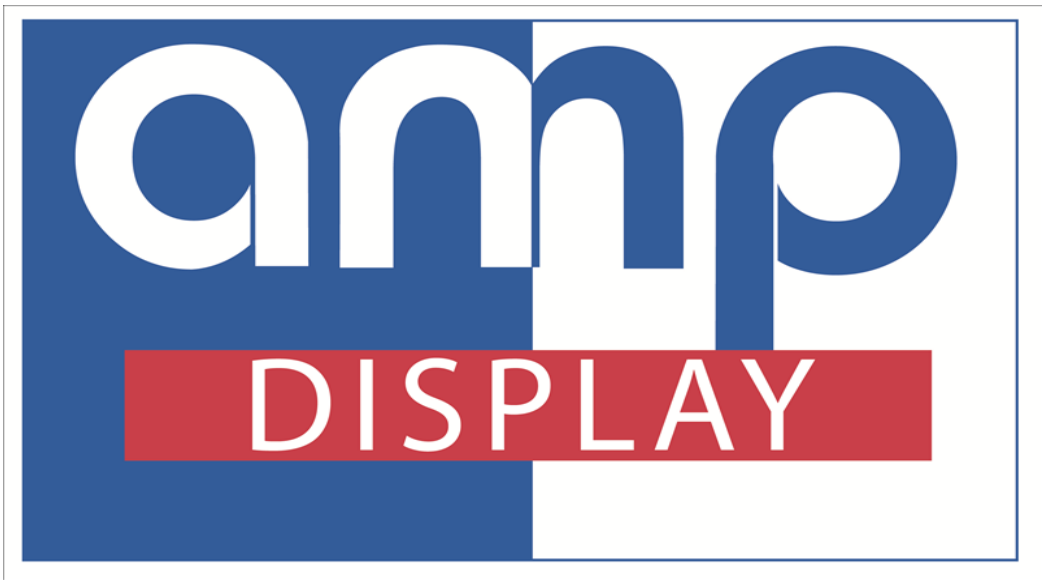

DWIN_OS

Development Reference

V2.2

Revision on May,2013

Amp Display Inc.

DWIN_OS Development Reference

1 How to use DWIN_OS:

User can use "DWIN_OS_Builder" program to write codes, then it can be compiled into 23.bin file. Please use SD card to download it to the DGUS LCMs Flash memory, and then enable the DWIN_OS in the config.txt for DGUS system by adding following parameters at the end of the content:

RUN_DWIN_OS to enable DWIN_OS; or make RC.6=1

STOP_DWIN_OS to disable DWIN_OS; or make RC.6=0

2 Instructions

2.1 Instruction format

In DWIN_OS, it can run up to 32674 instructions. Each instruction will contain 4 bytes:

Address (Byte)	0x00	0x01-0x03
Length (Byte)	1	3
Definition	Commands	Parameters

2.2 Definitions

Program Pointer Variables:

Used to indicate the address of each instruction, the range is 0x0000-0x7FFB; accordingly, 0x00000-0x1FFEC in physical address is used for code storage.

OS Registers:

From R0 to R255, each Register holds 1 byte of data, 256 Bytes in total.

DGUS Registers:

Correspond to the Register space in DGUS system accessed by 0x80/81 command. (0x00 – 0xFF)

DGUS Variables:

Correspond to the Variables in DGUS accessed by 0x82/83 command.

(0x0000 - 0x6FFF)

Font Library Space:

Correspond to the Font Library 32-127 (0x20 – 0x7F), 24MB in total, can be imported from or exported to SD card.

2.3 Execution pattern:

The whole program will be executed ONCE within single DGUS time cycle (refer to the setting for DGUS, could be 80/120/160/200ms).

User program and DGUS software exchange data by variables in parallel-processing.

Unless DGUS LCMs is power-off or reset, register and Variables will hold the data when the program is running.

2.4 Pseudo Assembly Code:

EQU: substitute in compiling

Example:

```
PICID   EQU 3
WORD    EQU 2
MOVDR   PICID, R10, WORD        //   Equal to "MOVDR 3, R10, 2"
```

Tips： EQU definition can be found from Macro Define in OS software

DB: Define a BYTE or a WORD

Example:

```
LDADR   TAB1     //   Save the data in TAB1 (24bits) to R5, R6, R7
TAB1:   DB  1,2,3,4
        DB  1000, 2000, 3000, 4000, -100
        DB  "A Sample String"
```

"Symbol ;": For comments only.

3. Instructions

R# stands for the single/multiple Registers with certain index. R0-R255.

<> present Instant Numbers. In assembly, 100, 0x64, 64H, 064H are as the same as 100in decimal.

Functions	Commands CMDs	Parameters	Remark
None	NOP		A Non-Operation. NOP
Data exchange between DGUS Variables & DWIN_OS Registers	MOVXR	R#, <MOD>, <NUM>	R#: DWIN_OS Register(s). <MOD>: 0: Register to Variable; 1: Variable to Register <NUM>: Data length in Words for exchange: 0x00-0x80 When <NUM>=0x00; Length is determined by R9. DGUS Variable Pointer defined by R0, R1. *e.g.: **MOVXR R20, 0, 2***
Load several 8 bit numbers to DWIN_OS Registers	LDBR	R#, <DATA>, <NUM>	R#: DWIN_OS Register(s). <DATA>: Data to load <NUM>: The indexes of Registers to hold the data, 0x00 present 256 Registers. *e.g.: **LDBR R8, 0x82, 3***
Load a 16 bit number to DWIN_OS Register	LDWR	R#, <DATA>	R#: DWIN_OS Registers. <Data>: Loaded data. *e.g.: **LDWR R8, 1000*** ***LDWR R8,-300***
Look up in Program Space (Program Space to DWIN_OS Registers)	MOVC	R#, <NUM>	R#: DWIN_OS Register(s). <NUM>: Data length of result for look up Table address pointer is defined by R5, R6, R7 *e.g.: **MOVC R20, 10***

Data Exchange between Registers in DWIN_OS	MOV	R#S, R#T, <NUM>	R#S: Source Register(s) OR registers R#T: Target Register(s) OR registers <NUM>: Data length for Exchanging. 0x00 indicates the length determined by R9. *e.g.: **MOV R8, R20, 3***
Data transfer from DWIN_OS Register to DGUS Register	MOVRD	R#, D#, <NUM>	R#: DWIN_OS Register(s). D#: DGUS Registers(s). <NUM>: Data length for Exchanging. *e.g.: **MOVRD R10, 3, 2***
Data transfer from DGUS Register to DWIN_OS Register	MOVDR	D#, R#, <NUM>	R#: DWIN_OS Register(s). D#: DGUS Registers(s). <NUM>: Data length for Exchanging. *e.g.: **MOVDR 3, R10, 2***
Data Exchange between DGUS Variables and Font Library	MOVXL	<MOD>, <NUM>	<MOD>:0= Transfer from Font Lib to DGUS Variables 1= DGUS Variable to Font Library <NUM>: Data length (Word) Address of DGUS Variable is defined by R0:R1 Font index is defined by R4 (0x20 – 0x7F), R5:R6:R7 is the operation starting address in Font Lib. Cancelled if out of boundary. *e.g.: **MOVXL 0, 300***
Data exchange between DGUS Variables	MOVXX	<NUM>	<NUM>: Data length (Word) If <NUM> is 0. it means the length to be defined by R8:R9. Address for DGUS Source Variable is R0:R1; Address for DGUS Target Variable is R2:R3 *e.g.: **MOVXX 100***
Registers Indexed Addressing	MOVA		R2 defines the address for source Register(s) R3 defines the address for target Register(s) R9 defines the length for exchange, in BYTES. *e.g.: **MOVA***
32bit integers addition	ADD	R#A, R#B, R#C	C=A+B, A,B are 32bit integers, C is 64bit integer. *e.g.: **ADD R10, R20, R30***
32bit integers subtraction	SUB	R#A, R#B, R#C	C=A-B, A,B are 32bit integers, C is 64bit integer. *e.g.: **SUB R10, R20, R30***
64bit MAC for long integers	MAC	R#A, R#B, R#C	C=(A*B+C), A, B are 32bit integers, C is 64bit integer. *e.g.: **MAC R10, R20, R30***
64bit integers division	DIV	R#A, R#B, <MOD>	A/B, A is quotient, B is reminder. A and B are 64bit register. <MOD>: 0: The quotient will not be rounded. 1: The quotient WILL BE ROUNDED. *e.g.: **DIV R10, R20, 1***
Expand Variable to 32bit	EXP	R#S, R#T, <MOD>	Expand the data in R#S to 32bit and save to R#T R#S: Source register(s)

			R#T: Target register <MOD>: Data type of R#S. 0=8Bit unsigned；1=8bit signed 2=16bit unsigned；3=16bit integer *e.g.:* ***EXP R10, R20, 2***
32bit unsigned MAC	SMAC	R#A, R#B, R#C	C=A*B+C A and B are 16bit unsigned integer, C is 32bit unsigned integer. *e.g.:* ***SMAC R10, R20, R30***
Register self-increase	INC	R#, <MOD>, <NUM>	R#=R#+NUM, unsigned self-increasing calculation <MOD>: Data type of R#; 0=8bit;1=16bit *e.g.:* ***INC R10, 1,5***
Register self-decrease	DEC	R#, <MOD>, <NUM>	R#=R#-NUM, unsigned self-decreasing calculation <MOD>: Data type of R#; 0=8bit;1=16bit *e.g.:* ***DEC R10, 0,1***
Load Address	LDADR	<ADRH>, <ADRM>, <ADRL>	Load <ADRH:ADRM:ADRL> to R5:R6:R7 *e.g.:* ***LDADR TAB*** ***LDADR 0x123456***
Logical Calculation: AND	AND	R#A, R#B, <NUM>	A=A AND B, Logical “AND” calculation for series of Registers. <NUM>: Data length of R#A, R#B in BYTES *e.g.:* ***AND R10, R20,1***
Logical Calculation: OR	OR	R#A, R#B, <NUM>	A=A OR B, Logical “OR” calculation for series of Registers. <NUM>: Data length of R#A, R#B in BYTES *e.g.:* ***OR R10, R20,1***
Logical Calculation: XOR	XOR	R#A, R#B, <NUM>	A=A XOR B, Logical “XOR” calculation for series of Registers. <NUM>: Data length of R#A, R#B in BYTES *e.g.:* ***XOR R10, R20,1***
Integer Linear Equation	ROOTLE	00, 00, 00	Calculate the Y value according to the given X value, which is a point on the line defined by (X_0, Y_0) and (X_1, Y_1) in 16bit integer. Input: X=R10, X_0=R14, Y_0=R16, X_1=R18, Y_1=R1A Output: Y=R12 *e.g.:* ***ROOTLE***
ANSI CRC-16	CRCA	R#S, R#T, R#N	Perform ANSI CRC-16 calculation on series of Registers. ANSI CRC-16(X16+X15+X2+1) R#S: Registers for Input R#T: Registers to hold the result, 16bit, LSB mode. R#N: Save the length for CRC byte data, 8bit *e.g.:* ***CRCA R10, R80, R9***
CCITT CRC-16	CRCC	R#S, R#T, R#N	Perform CCITT CRC-16 calculation on series of Registers. CCITT CRC-16(X16+X12+X5+1)

			R#S: Registers for Input R#T: Registers to hold the result, 16bit, MSB mode. R#N: Save the length for CRC byte data, 8bit *e.g.:* ***CRCC R10, R80, R9***
Read MODBUS data frame from COM_0_Rx_FIFO	RMODBUS	R#A, R#T, R#C	Check the FIFO in COM_0 received valid MODBUS data frame, if yes, will move the data to register and clear the Receiving FIFO. R#A: Specified Registers will store the first 3 bytes of MODBUS data pack (Address, CMDs, and Data Length). ***If length is 0x00, it present no length matching, data after it (the 4th byte) indicate the length exclude address, instruction and CheckSum.*** R#C: Register for return value/status. It will hold the data returned; 0x00 indicates no valid MODBUS data frame is received; 0xFF stands for valid MODBUS data frame is received and stored in R#T registers. R#T: Target register to store the MODBUS data after validation is successful. *e.g.:* ***RMODBUS R10, R20, R13***
Bit decomposition	BITS	R#, <VP>	Decompose the 8 bits in R# to 8 DGUS Variable (Byte) specified by VP. Bit 1 becomes 0x0001, bit 0 becomes 0x0000. R#: The register need to decompose. <VP>: DGUS VP address. *e.g.:* ***BITS R10, 0x2000***
Bit integration	BITI	R#, <VP>	Integrate 8 DGUS Variable (Byte) specified by VP into 1 byte Bit Variable (MSB). 0x0000 becomes bit 0, other value become bit 1. R#: The register need to decompose. <VP>: DGUS VP address. *e.g.:* ***BITI R10, 0x2000***
HEX to ASC	HEXASC	R#S, R#T, <MOD>	R#S: 32bit Integer needs to be converted to ASCII R#T: Target registers for ASCII after conversion. <MOD>: Convert Mode. High 4 bits indicate the length of integers; Lower 4 bits indicates the length of decimals. The ASCII string after conversion is signed, right aligned; empty slots will be filled by 0x20. For data 0x12345678: <MOD>=0x62, result is +054198.96; <MOD>=0xF2, result is +3054198.96 *e.g.:* ***HEXASC R20, R30, 0x62***
Sequence comparison	TESTS	R#A, R#B, <NUM>	Compare the values in R#A and R#B by sequence. If not match, return the current address of R#A to R0

			register; If match, return 0x00 to R0 register. R#A: Starting register for register series A; R#B: Starting register for register series B; <NUM>: max length for data comparison. *e.g.:* ***TESTS R10, R20, 16***
Configuration for COM_1	COMSET	<MODE>, <BSH>, <BSL>	Set the configuration for Serial port COM_1: <MODE>: 0x00=N81 mode; 0x01=E81 mode; 0x02=O81 mode; 0x03=N82 mode; <BSH:L>: Factors for Baud Rate. Value is 6250000/(Desired Baud Rate). The receive FIFO will be cleared when you set the baud rate. *e.g.:* ***COMSET 0, 54***
Conditional bitjump	JB	R#, <BIT>, <NUM>	Evaluate the <bit> in R# register. If 1, jump to <NUM>; if 0, proceed to next instruction. R#: The register contains data to be evaluated. <Bit>: the index of the bit to be evaluated. 0x00-0x0F (MSB). <NUM>: Jump position control. Num.7 control the direction: 1 = forward; 0=backward; result for NUM&0x7F indicates the number of instructions to jump. *e.g.:* ***JB R10, 15,TEST1*** ***NOP*** ***TEST1: ADD R8,R12, R16***
Variable conditional jump (Not Equal)	CJNE	R#A, R#B, <NUM>	Compare the value of 2 8bit registers (R#A and R#B). If equal, proceed to next instruction; if not equal, jump to <NUM>. *e.g.:* ***TEST1: NOP*** ***INC R10, 0, 1*** ***CJNE R10,R11, TEST1***
Integer conditional jump (Less than)	JS	R#A,R#B, <NUM>	Compare the value for 2 bit integer in R#A and R#B. If A>=B, proceed to next instruction; If A<B, jump to <NUM> *e.g.:* ***JS R10, R12, TEST1*** ***NOP*** ***TEST1: NOP***
Value conditional jump (Number and Variable)	IJNE	R#, <INST>, <NUM>	Compare the value in 8 bit Register and a instant Number <INST>. If equal, process to next instruction; if not equal, jump to <NUM>. *e.g.:* ***IJNE R10, R100, TEST1*** ***NOP***

			TEST1: NOP
Compulsorily terminate current input thread	EXIT	R#A, R#B, 00	Compulsorily terminate current input thread. R#A: decide to change page. 0x00= Don't change; 0x01= change page. R#B: The Picture ID to return back (16bit) *e.g.:* ***EXIT R10, R11***
Return	RET	00, 00, 00	Return to main program by calling this function in sub-program. *e.g.:* ***RET***
Call sub-function	CALL	<PCH>, <PCL>, 00	Call sub-program in position of program counter <PCH:L> (0x0000-0x7FFB). Maximum support 32 levels of Program Nesting. *e.g.:* ***CALL TEST***
Direct Jump	GOTO	<MOD>, <PCH>, <PCL>	<MOD>=0x00: Jump to <PCH:L> <MOD>=0x01: Relatively Jump to (PC+1+<PCH:L>) <MOD>=0x02: Relatively Jump to (PC+1-<PCH:L>) <PCH:L>=0xFFFF indicates the value is in R0:R1 *e.g.:* ***GOTO TEST1*** ***NOP*** ***TEST1: NOP***
Data send by Serial port	COMTXD	<COM>, R#S, R#N	Send data to the specified serial port. <COM>: Serial port select. 0=COM1(DGUS User port); 1=COM2(System reserved) R#S: the registers hold the data to send R#N: The registers contained the byte length info to send. If 0x00 indicates sending 256 bytes of data. *e.g.:* ***COMTXD 0, R10, R9***
Print via serial port	CPRTS	<COM>, <VPH>, <VPL>	Check the content for print is exist at the DGUS Variables which <VP> pointed to, if yes, print it via serial port. <VP> correspond to the VP value defined in 0xFE07 in DGUS LCMs. Printing Status will be cleared after printing is done. <COM>: Serial port select. 0=COM1(DGUS User port); 1=COM2(System reserved) *e.g.:* ***CPRTS 0, 0x2000***
Check COM_0_Rx_FIFO	RDXLEN	00, R#, 00	Return the length in byte (0-253) for received data in FIFO buffer are for COM_1 and save it in R# register. 0x00 indicates empty. *e.g.:* ***RDXLEN 0, R10***
Read from COM_0_RX_FIFO	RDXDAT	00, R#A, R#B	Read <R#B> (1-253) bytes from FIFO buffer of COM1 and move it to R#A registers. FIFO's length will adjust

			automatically. *e.g.:* ***RDXDAT 0, R11, R10***
Direct send data via Serial-port	COMTXI	00, R#, <NUM>	Send the data inside <NUM> series of Registers indexed with R#. *e.g.:* ***COMTXI 0, R20, 16***
Read the content for current Input Method	SCAN	R#, <NUM>, 00	Load <Num> of character that inputted under current input method to register at (R#+1), and R# will hold the length info. Characters are counted backward from the current cursor. *e.g.:* ***SCAN R20, 6***
Write Curve buffer for specified channel	WRLINE	R#S, R#1, <CH>	Calculate the result of numbers of 16bit unsigned integers by R#S added an offset value V_BIAS then write it to the buffer for curve defined by <CH> (0x00-0x07). R#I indicate the register hold 3 bytes, N, V_BIAS. *e.g.:* ***WRLINE R80, R10, 2***
Erase specified Font Lib	ERASE	<L_ID>, 5A, A5	<L_ID>: The Font Library ID to be erased. From 0x20-0x7f *e.g.:* ***ERASE 40***
Sum of addition (Error detection)	SUMADD	R#S, R#T, R#N	Calculate the sum of data in 1 byte for error detection. R#S: Registers to calculate (Input) R#T: Result in 1 byte, 8 bit. R#N: Register for length of series. 8 bit. *e.g.:* ***SUMADD R10, R80, R9***
Sum of addition with Carry (Error detection)	SUMADDC	R#S, R#T, R#N	Calculate the sum of data in 1 byte with carry for error detection. R#S: Registers to calculate (Input) R#T: Result in 1 byte, 8 bit. R#N: Register for length of series. 8 bit. *e.g.:* ***SUMADDC R10, R80, R9***
Sum of XOR calculation (Error detection)	SUMXOR	R#S, R#T, R#N	Calculate the result for XOR operation for data in 1 byte for error detection. R#S: Registers to calculate (Input) R#T: Result in 1 byte, 8 bit. R#N: Register for length of series. 8 bit. *e.g.:* ***SUMXOR R10, R80, R9***
Convert HEX to Compressed BCD code	HEXBCD	R#S, R#T, <MOD>	Convert data in HEX to compressed BCD code. 0x1000 will be converted to 0x10, 0x00. R#S: Initial address for registers stored data in HEX R#T: Starting address for registers stored data for result in BCD code. <MOD>: High 4 bits indicate the numbers of byte for HEX data. (0x01-0x08); Low 4 bits indicate the numbers of byte for BCDoutput. (0x01-0x0A). *e.g.:* ***HEXBCD R10, R80, 0x23***

Convert Compressed BCD to HEX code	BCDHEX	R#S, R#T, <MOD>	Convert data in compressed BCD to HEX. 0x1000 will be converted to 0x3E8 (1000). R#S: Initial address for registers stored data in compressed BCD code. R#T: Starting address for registers stored data for result in HEX. <MOD>: High 4 bits indicate the numbers of byte for compressed BCD data. (0x01-0x0A); Low 4 bits indicate the numbers of byte for HEX output. (0x01-0x08). *e.g.:* ***BCDHEX R10, R80, 0x32***
Convert ASCII string to HEX characters	ASCHEX	R#S, R#T, <LEN>	Convert ASCII String to signed 64 bit HEX data. R#S: Starting address for registers stored ASCII Strings R#T: A 64bits register to hold the output 64bit Hex data. <LEN>: The length for ASCII string, include sign bit and decimal point. 0x01-0x15. *e.g.:* ***ASCHEX R10, R80, 0x05***
Read DL/T645 data frame from COM_0_Rx_FIFO	RD645	R#A, R#T, R#C	Check the FIFO in COM_0 received valid DL/T645 data frame, if yes, will move the data to register and clear the Receiving FIFO. R#A: Address Register stores 6 bytes of data. R#C: Register for return value/status. It will hold the data returned; 0x00 indicates no valid DL/T645 data frame is received; 0xFF stands for valid DL/T645 data frame is received and stored in R#T registers. R#T: Target registers to store the DL/T645 data after validation is successful in format: *CMDCode + DataLength + data* *e.g.:* ***RMODBUS R10, R20, R13***
Time sequence funcion	TIME	R#A,R#B,<MOD>	R#A and R#B：register for saving 6bytes time variables which format is BCD; MOD=0, calculating A=A-B，to count relative value between two time data. A MUST BE greater than B, when A<B, 0xEF which is the first word of R#A was returned automatically without calculation MOD=1，compute A=B-RTC； MOD=2，compute A=RTC-B。 e***.g.: TIME R0,R10,0***
Add display variables	ADDL14	R#A,R#B,<MOD>	R#A: register for saving one display variable(32Bytes); R#B: site position that added for variables, 0x00-0x1F, maximum 32 pcs of variables can be added. <MOD>: 0x5A= Add to designated position Other=Delete designated position and null for R#A at

			this point. e.g.: ***ADDL14 R80,R81,0x5A***
Square root Computing	SQRT	R#A,R#B	Compute a square root of 64-bit unsigned number R#A and save to R#B R#A: Saved a 8 byte unsigned number; R#B: Saved a 4 byte unsigned number e.g.: ***SQRT R80,R90***
End of the program	END	FF, FF, FF	*e.g.:* ***END***