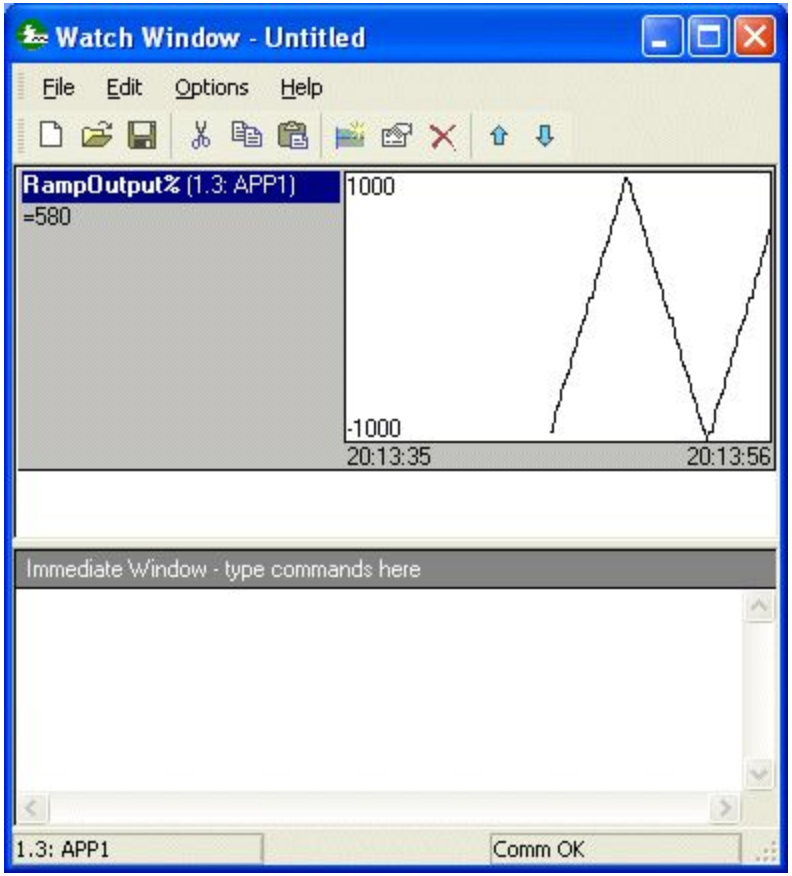

```
    // Perform the RAMP.
    RampOutput% = RAMP(#18.20, 0, 0, UpRate%, DownRate%, 0)

    // When the RAMP output reaches one limit, make it RAMP towards the
    // opposite limit.
    if RampOutput% = #18.20 then
        #18.20 = -#18.20
    endif

} //Clock
```

Ready | Ln 20, Col 1 | INS


DPL Editor - [APP1.DPL (APP1)]
File Edit Insert Format Project Library Run Options Window Help
Workspace
Workspace
APP1.DPL (APP1)
Initial
Clock
Task Manager
APP1.DPL
CPU Free: 93%
Initial Completed
Clock Running
Clock{
// put your section code here
Diagram 1
// Perform the RAMP.
RampOutput% = RAMP(#18.20, 0, 0, UpRate%, DownRate
// When the RAMP output reaches one limit, make it
// opposite limit.
if RampOutput% = #18.20 then
#18.20 = -#18.20
endif
#18.20 = 1000
} //Clock
Output
--------- Building: C:\Program Files\Control Techniques\SYPT\Projects\QuickExample\APP1.DPL (16/04/2004 11:43:14) ---------
DPLC2 Compiler Version 99.40.05
Compiling DPL code
Linking header and executable code
Postprocessing image
Code Usage : <1%
RAM Usage : <1%
APP1.DPL - Build OK
Build Log
Ready
Ln 14, Col 20
Running


Watch Window - Untitled
File
Edit
Options
Help
RampOutput% (1.3: APP1)
=580
1000
-1000
20:13:35
20:13:56
Immediate Window - type commands here
1.3: APP1
Comm OK

Function Block	Description
%	Modulo operator (remainder)
!, !(a,b)	Bit invert operator
&, |, ^	Bitwise AND, OR, XOR operators
+,-,/,*	Basic operators
<,<=,=,>,>=,<>	Comparison operators
ABS	Returns the absolute of a value
AND4	Four-input binary AND gate
ANSIREAD	Sends a read request to another unit
ANSIREADN	Sends a read request to another unit
ANSIREPLY (UD70 & MD29) ANSIREPLY (SM-Applications)	Gets a reply from a read or write request
ANSIWRITE	Sends a write request to another unit
ANSIWRITEN	Sends a write request to another unit
ARCTAN	Returns the arctangent of an angle
ARRAYGET	Gets an array element
ARRAYSET	Sets an array element
ARRAYSORT	Sorts an array
ARRAYSTAT	Returns information on an array
ASSRAM	Associate files with arrays
AUTOSYNC	CTNet Sync Master Redundancy
AUTOSYNCERROR	Error block used in conjunction with AUTOSYNC
AVERAGE	Returns the running average of a value
BCD2BIN	Converts from BCD to BINary
BCD2SEG	Converts from BCD to seven-segment LED data
BIN2BCD	Converts from BINary to BCD
BLKDEF	Block parameter write
BRIDGE	High speed protocol bridge
CAMBOX	CAM function generators
CAMBOX2	
CAMINIT	Initialises CAM table for position controller (UD70, MD29 targets only)

ConvertForwards	Multiplies a value by the Numerator and divides by the Denominator
ConvertBackwards	Multiplies a value by the Denominator and divides by the Numerator
CRC16	Calculates cyclic redundancy check.
CTD	Down-Counter
CTU	Up-Counter
CTUD	Up/Down Counter
CTNETDIAGNOSTICS	CTNet diagnostics
CTSyncDisableOutputChannel	Disable CTSync output channel
CTSyncEnableOutputChannel	Enable CTSync output channel
CTSyncGetSlaveReferences	Get reference data generated by the CTSync Master
CTSyncSetMasterReferences	Generate CTSync reference data
CTSyncSetupOutputChannel	Setup the CTSync output channel
CTSyncWriteOutputChannel	Write data to the CTSync output channel
DATASTUFF	Stuff bits into variable/parameter
DECODER4	Decodes a decimal value to four binary outputs
DECODER8	Decodes a decimal value to eight binary outputs
DELAY	Delay
DIGSEL8	Digital select, eight-input
iDMUX/iDMUX8 fDMUX	De-multiplexer blocks
DTERM	Differentiator block
EDITFASTLINK	Edits a fast cyclic data link
EDITSLOWLINK	Edits a slow cyclic data link
EGB	Electronic gearboxes
EGB2	
EnableCANTrips	Specifies that CAN events trip the drive
ENCODER4	Forms a decimal value from four binary inputs
ENCODER8	Forms a decimal value from eight binary inputs
EXP	Returns the exponential function of a value
FILTER	First-order digital filter
FLOAT	Converts from integer to floating point
FLOAT32	Converts a floating point value into IEEE single precision and forces into a DPL 32-bit integer
GETCAN	Receives a CAN frame from a specified CAN slot

GETCHAR	Receives a single character on the RS485 port
GETKEY	Receives a single character on the RS232 port
GetNextSmartCardFile	Returns the details of the next file on the SmartCard
GetParAttr	Returns the attributes of a drive or option module parameter
GetSamplesPerSec	Returns the number of samples per second for the current realtime task
GetTaskID	Returns ID for the current Task
HOME1 HOME2	Homing function blocks
HiLoDet	Detects the highest and lowest values that the input reaches.
INT	Converts from floating point to integer
iInvert/fInvert	Integer/floating point inversion

ITERM	Integrator block
iWindow	Shows the relationship between two integers within a specified tolerance window
LIMIT	Specify upper limit for integer or floating point values
iLIMIT/fLIMIT iLIMIT2/fLIMIT2	Specify upper and lower limits for integer/floating point values
LinInt	Linear interpolation
LN	Returns the natural logarithm of a value
LOWER	Returns the lower bound of an array
LT,LE,EQ,GE,GT,NE	Comparison operators (in FBD editor-equivalent to <,<=,=,>,>=,<>)
MAX	Returns the maximum of two values
MIN	Returns the minimum of two values
MODEXFER	Drive mode change
MOVE	Assign an input to an output

MPOT	Motorised pot
MULDIV	64-bit multiply and divide
MULDIVRM	64-bit multiply and divide with remainder
iMUX/iMUX2/iMUX8 fMUX/fMUX2	Multiplexers
NAND4	Four-input NAND
NEG	Negates an input value
NETREPLY	Returns the value read from a RDNET command
NETSTATUS	Returns network driver status
NOR4	Four-input NOR
OpenReadSmartCard	Open SmartCard for reading
OpenWriteSmartCard	Open SmartCard for writing
OR4	Four-input OR
iOS/fOS	Integer/floating point offset and scale a value
PFIXREAD PFIXREAD6	Fixed precision parameter read (3 decimal places) Fixed precision parameter read (6 decimal places)
PFIXWRITE PFIXWRITE6	Fixed precision parameter write (3 decimal places) Fixed precision parameter write (6 decimal places)
PID POSLOOP POSLOOP2	PID Blocks
PRECISIONTIMER	Measure pulse width
PT	Pulse Timer
PTD	Pulse Timer (extended)
PUTBLOCK	Transmit array of data via RS485 port
PUTCAN	Place CAN frame in CAN slot ready for transmission
PUTCHAR	Sends a single character on the RS485 port
PUTKEY	Sends a single character on the RS232 port
PutBottomWord PutTopWord	Word manipulation
PWM	PWM digital output
RAMLength	Returns the size of a RAM file
RAMP	Linear ramp generator block

RDNET	Read a parameter or block of parameters from another CTNet node
RDNETB	
ReadNet	
ReadNetB	
ReadReadOnlyBit	Returns the read-only status of the SmartCard
ReadSmartCardByte	Reads a byte of data from the SmartCard
REG	Registration block
REINIT	Re-initialises target set-up
ResetCANTimer	Resets the CAN timestamp timer
RS	R-S bistable latch

RTUReadRegs RTUReadParas RTUReadInputRegs RTUPresetRegs RTUPresetParas RTUMasterReply RTUMasterStatus	RTU Master commands
RUNTIME	Returns the accumulative time the equipment has been running
SCAN	Returns the scan sample time of the current task
SCHEDULEEVENT	Schedule an Event task to execute
SetConverterDenominator	Denominator for user-defined unit conversion
SetConverterNumerator	Numerator for user-defined unit conversion
SetCTNSync	Dynamically change CTNet sync generation
SetRAMLength	Set the length of a RAM file
SetUserID	Set the user program ID (#81.49)
SGN	Returns the sign of a number
SHIFT	Shift register

SHIFTLR	Shift register
SignedBottomWord SignedTopWord	Word manipulation
SIN	Returns the sine of an angle
SLEW	Simple ramp generator
iSO/fSO	Integer/floating point scale and offset a value
SPLITTER	Splits a 32-bit integer input into two outputs each of x- bits long
SPGateway	CTNet to fieldbus gateway
Sq1Int/Sq2Int	Square interpolation
SQR	Returns the square-root of a number
SR	S-R bistable latch
SRAMP SSRAMP	S-Ramp profile generator blocks
STUFF2/STUFF4/STUFF8	Stuffs 2, 4 or 8 bits into a 32-bit word
TAN	Returns the tangent of an angle
TCYCLIC	Cyclic timer
TIME	Returns milliseconds since power-up
TOFF	Off-delay timer
TOFF2	Off-delay timer (extended)
TOFFRET	Retentive off-delay timer
TON	On-delay timer
TON2	On-delay timer (extended)
TONRET	Retentive on-delay timer
TRL	Triggered latch
TRUNC	Truncate floating point variable (convert to integer)
UnAssRAM	Dissociate files from arrays
UnsignedBottomWord UnsignedTopWord	Word manipulation
UPPER	Returns the upper bound of an array
VEL2POS	Velocity profile integrator
WDOG	Watchdog function
WRNET	Write a parameter or block of parameters to another CTNet node
WRNETB	
WriteNet	
WriteNetB	
WriteReadOnlyBit	Set the read-only bit of the SmartCard
WriteSmartCardByte	Writes a byte of data to the SmartCard
XOR4	Four-input XOR gate

Arrays	ARRAYGET	Gets an array element
	ARRAYSET	Sets an array element
	ARRAYSORT	Sorts an array
	ARRAYSTAT	Returns information on an array
	LOWER	Returns the lower bound of an array
	UPPER	Returns the upper bound of an array
Bit Manipulation	&, |, ^	Bitwise AND, OR, XOR operators
	!, !(a,b)	Bit invert operator
	AND4	Four-input binary AND gate
	DATASTUFF	Stuff bits into a variable/parameter
	NAND4	Four-input NAND
	NOR4	Four-input NOR
	OR4	Four-input OR
	SHIFT	Shift register

Bit Manipulation	SHIFTLR	Shift register
	SPLITTER	Splits a 32-bit integer input into two outputs each of x-bits long
	STUFF2/STUFF4/STUFF8	Stuffs 2, 4 or 8 bits into a 32-bit word
	XOR4	Four-input XOR gate
CAN Communications	CANConfigEvent	SM-Applications Event task trigger
	CANGetNodeAddr	Return CAN module's node address
	CANReady	CAN reset complete status
	CANSetup	CAN module configuration
	CANStatus	Get status of CAN slot
	CANStatusAll	Get status of all CAN slots
	EnableCANTrips	Specifies that CAN events trip the drive
	GETCAN	Receives a CAN frame from a specified CAN slot
	PUTCAN	Place CAN frame in CAN slot ready for transmission
	ResetCANTimer	Resets the timestamp of the CAN timer
Casting	INT	Converts from floating point to integer
	FLOAT	Converts from integer to floating point
	FLOAT32	Converts a floating point value into IEEE single precision and forces into a DPL 32-bit integer
	PutBottomWord PutTopWord SignedBottomWord SignedTopWord UnsignedBottomWord UnsignedTopWord	Generic casting operations
	TRUNC	Truncate floating point variable (convert to integer)
Coding and Multiplexing Functions	ENCODER4	Forms a decimal value from four binary inputs
	ENCODER8	Forms a decimal value from eight binary inputs
	DECODER4	Decodes a decimal value to four binary outputs
	DECODER8	Decodes a decimal value to eight binary outputs
	DIGSEL8	Digital select, eight-input
	iDMUX/iDMUX8 fDMUX	De-multiplexer blocks
	iMUX/iMUX2/iMUX8 fMUX/fMUX2	Multiplexer blocks

Communications	ANSIREAD	Sends a read request to another unit
	ANSIREADN	Sends a read request to another unit
	ANSIWRITE	Sends a write request to another unit
	ANSIWRITEN	Sends a write request to another unit
	ANSIREPLY (UD70 & MD29) ANSIREPLY (SM-Applications)	Gets a reply from a read or write request
	BRIDGE	High speed protocol bridge
	CRC16	Calculates CRC
	GETCHAR	Receives a single character on the RS485 port
Communications	GETKEY	Receives a single character on the RS232 port
	PUTCHAR	Sends a single character on the RS485 port
	PUTKEY	Sends a single character on the RS232 port
	RTUReadRegs RTUReadParas RTUReadInputRegs RTUPresetRegs RTUPresetParas RTUMasterReply RTUMasterStatus	RTU Master commands
	SPGATEWAY	CTNet to fieldbus gateway
Counters and Timers	CTU	Up-Counter
	CTD	Down-Counter
	CTUD	Up/Down Counter
	PT	Pulse Timer
	PTD	Pulse Timer (extended)
	PWM	PWM digital output
	TCYLIC	Cyclic timer
	TIME	Return milliseconds since power-up
	TON	On-delay timer
	TON2	On-delay timer (extended)

Counters and Timers	TONRET	Retentive on-delay timer
	TOFF	Off-delay timer
	TOFF2	Off-delay timer (extended)
	TOFFRET	Retentive off-delay timer
CTNet Communications	AUTOSYNC	CTNet Sync Master Redundancy
	AUTOSYNCERROR	Error block used in conjunction with AUTOSYNC
	CHECKNODE	Checks for CTNet node on network
	CTNETDIAGNOSTICS	CTNet diagnostics
	EDITFASTLINK	Edits a fast cyclic data link
	EDITSLOWLINK	Edits a slow cyclic data link
	NETREPLY	Returns the value read from a RDNET command
	NETSTATUS	Returns network driver status
	RDNET	Read a parameter or block of parameters from another node
	RDNETB	
	READNET	
	ReadNetB	
	SetCTNSync	Dynamically change CTNet sync generation
	WRNET	Writes to a parameter or block of parameters in another node
	WRNETB	
	WRITENET	
	WRITENETB	
CTSync Functions	CTSyncDisableOutputChannel	Disable CTSync output channel
	CTSyncEnableOutputChannel	Enable CTSync output channel
	CTSyncGetSlaveReferences	Get reference data generated by the CTSync Master
	CTSyncSetMasterReferences	Generate CTSync reference data
	CTSyncSetupOutputChannel	Setup the CTSync output channel
	CTSyncWriteOutputChannel	Write data to the CTSync output channel
Latch Functions	SR	S-R bistable latch
	RS	R-S bistable latch
	TRL	Triggered latch
Logic Functions	<,<=,=,>,>=,<>	Comparison operators
	LT,LE,EQ,GE,GT,NE	Comparison operators (in FBD editor-equivalent to <,<=,=,>,>=,<>)
	AND4 NAND4 OR4 NOR4 XOR4	Four-input Boolean logic functions
	CMP	Compares two values
	+,-,/,*	Basic operators

Mathematical and Signal Processing Functions	%	Modulo operator (remainder)
	ABS	Returns the absolute of a value
	AVERAGE	Returns the running average of a value
	COS	Returns the cosine of an angle

Mathematical and Signal Processing Functions	CRC16	Calculates CRC
	EXP	Returns the exponential function of a value
	FILTER	First-order digital filter
	FLOAT	Converts integers to floating point values
	FLOAT32	Converts a floating point value into IEEE single precision and forces into a DPL 32-bit integer
	INT	Converts floating points to integer values
	iInvert/fInvert	Integer/floating point inversion
Mathematical and Signal Processing Functions (Continued)	ARCTAN	Returns the arctangent of an angle
	ConvertForwards	Multiplies a value by the Numerator and divides by the Denominator
	ConvertBackwards	Multiplies a value by the Denominator and divides by the Numerator
	SetConverterDenominator	Denominator for user-defined unit conversion
	SetConverterNumerator	Numerator for user-defined unit conversion
	LinInt	Linear interpolation
	LN	Returns the natural logarithm of a value
	MAX	Returns the maximum of two values
	MIN	Returns the minimum of two values

Mathematical and Signal Processing Functions	MULDIV	64-bit multiply and divide
	MULDIVRM	64-bit multiply and divide with remainder
	NEG	Negates an input value
	iOS/fOS	Integer/floating point Offset and Scale
	PRECISIONTIMER	Measure pulse width
	SGN	Returns the sign of a number
	SIN	Returns the sine of an angle
	Sq1int/Sq2Int	Square interpolation
	iSO/fSO	Integer/floating point Scale and Offset
	SQR	Returns the square-root of a number
	TAN	Returns the tangent of an angle
Miscellaneous Functions	BCD2BIN	Converts from BCD to BINary
	BCD2SEG	Converts from BCD to seven-segment LED data
	BIN2BCD	Converts from BINary to BCD
	CAMINIT	Initialises CAM table for position controller
	CRC16	Calculates cyclic redundancy check.
	DELAY	Delay
	DTERM	Differentiator block
	GetSamplesPerSec	Returns the number of samples per second for the current realtime task
	GetTaskID	Returns the the ID for the current task
	ITERM	Integrator block
	CMODEXFER MODEXFER	Drive mode change
	MOVE	Assign an input to an output
	REINIT	Re-initialises target set-up
	RUNTIME	Returns the accumulative time the equipment has been running
	SCAN	Returns the scan sample time of the current task
	SCHEDULEEVENT	Schedule and Event task to execute
	SetUserID	Set the user program ID (#81.49)
	WDOG	Watchdog function

Parameter Access	#mm.pp	General parameter read/write
	#INTmm.pp	Writes to integer value to parameter
	BLKDEF	Block parameter write
	CONDPARW	Conditional Parameter Write
	GetParAttr	Returns the attributes of a drive or option module parameter
	PFIXREAD PFIXREAD6	Fixed Precision Parameter Read (3 decimal places) Fixed Precision Parameter Read (6 decimal places)
	PFIXWRITE PFIXWRITE6	Fixed Precision Parameter Write (3 decimal places) Fixed Precision Parameter Write (6 decimal places)
RAM Files	AssRAM	Associate files with arrays
	RAMLength	Returns the size of a RAM file
	SetRAMLength	Set the length of a RAM file
	UnassRAM	Dissociate files from arrays
Ramps and Position Control Blocks	CAMBOX	CAM generator
	CAMBOX2	CAM generator
	EGB/EGB2	Electronic gearboxes
	MPOT	Motorised pot
	HOME1	Homing function blocks
	HOME2	
	POSLOOP	Position Loop Controllers
	POSLOOP2	
	RAMP	Linear ramp generator
	REG	Registration block
	SLEW	Simple ramp generator
	SRAMP	SRamp profile generator
	SSRAMP	SRamp profile generator
	VEL2POS	Velocity profile integrator
Range and Limit Block	HiLoDet	Detects the the highest and lowest values that an input reaches
	MIN	Returns the minimum of two values

Range and Limit Block	MAX	Returns the maximum of two values
	LIMIT	Limits a value to a given range
	iLIMIT/fLIMIT iLIMIT2/fLIMIT2	Applies minimum and maximum limits to a integer/floating point value
	iWindow	Shows the relationship between two integers within a specified tolerance window
Real-Time Data Functions	AVERAGE	Returns Average of value
	FILTER	First order digital filter
	PID	PID control loop
SmartCard Functions	GetNextSmartCardFile	Returns the details of the next file on the SmartCard
	OpenReadSmartCard	Open SmartCard for reading
	ReadSmartCardByte	Reads a byte if data from the SmartCard
	OpenWriteSmartCard	Open SmartCard for writing
	WriteSmartCardByte	Writes a byte of data to the SmartCard
	CloseSmartCard	Close SmartCard file
	ReadReadOnlyBit	Read the read-only status of the SmartCard
	WriteReadOnlyBit	Set the read-only bit of the SmartCard

Function Block
Control and Access Functions:
APCSetRunMode
APCReset
APCGetOutputSpeed
APCGetOutputSpeedRpmx10
APCSelectAbsoluteMode
APCSelectRelativeMode
APCReadPar
APCSetPositionResetOffset
APCResetSourcesOnDisable
APCDoNotResetSourcesOnDisable
Reference and Feedback Encoder:
APCSetReferenceSource
APCSetFeedbackSource
APCSetNumOfTurnsBits
APCEnableRefSourceMarker
APCDisableRefSourceMarker
APCEnableFbckSourceMarker
APCDisableFbckSourceMarker
APCResetRefSourceMarkerFlag
APCResetFbckSourceMarkerFlag
APCResetRefSourceFreezeFlag
APCResetFbckSourceFreezeFlag
APCInvertRefSource
APCDoNotInvertRefSource
APCInvertFbckSource
APCDoNotInvertFbckSource
APCSetReferencePosition
APCSetFeedbackPosition
APCGetReferenceStatus
APCGetFeedbackStatus

Function Block
References:
APCSelectReference
APCSelectActionOnFreeze
APCSetStopMode
APCSetPositionSetpoint
APCSetSpeedSetPoint
APCCAMInitialise
APCCAMInitialise1
APCCAMInitialise2
APCCAMInitialise3
APCSetCAMStartIndex
APCSetCAMSize
APCSetCAMDeltaSeqLimit
APCSetCAMInterpolationMode
APCSetCAMOutRatioNumerator
APCSetCAMOutRatioDenominator
APCSelectCAMAbsoluteReset
APCSelectCAMZeroReset
APCSetCAMAbsResetIndex
APCSetCAMAbsResetPositionInSeg
APCEnableCAMSingleShot
APCDisableCAMSingleShot
APCSetDigLockMode
APCSetDigLockRatioNumerator
APCSetDigLockRatioDenominator
APCSetDigLockockingSpeed
APCSetDigLockockingPosition
APCSetSpeedOffset
APCSetPositionOffset
APCEnableRigidLock
APCDisableRigidLock

Function Block
Profile Generator:
APCEnableProfile
APCDisableProfile
APCSetProfileAccelRate
APCSetProfileDecelRate
APCSetProfileMaxSpeedClamp
Position Loop:
APCSetPGain
APCSetPGainSpeedClamp
APCEnableExternalRefSpeed
APCDisableExternalRefSpeed
APCSetExternalRefSpeed
APCEnableExternalRefPosition
APCDisableExternalRefPosition
APCSetExternalRefPosition
APCSetOutputRatioNumerator
APCSetOutputRatioDenominator
APCSetupOutputChannel
APCEnableOutputChannel
APCDisableOutputChannel
APCWriteOutputChannel
APC Conversion Functions:
SetUPR
GetUPR
UserToAPCPosition
APCToUserPosition
UserToAPCVelocity
APCToUserVelocity
UserToAPCAcceleration
APCToUserAcceleration

```
Q% = ABS(IN%)
```

Example

```
// Val% is assigned the value 100.
Val% = ABS(100)

// Val% is assigned the value 57 (positive).
Val% = ABS(-57)
```

Target/ Language Restrictions None.

This function returns the result of ANDing its four inputs.

```
Q% = AND4(IN1%, IN2%, IN3%, IN4%)
```

Arguments
IN1% to IN3%
The values to be ANDed together. Only the bottom bit (bit 0) of each input will be ANDed. All other bits in the input will be ignored.
Example

```
// Val% is assigned the value 1.
Val% = AND4(1, 1, 1, 1)

// Val% is assigned the value 0.
Val% = AND4(0, 0, 0, 1)

// Val% is assigned the value 0 because the bottom bit of the 3rd input (value 2)
// is clear.
Val% = AND4(1, 1, 2, 1)
```

Target/ Language Restrictions
None.
See Also
- NAND4 NOR4 OR4 XOR4

Технические характеристики

• Высокоскоростной специализированный микропроцессор
• Флэш-память 384 кбайт для программ пользователя
• Оперативная память 80 кбайт для программ пользователя
• Порт EIA-RS485 с поддержкой протоколов ANSI, Modbus-RTU ведомый и ведущий и Modbus-ASCII ведомый и ведущий
• Интерфейс к высокоскоростной сети CTNet со скоростью до 5 Мбит/с.
• Два цифровых входа 24 В
• Два цифровых выхода 24 В
• Двухпортовой интерфейс ОЗУ для обмена данными с Unidrive SP и другими дополнительными модулями
• Система программирования на базе задач, позволяющая в режиме реального времени управлять приводом и процессом.

Модуль SM-Applications содержит две базы данных параметров:

- База данных Unidrive SP

Эта база содержит полный набор параметров привода. Модуль SM-Applications кэширует эту базу данных в своей собственной энергонезависимой *флэш* памяти. При включении питания модуль проверяет, соответствует ли кэш содержимому привода. Если нет, то база данных загружается с привода, при этом на дисплее привода на несколько секунд появляется слово “Loading”. Такая загрузка не повторится до тех пор, пока модуль SM-Applications не будет переставлен в другой привод с другой версией микропрограммы или не будет обновлена микропрограмма привода.

- База данных модуля SM-Applications

Эта база данных содержит все параметры, хранящиеся локально на модуле, например, регистры ПЛК, а также другие “короткие” параметры (меню 90, 91, и т.д.)

Псевдонимы

Сразу же после заголовка пользователь может разместить раздел *псевдонимов*.
Псевдонимы используются в тексте программы 'вместо' различных выражений или констант, то есть могут заменять:

- численное константное выражение
- адрес регистра или параметра
- выражение или оператор DPL

Псевдонимы определяются с помощью оператора (директивы) $DEFINE.

$DEFINE *имя значение*

Например, хорошим тоном считается использование псевдонимов для именования всех используемых в программе параметров привода.

```
$DEFINE PRESET_REF_1 #1.21
$DEFINE PRESET_REF_2 #1.22
$DEFINE SPEED_FB #3.02
```

Рекомендуется также использовать в именах всех псевдонимов только ЗАГЛАВНЫЕ буквы, чтобы их легко можно было отличить от обычных переменных.

Директива $DEFINE НЕ создает никакого машинного кода и никак не ускоряет выполнение вашей программы - она просто позволяет вам использовать более удобные имена для обозначения некоторых элементов в программе.

Задачи

Программа DPL состоит из отдельных разделов, называемых задачами. Внутри задач пользователь записывает инструкции программы, которые должны выполняться микропроцессором при определенных условиях или через определенные интервалы времени. Каждая задача имеет свое конкретное имя, назначение и приоритет и только одна из каждой задачи может присутствовать в программе DPL.

Имя задачи (Приоритет)

INITIAL (3)

Самая первая задача, которая выполняется после включения питания или после сброса. Эта задача обычно используется для инициализации параметров привода и переменных программы. Ни одна другая задача не может выполняться, пока эта задача не будет завершена.

BACKGROUND (1)

Задача низкого приоритета, используемая для не критичных по времени функций. Способ выполнения этой задачи близко аналогичен циклу сканирования в ПЛК. Обычно эта задача организована в виде одного большого цикла, и в конце задачи стоит инструкция перехода назад на начало. Если эта задача будет завершена, то она больше не будет выполняться.

CLOCK (2)

Задача, выполняемая через фиксированный интервал времени (от 1 до 200 мсек) и используемая в некоторых связанных с временем приложениях, например, для создания профиля рампы. Эта задача теперь синхронизована с циклом управления привода уровня 2.

POS0

POS1 (4)

Две задачи реального времени, которые выполняются через кратное число циклов управления приводом (от 250 мксек до 8 мсек). Обычно эти задачи используются для управления контуром скорости или тока привода в таких приложениях, как позиционирование. Сначала выполняется задача POS0, и сразу после нее выполняется задача POS1.

EVENT (5)

Задачи событий выполняются только при возникновении определенного события. События могут собираться с разных источников, например, из CTNet, других дополнительных модулей в приводе Unidrive SP или из программы пользователя. Задачи EVENT имеют высший приоритет, поэтому они обычно содержат очень малое число инструкций. Их можно связывать в цепочки для прерывания сервисных подпрограмм.

EVENT1 (5) EVENT2 (5) EVENT3 (5)

ERROR (1)

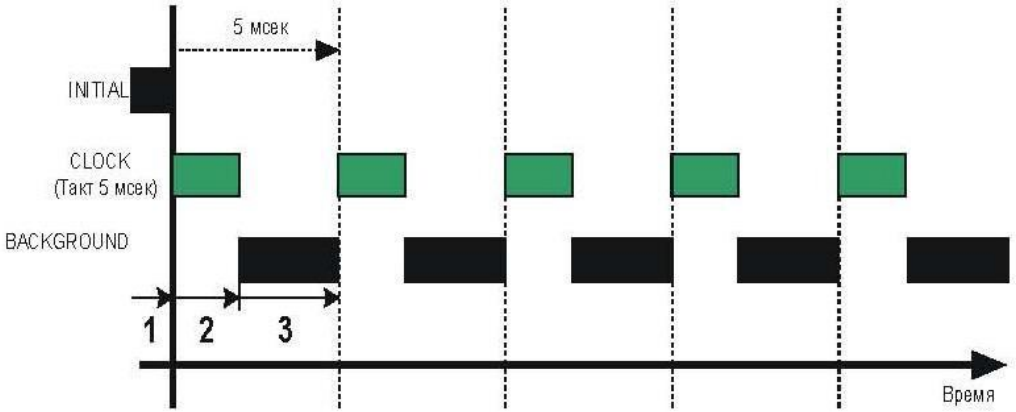
Эта задача выполняется только при возникновении в программе DPL пользователя ошибки времени выполнения (например, деления на нуль). Ее можно использовать для безопасного завершения ненормального выполнения программы. Перед запуском задачи ERROR останавливается всех других задач

При использовании задач CLOCK, POS0 и POS1 рекомендуется не применять в них такого кода, как циклы FOR и DO WHILE. Они могут вызвать в DPL ошибку переполнения (tr54).

Все инструкции программы **должны** располагаться внутри задачи. Для периодически выполняемых задач, например, POS0, POS1 и CLOCK, все инструкции задачи должны быть выполнены в течение определенного времени, поэтому внутри этих задач можно выполнять только критичные по времени функции.

Задачи имеют разные уровни приоритета, поэтому одна задача может прерывать другую задачу. Чем выше номер приоритета в показанной выше таблице, тем выше приоритет этой задачи. Поэтому задача POS0 может прерывать задачу CLOCK, которая может прерывать фоновую задачу BACKGROUND.

Концепция взаимного прерывания задач показана на следующей схеме:

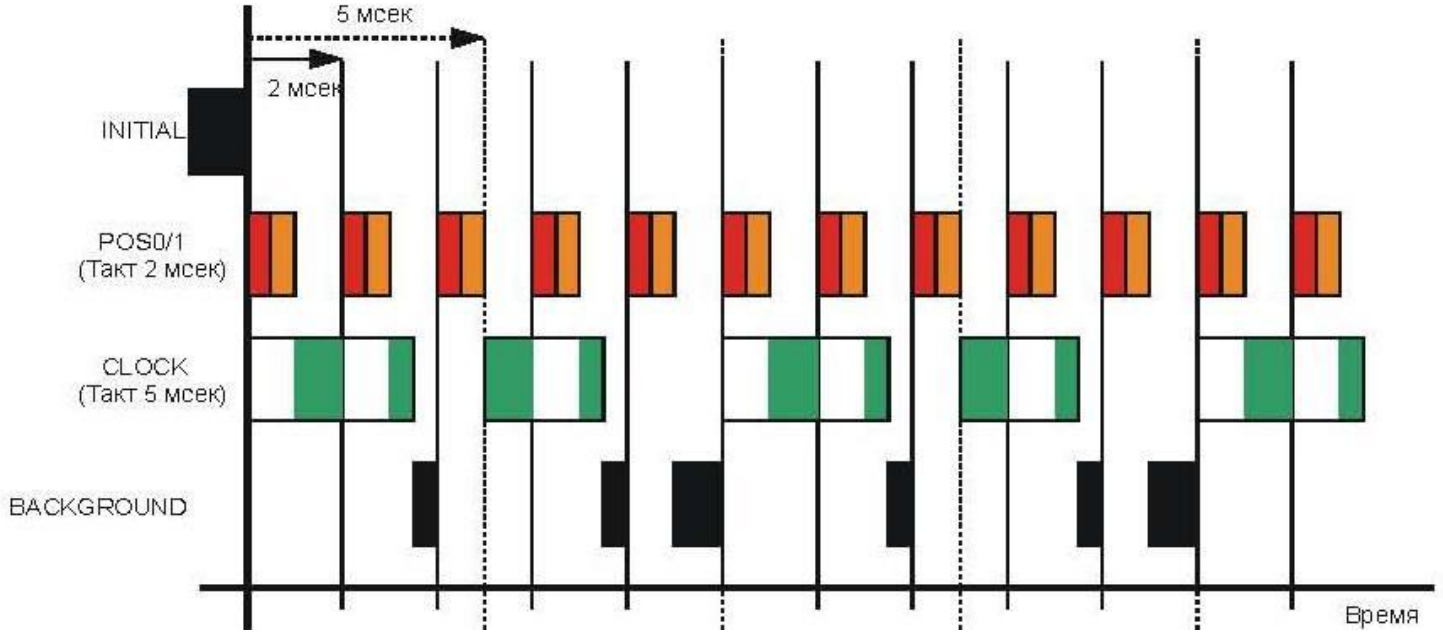


Обозначения:

1. Задача INITIAL имеет исключительное управление ресурсами. Не может выполняться никакая другая задача.
2. Приоритет задачи CLOCK выше, чем у фоновой задачи BACKGROUND.
3. Задача CLOCK завершена и теперь может выполняться задача BACKGROUND - до следующего такта системных часов.

Обратите особое внимание на то, что задача CLOCK запускается периодически через фиксированные интервалы времени (на схеме выше через 5 мсек). Это означает, что все инструкции внутри задачи CLOCK ДОЛЖНЫ быть выполнены менее чем за 5 мсек, иначе задача BACKGROUND не получит промежутка для своего выполнения, либо может произойти отключение по перегрузке процессора.

Здесь видно, что задачи POS0 и POS1 прерывают задачу CLOCK, которая в свою очередь прерывает фоновую задачу BACKGROUND. Как видно, это сильно загруженная программа, так как для выполнения фоновой задачи остается совсем немного времени. Для определения того, насколько велика загрузка модуля SMApplications можно использовать параметр свободного ресурса процессора #81.04.



Переменные

Типы

Имеются три основных типа переменных:

1. Целая переменная
2. Переменная с плавающей запятой двойной точности
3. Переменная с плавающей запятой обычной точности

Целая переменная указывается символом % в конце имени переменной. Если в имени переменной нет символа %, то это переменная формата плавающей запятой.

Тип	Представление	Диапазон
Целый	32-битовая со знаком.	-2147483648 до 2147483647
Одинарный плавающий	32-битовая, 1 бит знака, 8 – экспоненты и 23 - мантиссы.	±3.40282e+038
Двойной плавающий	64 бита: 1 бит знака, 52 бита мантиссы, 11 битов экспоненты	±1.79769e+308

Примеры переменных:

```
speed% = 1234 // целая переменная
value = 55.6 // переменная с плав. запятой
```

Для указания того, какой тип переменных с плавающей запятой - одинарной или двойной точности - используется во всей программе, в начале программы ставится специальный оператор. По умолчанию используются переменные двойной точности. Для использования переменных одинарной точности сразу после секции заголовка программы (с $TITLE и т.п.) надо вставить следующую строку:

```
$flt single
```

Имена переменных

Первым символом имени переменной должна быть буква. Последующие символы могут быть буквами, цифрами, и символом подчеркивания (_).

Имена переменных чувствительные к регистру (например, имена speed%, SPEED% и Speed% относятся к трем разным переменным).

Редакторы SYPT QuickLD и FBD позволяют использовать только имена переменных длиной не более 16 символов, включая символ %.

Инициализация переменных

Перед использованием любой переменной в программе ей нужно присвоить начальное значение. Обычно это делает задача INITIAL. Например,

```
Initial {
speed_sp% = 0
ramp% = 0
}
```

Область и время жизни переменной

Переменные могут быть глобальными или локальными. Все переменные, определенные в программах DPL, являются глобальными, то есть к ним имеет доступ и их может изменять любая задача. Исключением являются переменные в определенном пользователем функциональном блоке, которые являются локальными (то есть к ним нельзя обратиться снаружи определенного пользователем функционального блока).

Ни одна переменная DPL не переживает сброса модуля SM-Applications. Не забывайте, что восстановление привода из состояния отключения также вызывает сброс модуля.

Массивы неизменного размера

Программа DPL может содержать массивы целых или плавающих переменных. Разрешаются массивы только фиксированного размера (одномерные). Массив необходимо сначала объявить с помощью оператора DIM (обычно в задаче Initial), при этом число элементов массива указывается в квадратных скобах после имени массива, например:

```
DIM myarray%[20] // Целый массив из 20 элементов
DIM array2[30] // Массив из 30 плавающих элементов
```

Все элементы массива нумеруются от 0 до число_элементов - 1. В приведенном выше примере первый элемент массива myarray%[] - это:

```
myarray%[0]
```

а последний элемент - это:

```
myarray%[19]
```

Предусмотрены две функции, с помощью которых во время выполнения программы можно определить верхнюю и нижнюю границу индекса массива. Это функции UPPER и LOWER. Для массива myarray%[], UPPER вернет значение 19, а LOWER вернет 0.

Константные массивы

Константные массивы, как следует из их названия, содержат неизменные предопределенные значения. Значения константного массива определяются в программе DPL с помощью специального раздела (смотрите термин CONST в оперативной справочной системе). Можно определять только целые значения. Преимущество константного массива заключается в том, что размер массива ограничивается только размером памяти программы, а не ОЗУ переменных. Размер памяти программы составляет 384 кбайт, он используется для хранения скомпилированного файла DPL, данных константых массивов и опционно самого файла DPL.

Память для хранения - число переменных

Все переменные, динамические массивы и регистры ПЛК располагаются в области памяти размером 80 кбайт. Каждая целая переменная и переменная с плавающей запятой одинарной точности занимает 4 байта (32 бита), а переменная с плавающей запятой двойной точности занимает 8 байт (64 бита). Кроме того, в памяти размещаются другие элементы, например, индексы для параметров. Компилятор DPL оповестит вас, если вы достигнете предела имеющейся памяти.

Адресация битов в переменных

Все целые переменные и массивы можно адресовать побитно. Это означает, что внутри переменной можно отдельно читать или записывать каждый отдельный бит. Для доступа к битам после имени переменной поставьте точку (.) и затем номер бита от 0 до 31.

Пример 1 (простая переменная):

```
flags% = 0 // инициализируем все 32 бита в 0
flags%.0 = 1 // настроим бит 0 в 1

// теперь проверим, настроены ли бит 0 И бит 1 в значение 1.
IF flags%.0 & flags%.1 = 1 THEN
    PRINT "Условие выполнено."
ENDIF
```

Пример 2 (Массивы)

```
DIM myarray%[10]              Номер бита должен быть константой - переменные не допускаются.
...
IF myarray%.1[4] = 1 THEN;проверим бит 1 элемента #4.
   PRINT "Условие выполнено."
ENDIF
```

Регистры ПЛК

Область “ПЛК” - это специальный набор предопределенных 32-битных регистров. Регистры ПЛК делятся на 6 наборов из 100 параметров, пронумерованных от 00 до 99. К регистрам можно также обратиться из программы DPL пользователя с помощью специального имени переменной или имени массива. Четыре из наборов регистров можно сохранять во флэш-памяти модуля SM-Applications.

Файлы ОЗУ

Файлы ОЗУ позволяют пользователю сохранить “файлы” в ОЗУ пользователя в модуле SM-Applications. Их можно загрузить и выгрузить с помощью команд DPL. Их преимущество заключается в том, что за одну операцию вы можете вызвать или записать массив чисел, а не отдельный элемент массива.

Параметры

Параметры можно разделить на две категории:

- Параметры привода
- Параметры SM-Applications

Параметры привода размещаются в ведущем приводе. Большинство из них влияет на работу привода, однако некоторые зарезервированы как “прикладные программы”. Это параметры меню 18, 19 и 20. Параметры SM-Applications являются локальными и доступны только внутри модуля SM-Applications. Эти параметры предоставляют доступ к дополнительным функциям модуля SM-Applications и обеспечивают более быстрый доступ к некоторым параметрам привода.

Модуль SM-Applications гарантирует, что используемая им база данных параметров привода всегда совпадает с этой базой в ведущем приводе Unidrive SP. Когда модуль SM-Applications только установлен в Unidrive SP и в первый раз включается питание, на дисплее привода может на несколько секунд появиться слово “Loading”. Это указывает процесс синхронизации баз данных в модуле SM-Applications. Такая синхронизация выполняется только при первой установке модуля в привод. При последующих включениях питания слово “Loading” появляется только кратковременно.

Чтение и запись параметров

Чтение и запись параметров выполняется с помощью команды #. Доступ к параметрам ведется в формате #MM.PP точно так же, как с клавиатуры привода.Например, для чтения параметра обратной связи по скорости (параметр 03.02),используйте: speed% = #3.02

для записи параметра задания скорости (например, 01.22), используйте: #01.22 = 1500

Обратите внимание, что передний нуль в поле меню/параметра можно опускать. Например, все конструкции #3.02, #03.02, #03.2 и #3.2 обращаются к одному параметру.

Целые параметры с фиксированной запятой

Работа с целыми параметрами с фиксированной запятой выполняется медленнее, чем с целыми параметрами. Для ускорения при записи и чтении параметров можно использовать специальную команду #INT. При использовании этой команды для целых с фиксированной запятой десятичные позиции будут автоматически удаляться.

Например, параметр #1.19 имеет диапазон значений 0.000 - 0.099. При чтении этого параметра с помощью:

speed_fine% = #INT1.19

будут возвращены целые значения от 0 до 99. При записи команда: #INT1.19 = 45

настроит параметр в значение 0.045 (как команда #1.19=0.045). Преимуществом этого является то, что программа DPL может использовать целые переменные (%) вместо плавающих, что ускоряет работу.

Операторы

DPL предоставляет все стандартные операторы:

Стандартные операторы в порядке очередности выполнения

Оператор	Значение
-	Арифметическое изменение знака
!	Логическое отрицание (унарная)
!(..., nbit)	Отрицание битов *nbit*
*	Умножение
/	Деление
%	Модуль (остаток)
+	Сложение
-	Вычитание
&	Побитовое И
|	Побитовое ИЛИ
^	Побитовое исключающее ИЛИ (XOR)

Условные операторы в порядке очередности выполнения

Оператор	Значение
=	Равенство
<	Меньше чем
>	Больше чем
<=	Меньше или равно
>=	Больше или равно
<>	Неравенство
AND	Логическое И
OR	Логическое ИЛИ
NOT	Логическое НЕТ

Основные команды DPL

Цикл FOR

```
FOR переменная = целое_выражение to целое_выражение [STEP константа]
операторы
LOOP
```

CASE

```
SELECT целое_выражение
        CASE целая_константа
                операторы
        [CASE целая_константа, целая_константа ...
                [операторы]]
        [ELSE
                [операторы]]
ENDSELECT
```

Эта конструкция позволяет удобно проверить несколько константных значений. Можно включать любое число операторов CASE.